Tue Herlau, Mikkel N. Schmidt and Morten Mørup

Introduction to Machine Learning and Data Mining

Course notes fall 2016, version 13.

October 22, 2016

Technical University of Denmark

Notation cheat sheet

	Matlab var.	Type	Size	Description
	`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.
Regression	`y`	Numeric	$N \times 1$	Dependent variable (output): For each data object, `y` contains an output value that we wish to predict.
Classification	`y`	Numeric	$N \times 1$	Class index: For each data object, `y` contains a class index, $\texttt{y}_n \in \{0, 1, \ldots, C-1\}$, where C is the total number of classes.
	`classNames`	Cell array	$C \times 1$	Class names: Name (string) for each of the C classes.
	`C`	Numeric	Scalar	Number of classes.
Cross-validation				All variables mentioned above appended with `_train` or `_test` represent the corresponding variable for the training or test set.
	`⋆_train`	—	—	Training data.
	`⋆_test`	—	—	Test data.

The emphasis of this book is to convey the underlying machine-learning concepts rather than mathematical rigor. However it is impossible to discuss machine learning without making use of basic tools from linear algebra, probability theory and analysis. In the following, vectors will be denoted by lower-case roman letters $\boldsymbol{x}, \boldsymbol{y}, \cdots$ and matrices by bolder, upper case roman letters $\boldsymbol{A}, \boldsymbol{B}, \cdots$. A superscript T denote the transpose. For instance

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

The ith 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 $A_{2,3} = -2$. 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 $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N$ then any given observation will consist of M numbers:

$$\boldsymbol{x} = \begin{bmatrix} x_1 & \ldots, x_M \end{bmatrix}^T.$$

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

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

in which the ith 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*. Often each of the observations $\boldsymbol{x}_i$ will come with a *label* or *target* y_i corresponding to a feature of $\boldsymbol{x}_i$ which we are interested in predicting. In this case we will collect the labels in a N-dimensional vector $\boldsymbol{y}$ and the pair $(\boldsymbol{X}, \boldsymbol{y})$ will be all the data available for the machine learning method.

Contents

Part I

Data: Types, Features and Visualization

1

Introduction

In this section, 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 an intelligent machine? More than 65 years ago Alan Turing made this question the subject of his famous essay *"Computing machinery and intelligence"* [Turing, 1950]. Alan Turing proposed that if we phrased the question in this manner we would 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 search by defining intelligence. A second part of Turing's essay treats *how* we should build a human-imitating machine. Turing proposed that instead of writing a computer program that behave 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 translate from English to French, we should construct a machine which initially could not translate at all but after observing many examples of translated sentences could *learn* how to translate, 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 to improve at some task in the future based on the past experience and this will simply be called *learning*, damn any philosophical issues with that word. The focus of machine learning is on automatic 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 is using computers to discover patterns or relationships in datasets and to transform this into an understandable 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 system 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 7 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 9. This will lead to better and more trustworthy predictions and, more importantly, it will save Susi a lot of time.

The amount of data that is readily available in any given domain is growing at a rapid rate and in nearly any domain. 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, of-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:

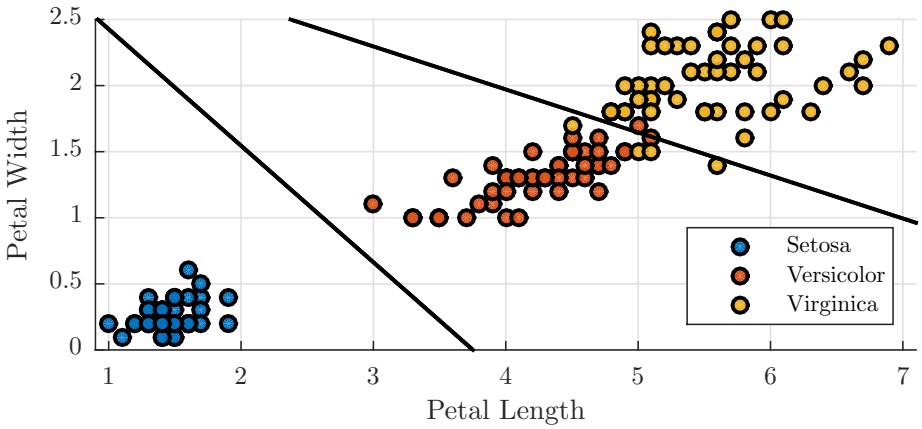


Fig. 1.1: A classification problem where we are given observations (the points) and class labels (the colors) and the goal is to come up with a rule for determining which class a point belongs to (one such rule is indicated by the lines). The rule can then be applied to new points.

Classification

In classification we are given observed values $\boldsymbol{x}$ 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).
- 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 $\boldsymbol{x}$ 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).

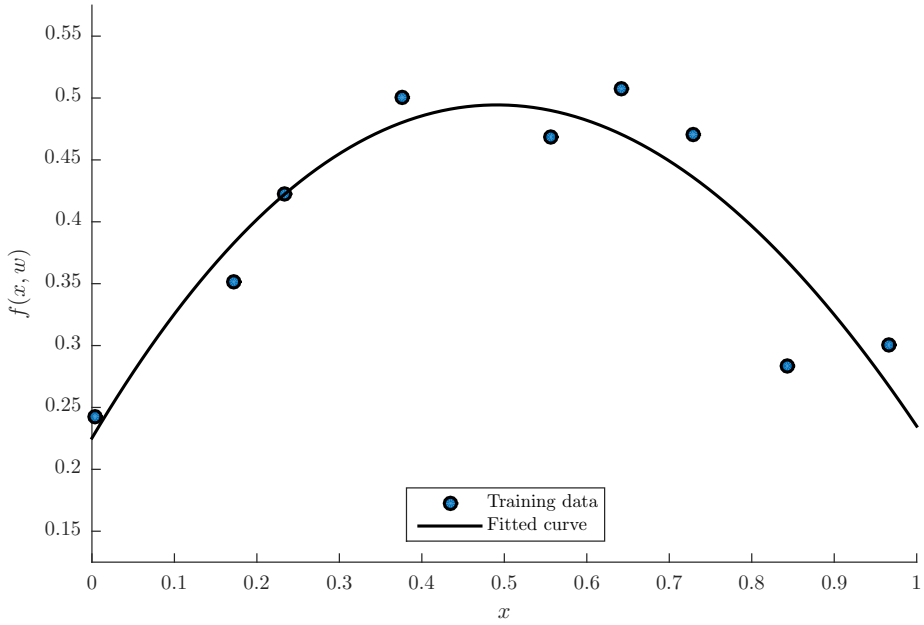


Fig. 1.2: A one-dimensional regression problem where we have to predict the y-values based on the x-values. The fitted regression model is indicated by the black line.

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

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.

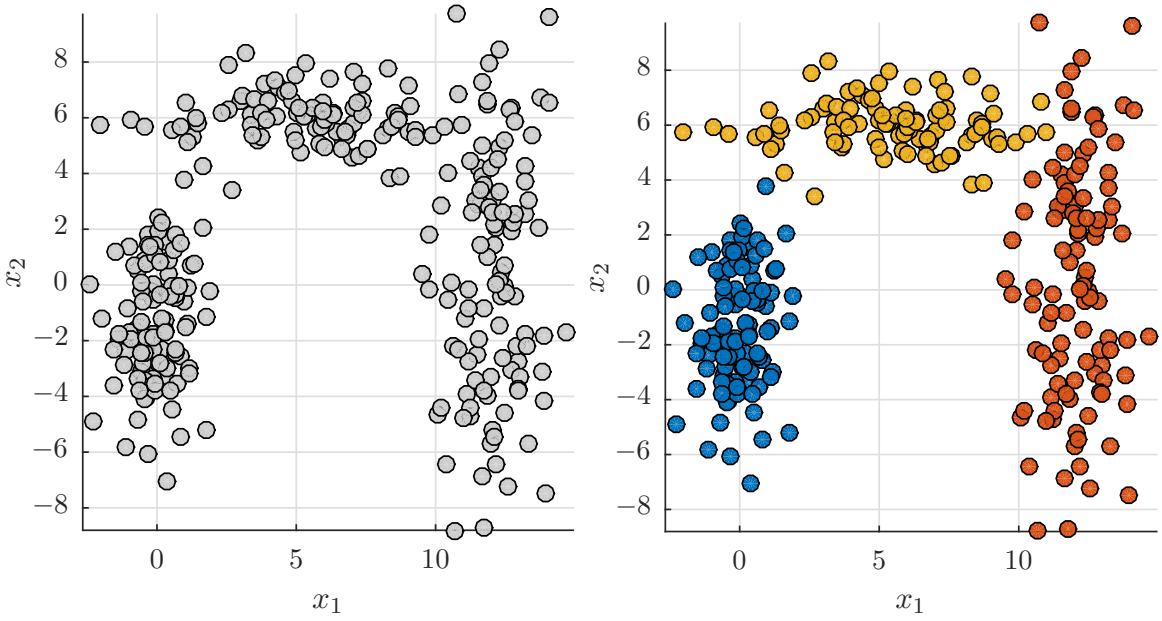


Fig. 1.3: A 2D clustering example. A clustering is given a dataset (here the 2D dataset shown in the left-hand pane) and has to estimate plausible divisions of the observations into clusters as indicated in the right-hand pane.

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 in fig. 1.5 to be significanly different from the other observations from a number of perspectives. This includes that it is simply quite far away from it's 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 an 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

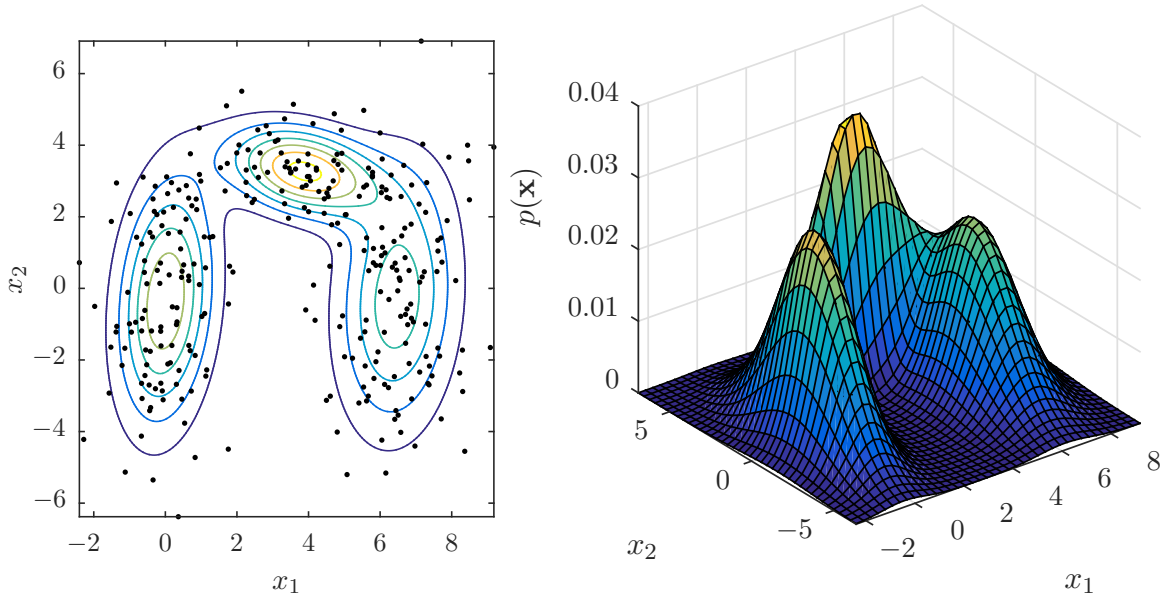


Fig. 1.4: A 2D density estimation example. Given the black points, the density is estimated and plotted in the right-hand pane.

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:

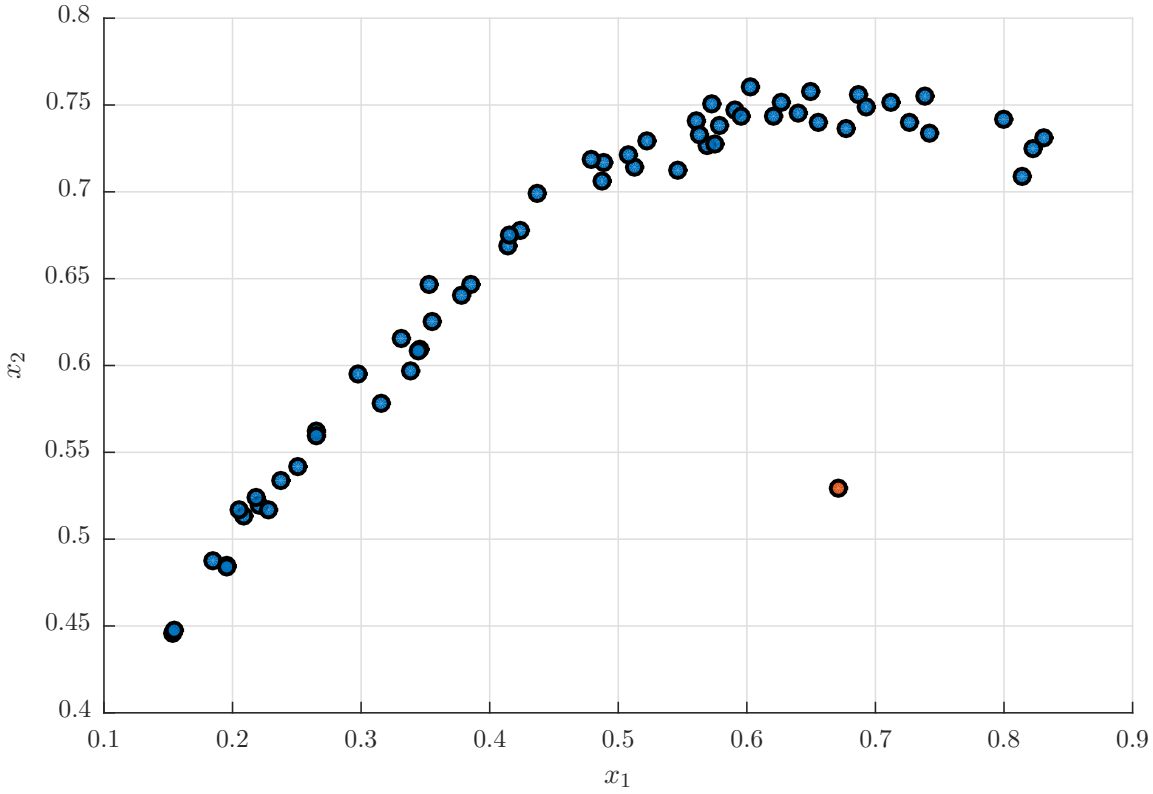


Fig. 1.5: Anomaly detection tries to discover observations that differ from the rest of the dataset.

Training Set

{Bread, Coke, Milk}
{Beer, Bread}
{Beer, Coke, Diaper, Milk}
{Beer, Bread, Diaper, Milk}
{Coke, Milk}

→

Rules Discovered

{Milk} → {Coke}
{Diaper, Milk} → {Beer}

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.

- 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

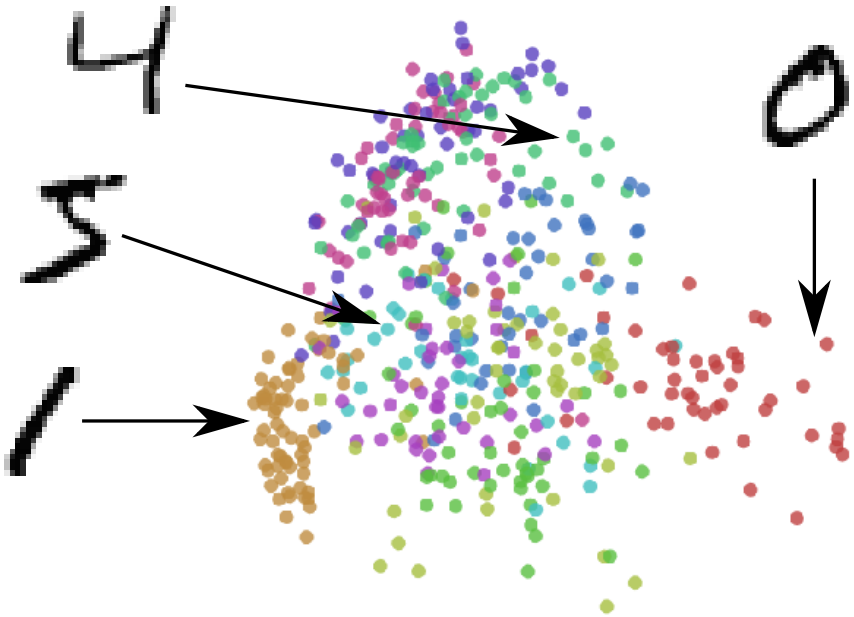


Fig. 1.7: Dimensionality reduction. High-dimensional images (each image contains 28×28 pixels corresponding to 784 dimensions) are mapped onto a 2D domain, that is compressed into a 2-dimensional vector. Colors indicate different digits $(0, 1, \ldots, 9)$. Notice, digits that are the most dissimilar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

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 looses 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. 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 wan't 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,

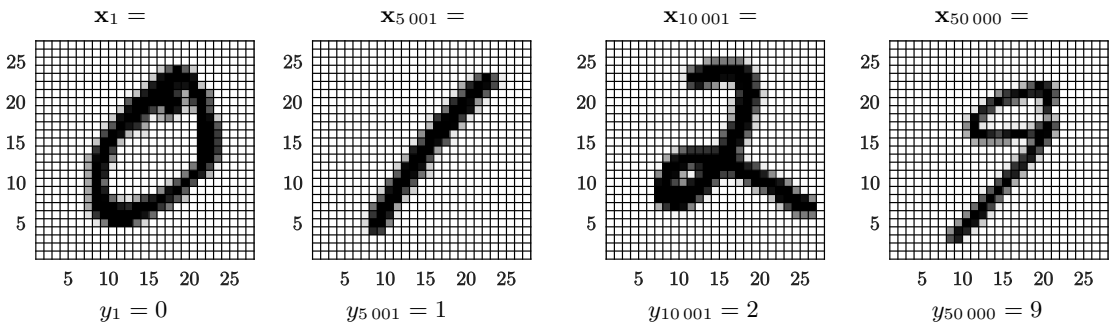


Fig. 1.8: Four observations (images of handwritten digits) from the MNIST dataset containing a total of $N = 60\,0000$ handwritten digits. Each observation consist of a 784-dimensional vector $\boldsymbol{x}_i$ and a label y_i which can be either $0, 1, \ldots, 9$.

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 which are introduced in this course 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 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 13.

1.3 Terminology of machine learning

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 $\boldsymbol{x}$ consisting of $M = 784$ real numbers [2] and we write this as $\boldsymbol{x} \in \mathbb{R}^M$. The goal is to build a machine which takes such an image $\boldsymbol{x}$ and outputs the identity of the digit, i.e. if it is $0, 1, \ldots, 9$. 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 $0, 1, \ldots, 9$ 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

[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].

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

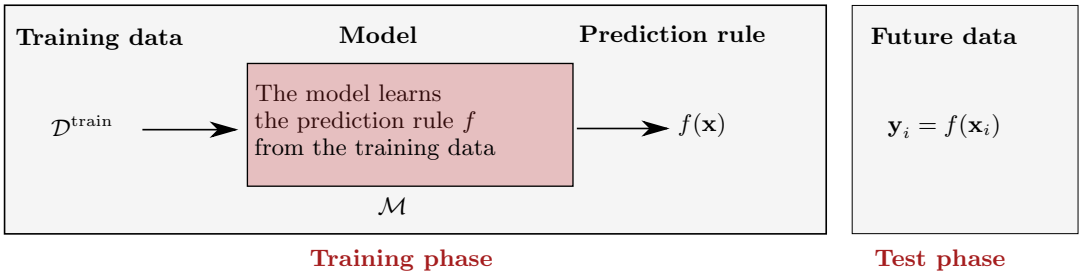


Fig. 1.9: A model in supervised learning. Given a set of training data, the model learns a prediction rule $f(\boldsymbol{x})$ in the training phase. Later, in the test phase, this rule can be applied to new, unobserved data.

(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, $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{5000}$, then 5 000 examples of the digit 1: $\boldsymbol{x}_{5001}, \boldsymbol{x}_{5002}, \ldots, \boldsymbol{x}_{10\,000}$ and so on up to image $\boldsymbol{x}_{50\,000}$ of the digit 9. Let $N = 50\,000$ be the number of examples, we collect all this information in a $N \times M$ matrix $\boldsymbol{M}$ and a N-dimensional vector $\boldsymbol{y}$:

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

where y_i is either $0, \ldots, 9$ depending on the digit in image $\boldsymbol{x}_i$. Taken together we let $\mathcal{D}^{\text{train}} = (\boldsymbol{X}, \boldsymbol{y})$ 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

$$f : \mathbb{R}^M \to \{0, \ldots, 9\}$$

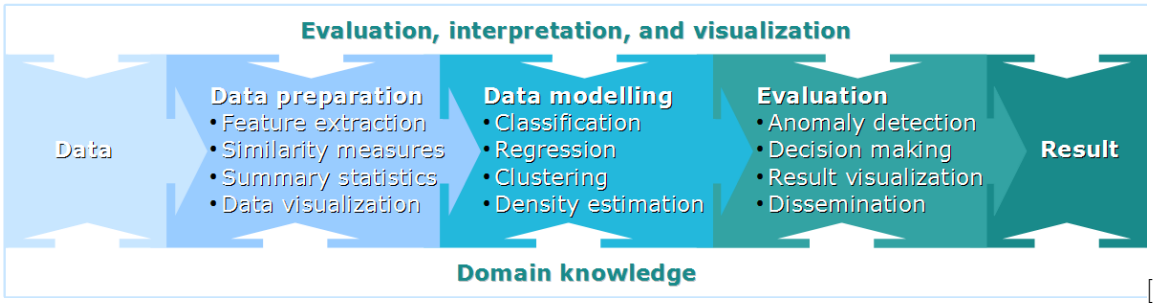
Learning the function f from the data is known as the *training phase* or alternatively as the *learning phase* or simply *learning*, see fig. 1.9. Once this function is learned it can be used to determine the identity of unobserved images of digits. For instance if for an image $\boldsymbol{x}$ we have that $f(\boldsymbol{x}) = 5$ 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 $\mathcal{D}^{\text{test}}$. 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 $\mathcal{M}_1, \ldots, \mathcal{M}_S$.

Generalization

Ideally we want the model to make as accurate predictions as possible on the test set. The ability to correctly categorize examples in the test set is known as *generalization* of the model. An important

point which can easily lead to confusion is that when a model $\mathcal{M}$ is trained on some data, how well the model generalizes to new data depends on what data the model was trained upon. That is, suppose we train the model on a specific dataset $\mathcal{D}^{\text{train}}$ and test it on a new dataset $\mathcal{D}^{\text{test}}$, then how well the model will perform will depend on both $\mathcal{D}^{\text{train}}$ and $\mathcal{D}^{\text{test}}$: If the test dataset was very difficult or unusual even the best model will perform very poorly and similarly if the training data set is very small or of poor quality. This implies that when we try to quantify the performance of models the performance is a random number since it depends on the training and test dataset which both have some randomness (for instance if the same group of people were asked to write new digits they would not be the same). This complicates the evaluation of machine learning methods and we will return to this point in chapter 9.

1.4 The machine learning workflow



[t]

Fig. 1.10: 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.

To summarize the above discussion we will now present the workflow of a machine learning practitioner fig. 1.10. 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 get a handle of our 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) 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 as $\mathcal{M}_1$.

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.

2

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 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 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) 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 cm^3.

Notice the types are a hierarchy. That is, if a variable is interval it must also be ordinal and nominal. 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 NaN [1]. 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.
- If we have many observations and only few have missing observations, we can discard the observations with missing values.

[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 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 $\boldsymbol{X}$ where $N = 142$ observations and $M = 8$ features. The corresponding matrix is simple:

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

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

transpose):

$$\boldsymbol{x}_2 = \begin{bmatrix}28 & 4 & 79 & 2625 & 82 & 4 & 18.6 & 1\end{bmatrix}^T$$

We will refer to element i, j of the matrix $\boldsymbol{X}$ 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 $\boldsymbol{y}$. All in all this means any dataset considered in this course will be a $N \times M$ matrix $\boldsymbol{X}$ and (if available) a N-dimensional vector $\boldsymbol{y}$ denoting the response or output variable (we will get back to the use of $\boldsymbol{y}$ 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 $\boldsymbol{X}, \boldsymbol{y}$ 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 $\boldsymbol{X}$ 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:

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

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

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

It is easy to verify that in the first case the mean of column j in the updated $\boldsymbol{X}$-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 $\boldsymbol{X}$ represent observations about animals where one column is the weight. Some

animals are very small (for instance weighting 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:

$$\tilde{\boldsymbol{X}} = \begin{bmatrix} \cdots & \log X_{1j} & \cdots \\ \cdots & \log X_{2j} & \cdots \\ & \vdots & \\ \cdots & \log X_{3j} & \cdots \end{bmatrix}$$

because in this case the difference in weight is: $\log(1.01) - \log(0.01) \approx 4.6$ compared to $\log(101) - log(100) \approx 0.01$ and thus much more informative for the smaller animal. We will call a transformation such as the logarithm *non-linear* whereas standardizing by subtracting mean and dividing with the variance is *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 $\boldsymbol{X}$ 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:

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

to $\boldsymbol{X}$ thereby obtaining a $N \times (M+1)$ 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 7.

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 $\boldsymbol{z}$ of dimension K where entry i is 1 only if $x = i$ and otherwise 0. For instance if $x = 3$ then $\boldsymbol{z} = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}^T$. 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:

$$\begin{bmatrix} 3 \\ 1 \\ 1 \\ 2 \\ 1 \\ 2 \\ 1 \\ 3 \\ \vdots \\ 1 \end{bmatrix} \leftrightarrow \begin{bmatrix} 0\ 0\ 1 \\ 1\ 0\ 0 \\ 1\ 0\ 0 \\ 0\ 1\ 0 \\ 1\ 0\ 0 \\ 0\ 1\ 0 \\ 1\ 0\ 0 \\ 0\ 0\ 1 \\ \vdots \\ 1\ 0\ 0 \end{bmatrix} \tag{2.2}$$

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 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 $\boldsymbol{x}$ this is written as

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

where

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

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 *somehow* 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) 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. Fall 2011 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. Fall 2013 question 1: 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. Fall 2014 question 1:

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-KarhunenLoève 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 was 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 a M-dimension vector space is simply the set of M-dimensional vectors $\boldsymbol{x}$ where

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{bmatrix}, \tag{3.1}$$

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

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

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

$$\boldsymbol{x}^T = \begin{bmatrix} x_1 & x_2 & \cdots & x_M \end{bmatrix}.$$

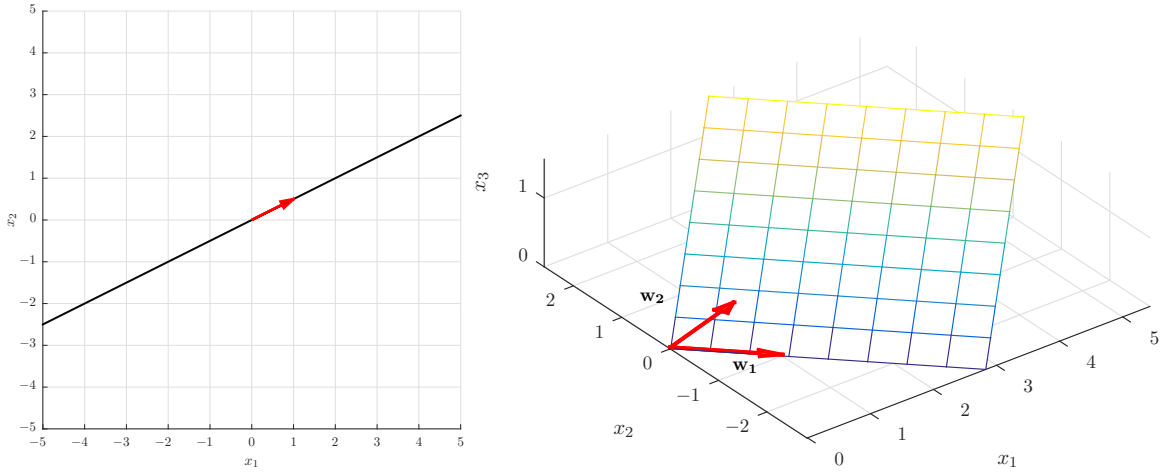


Fig. 3.1: Examples of a 1 and two dimensional subspace of $\mathbb{R}^2$ and $\mathbb{R}^3$ respectively. The subspaces can be thought of as generated by all linear combinations of the basis vectors shown as the red arrows.

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 $\boldsymbol{x}, \boldsymbol{y} \in V$ then

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

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 $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^M$ is any set of n vector in $\mathbb{R}^M$, we can then define the *span* of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ as all vectors $\boldsymbol{z}$ that can be written as

$$\boldsymbol{z} = a_1\boldsymbol{x}_1 + a_2\boldsymbol{x}_2 + \cdots + a_n\boldsymbol{x}_n \tag{3.2}$$

where $a_1, \ldots, a_n$ are arbitrary. We write this set of vectors as $V = \text{span}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ 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

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

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 $V_2 = \text{span}(\boldsymbol{w}_1, \boldsymbol{w}_2)$ where

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

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 $\boldsymbol{x}$ is

$$\|\boldsymbol{x}\| = \sqrt{\boldsymbol{x}^T\boldsymbol{x}} = \sqrt{x_1^2 + x_2^2 + \cdots x_M^2} \tag{3.5}$$

and two vectors $\boldsymbol{x}, \boldsymbol{y}$ are *orthogonal* if $\boldsymbol{x}^T\boldsymbol{y} = 0$. 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 $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ are said to be *linearly independent* if

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

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 $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ such that

$$\text{span}(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n) = V$$

and $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ are linearly independent.

The definition of a basis simply means that $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ 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 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. $\boldsymbol{v}_i^T\boldsymbol{v}_j = 0$ for $i \neq j$. If this is satisfied for a basis $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ 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 $\boldsymbol{x}$ in a subspace V with orthonormal basis $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$. Then by definition of a subspace $\boldsymbol{x}$ can be written as

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

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

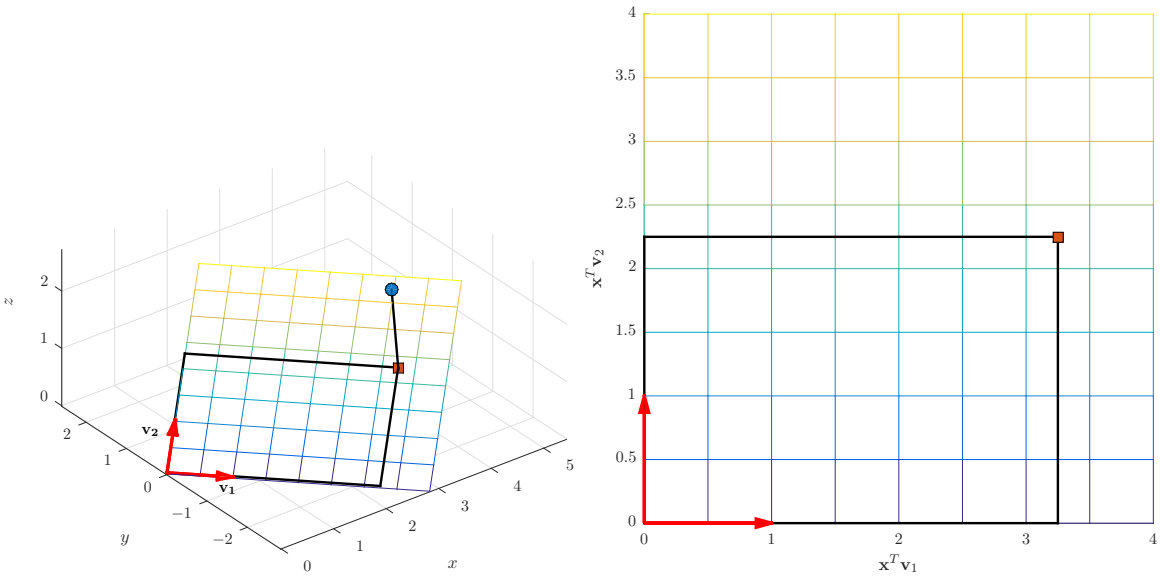


Fig. 3.2: Left pane: Projection of a 3D point $\boldsymbol{x}$ (blue circle) onto the subspace spanned by the orthonormal basis $\boldsymbol{v}_1, \boldsymbol{v}_2$. The projection, shown as the red square, lies within the subspace and the right pane shows the point in the 2D coordinate system given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

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

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

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

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

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

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

$$\boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_n \end{bmatrix}$$

is the coordinates of $\boldsymbol{x}$ in the subspace V. Notice, this is a n-dimensional vector whereas the space that $\boldsymbol{x}$ 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 $\boldsymbol{v}_1, \boldsymbol{v}_2$ shown as arrows. In the right pane we have plotted the projected point, $\boldsymbol{x}'$, with the coordinates in the new space V.

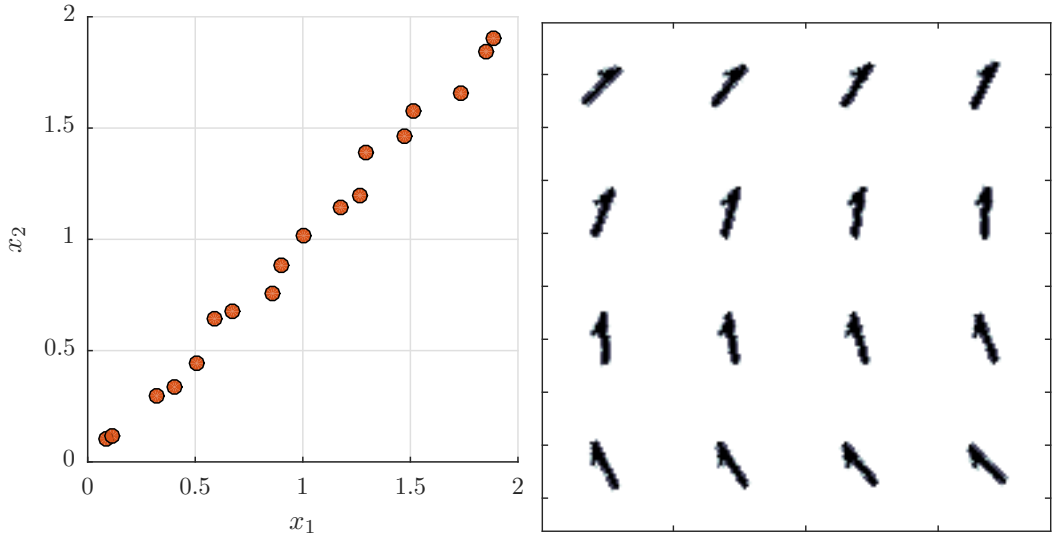


Fig. 3.3: (Left:) A simple 2D dataset comprise two features x_1 and x_2 that are noisy observations of the same quantity. Therefore, the dataset in fact only contains one "data-dimension" even though it is two-dimensional. (Right:) A more elaborate example dataset corresponding to $N = 9$ observations each corresponding to 48×48 pixel images of the same digit but rotated. From the perspective of the matrix $\boldsymbol{X}$ 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 in the first example but not in the second.

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

$$\boldsymbol{V} = \begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \ldots & \boldsymbol{v}_n \end{bmatrix} \tag{3.8}$$

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

$$\boldsymbol{b}^T = \boldsymbol{x}^T \boldsymbol{V} \tag{3.9}$$

Thus, suppose the original blue points was $\boldsymbol{x} \in \mathbb{R}^3$, 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:

$$\boldsymbol{b}^T = \begin{bmatrix} b_1 & b_2 \end{bmatrix} = \boldsymbol{x}^T \begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 \end{bmatrix}$$

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 $M = 2304$ dimensionsal 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 $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N \in \mathbb{R}^M$ 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

$$\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_N \in \mathbb{R}^n$$

where $n \leq M$ and such that $\boldsymbol{b}_i$ "represents" the observation $\boldsymbol{x}_i$. 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 $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ of a n-dimensional subspace V and define each $\boldsymbol{b}_i$ as the projection of $\boldsymbol{x}_i$ onto V. Since we want the projection to be invariant under addition of a constant we first subtract the mean from each $\boldsymbol{x}_i$. The general layout of the PCA algorithm is then:

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

where

$$\boldsymbol{V} = \begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_n \end{bmatrix}$$

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

So how do we select the projection matrix $\boldsymbol{V}$? 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 $\boldsymbol{v}_1$ implies. In fig. 3.4 (top panes) we have shown the projection of the same 2D dataset onto three different choices of $\boldsymbol{v}_1$ and in the bottom panes we have plotted the projected coordinates $b_i = \tilde{\boldsymbol{x}}_i^T \boldsymbol{v}_1$, i.e. the 1-dimensional "representation" of each $\tilde{\boldsymbol{x}}_i$. From an intuitive point of view the first projection is worse than the last since it lump 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 lump all classes together while in the third the residuals are much smaller and the data in the

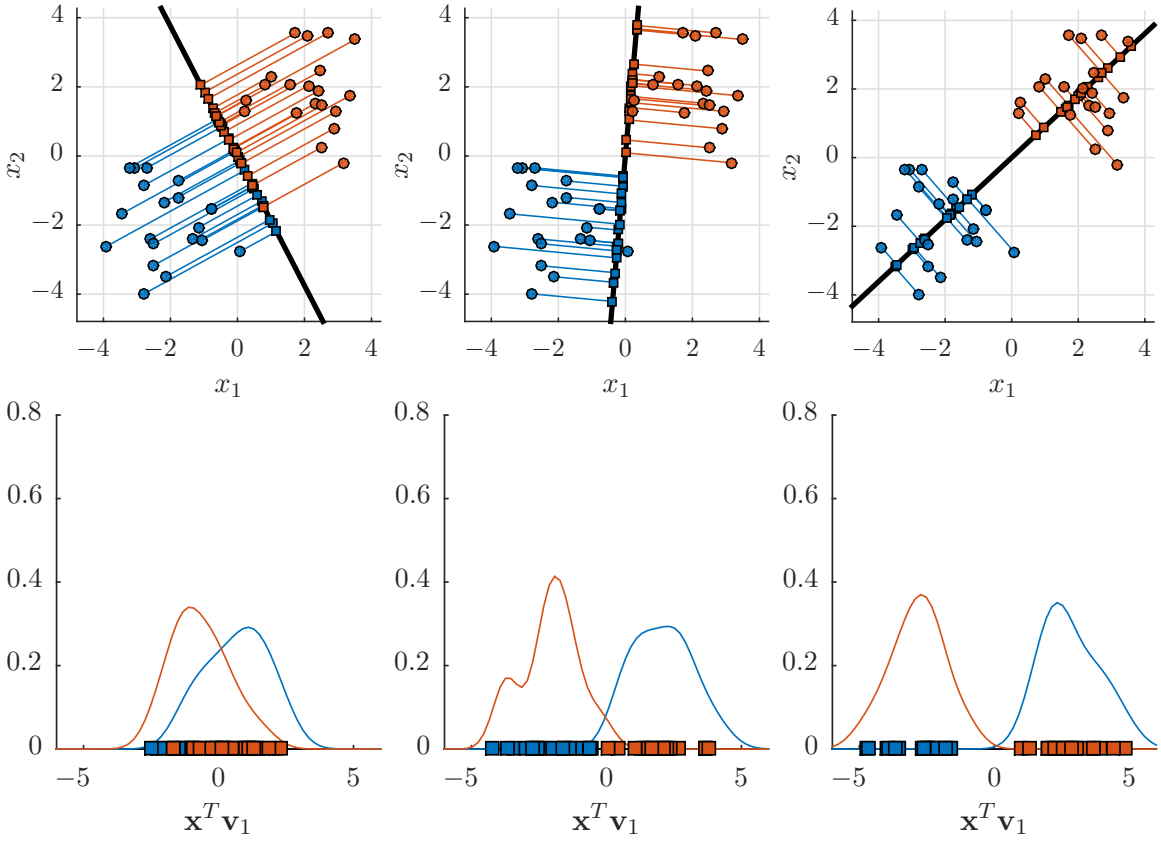


Fig. 3.4: An example 2D dataset colored for our convenience (the dots in the top panes are the same in all 3 panes) is projected onto three different 1D subspaces corresponding to different choices of basis vector $\boldsymbol{v}_1$. The dataset in the projected coordinate system is shown in the bottom 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.

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 $b_1, \ldots, b_N$, 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 $b_1, \ldots, b_N$ scaled by a factor of N [1]:

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

The reason for multiplying with N will be apparent later. This immediately gives us an idea for selecting $\boldsymbol{v}_1$: We simply make it the goal of PCA to select $\boldsymbol{v}_1$ to be the vector which maximizes the

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

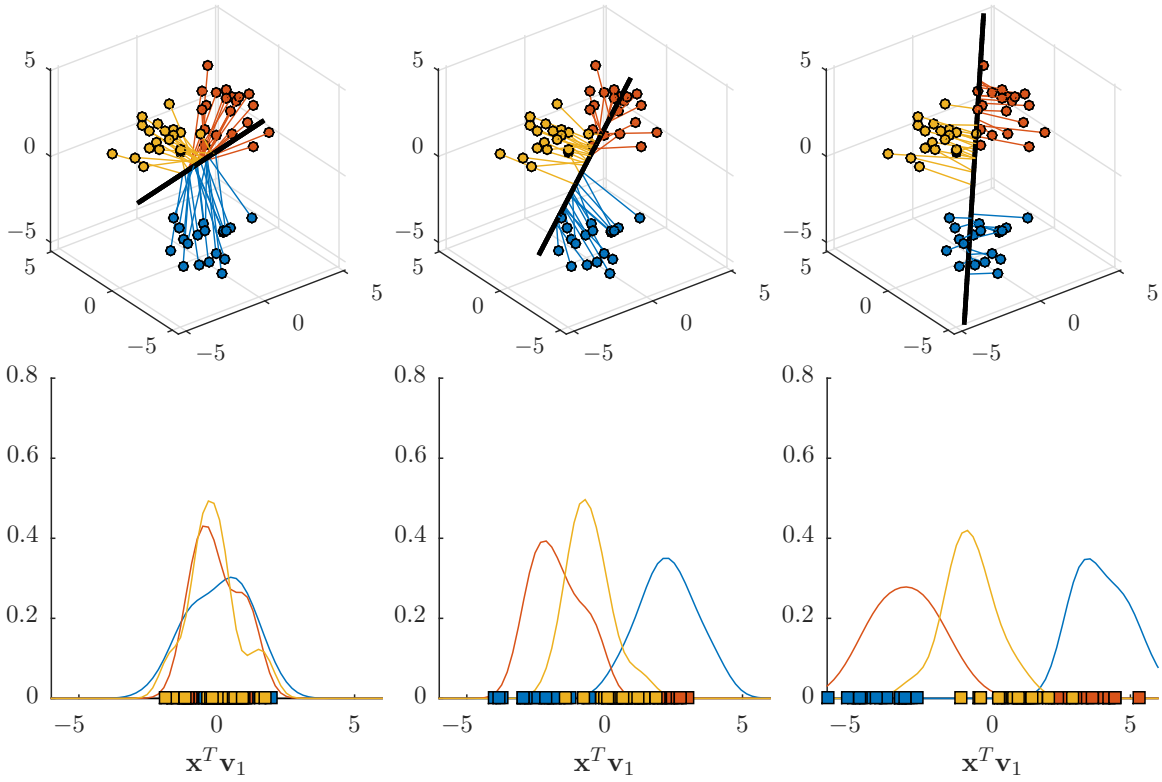


Fig. 3.5: Similar to the example fig. 3.4, a 3D dataset (colored for our convenience) is projected onto three different 1D subspaces corresponding to different choices of basis vector $\boldsymbol{v}_1$. The dataset in the projected coordinate system is shown in the bottom pane. As can be seen, different 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.

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,

$$\begin{aligned}\boldsymbol{v}_1 &= \text{The vector of length 1 that maximize } W \\ &= \arg\max_{\boldsymbol{v}^T\boldsymbol{v}=1} W.\end{aligned}$$

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 $\boldsymbol{X}$:

$$\bar{b} = \frac{1}{N}\sum_{i=1}^{N} b_i = \frac{1}{N}\sum_{i=1}^{N} \tilde{\boldsymbol{x}}_i^T \boldsymbol{v}_1 = \left(\frac{1}{N}\sum_{i=1}^{N} \tilde{\boldsymbol{x}}_i\right)^T \boldsymbol{v}_1 = \left(\left[\frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i\right] - \boldsymbol{m}\right)^T \boldsymbol{v}_1 = 0 \tag{3.11}$$

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

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

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

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

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

$$\frac{\partial}{\partial \lambda}\mathcal{L} = 1 - \boldsymbol{v}_1^T \boldsymbol{v}_1^T \quad = 0 \tag{3.14}$$

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

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

$$\boldsymbol{S}\boldsymbol{v}_1 = \lambda \boldsymbol{v}_1 \tag{3.16}$$

This means that v_1 should be an eigenvector of $\boldsymbol{S}$ 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:

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

Thus, we should select $\boldsymbol{v}_1$ as the eigenvector of $\boldsymbol{S}$ 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 a $N \times n$ matrix

$$\boldsymbol{B}^T = \begin{bmatrix} \boldsymbol{b}_1 & \boldsymbol{b}_2 & \cdots & \boldsymbol{b}_N \end{bmatrix}^T.$$

The projection can then be written as

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

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

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

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

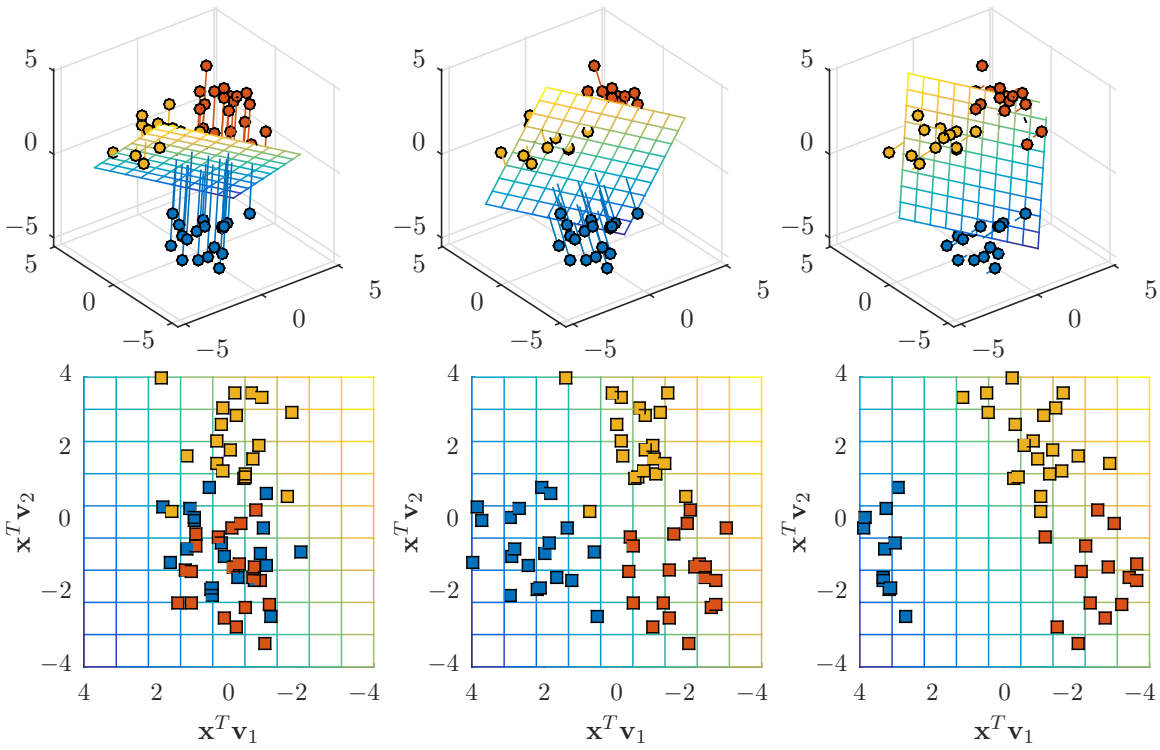


Fig. 3.6: This time, the same 3D dataset as in fig. 3.5 is projected onto three different 2D planes corresponding to the three panes in the top row. The inserts in the bottom row is the points in the coordinate system corresponding to the subspace. We again see the right most pane, where the points are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

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

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

It turns out the solution to this maximization problem is to select $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ 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 $v_1, \cdots, v_n$ 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 $\boldsymbol{X}$, the SVD computes three matrices

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

such that

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

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

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

This can be used to show:

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

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

$$\boldsymbol{V}_n = \begin{bmatrix} \boldsymbol{v}_1 \ \boldsymbol{v}_2 \cdots \ \boldsymbol{v}_n \end{bmatrix}$$

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 $\boldsymbol{x}_i^T$ onto the subspace spanned by the first n principal components can therefore be written as $\boldsymbol{b}_i^T = \boldsymbol{x}_i^T\boldsymbol{V}_n$.

3.3.1 The PCA algorithm

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

- Subtract the mean: $\tilde{\boldsymbol{x}}_i = \boldsymbol{x}_i - \boldsymbol{m}$, $\boldsymbol{m} = \frac{1}{N}\sum_{i=1}^N \boldsymbol{x}_i$
- 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: $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T = \tilde{\boldsymbol{X}}$
- The n first principal components are $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ and coordinates of observation i when projected onto the subspace spanned by the first n principal components are

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

  where $\boldsymbol{V}_n = \begin{bmatrix}\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n\end{bmatrix}$.

3.3.2 Variance explained by the PCA

The vectors $\boldsymbol{b}_i$ in the matrix $\boldsymbol{B}$ represents the coordinates of the vector $\boldsymbol{x}_i$ 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 $\boldsymbol{b}_i = \begin{bmatrix} b_{i1} & b_{i2} & \cdots & b_{in}\end{bmatrix}^T$ 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:

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

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

$$\boldsymbol{X}' = \left(\boldsymbol{V}_n\boldsymbol{B}^T\right)^T = \tilde{\boldsymbol{X}}\boldsymbol{V}_n\boldsymbol{V}_n^T. \tag{3.17}$$

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

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

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

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

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

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

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

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.

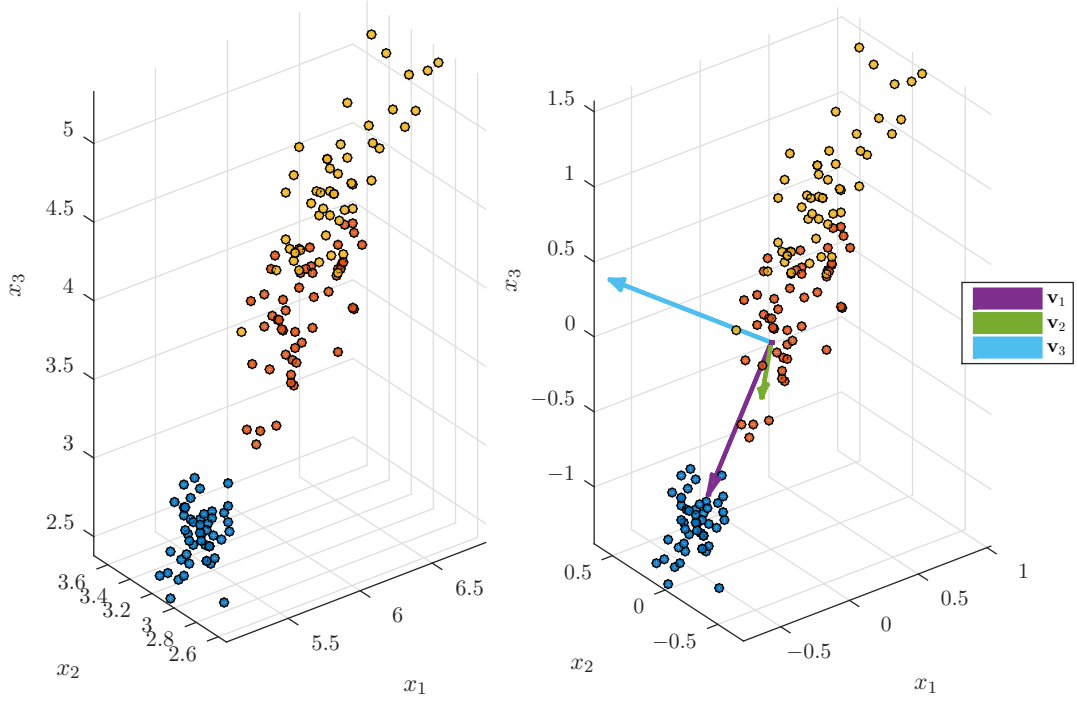


Fig. 3.7: The Fisher Iris dataset consisting of $N = 150$ observations of three types of flowers. In the left pane the dataset is plotted as a scatter plot whereas in the right pane the mean of the dataset has been substracted to obtain the centered matrix $\tilde{\boldsymbol{X}}$ and the three principal 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 PCA analysis is to subtract the mean of the dataset to obtain the centered matrix $\tilde{\boldsymbol{X}}$ ($\tilde{X}_{ij} = X_{ij} - \frac{1}{N}\sum_{i=1}^{N} X_{ij}$). Then, a SVD is performed to obtain the decomposition $\tilde{\boldsymbol{X}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$. The mean vector $\boldsymbol{m}$ and the principal component directions, i.e. columns of $\boldsymbol{V} = \begin{bmatrix}\boldsymbol{v}_1 & \boldsymbol{v}_2 & \boldsymbol{v}_3\end{bmatrix}$, are approximately:

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

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

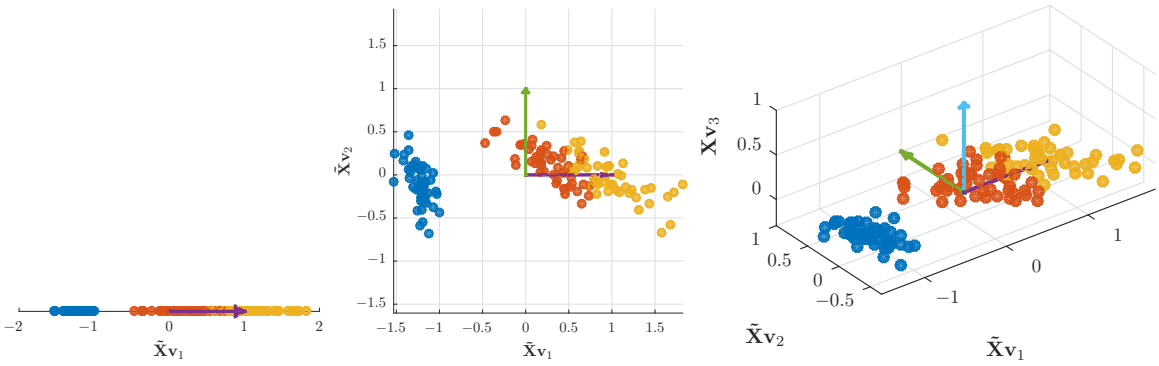


Fig. 3.8: The Fisher Iris dataset of fig. 3.7 projected onto the principal directions. In the first pane only the first principal direction is used corresponding to a large loss of information. In the second pane two principal directions are used better preserving the structure of the data and finally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

found by computing the vectors $\tilde{\boldsymbol{X}}v_1$ and $\tilde{\boldsymbol{X}}v_2$ and plotting these as a scatter plot. As can be 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 $\boldsymbol{v}_3$ and then $\boldsymbol{v}_3$ and $\boldsymbol{v}_2$ and therefore loose information. Finally let's turn to the variance explained by the principal directions. This can be computed from $\boldsymbol{\Sigma}$ and in our example

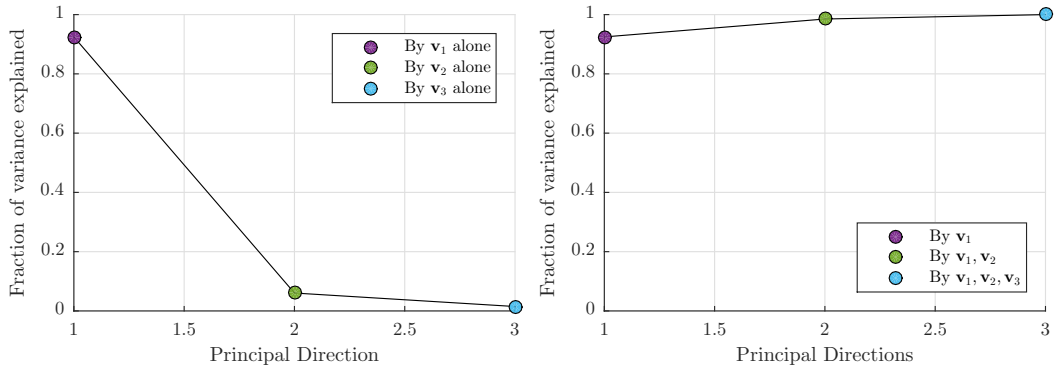


Fig. 3.9: (Left:) Variance explained by each of the three principal directions from the Fisher Iris example of fig. 3.7. Nearly all variation is explained by the first and second principal directions which can be interpreted as the original dataset residing on a 2D plane spanned by the first and second principal directions. (Right:) Cumulative variance explained by the subspaces spanned by only $\boldsymbol{v}_1$, then $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ and finally $\boldsymbol{v}_1$, $\boldsymbol{v}_2$ and $\boldsymbol{v}_3$. Compare to fig. 3.8.

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

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 A high-dimensional example

The previous illustrations of principal component analysis has considered 1- and 2-dimensional subspaces in 2- or 3-dimensional spaces. These applications are appealing since we can visualize each step, however, the utility of PCA stems from its applicability to very high-dimensional problems.

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 $\boldsymbol{X}$ of size $N \times M$ where $M = 784$. Suppose we compute the first two principal components of $\tilde{\boldsymbol{X}}$ using the PCA algorithm and plot $\tilde{\boldsymbol{X}}$ projected onto these first two components, $\tilde{\boldsymbol{X}}\boldsymbol{V}_2$. A plot of the projected data can be seen in the top of fig. 3.10 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. 3.10. 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 particular 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 corresponds 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.

To get an idea about how much information we lose, we consider the *full* MNIST dataset containing all 10 classes of digits. In the top row of fig. 3.11 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 and fig. 3.13 we have plotted the data 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 remaining PCA components largely overlap.

3.4.2 Uses of PCA

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

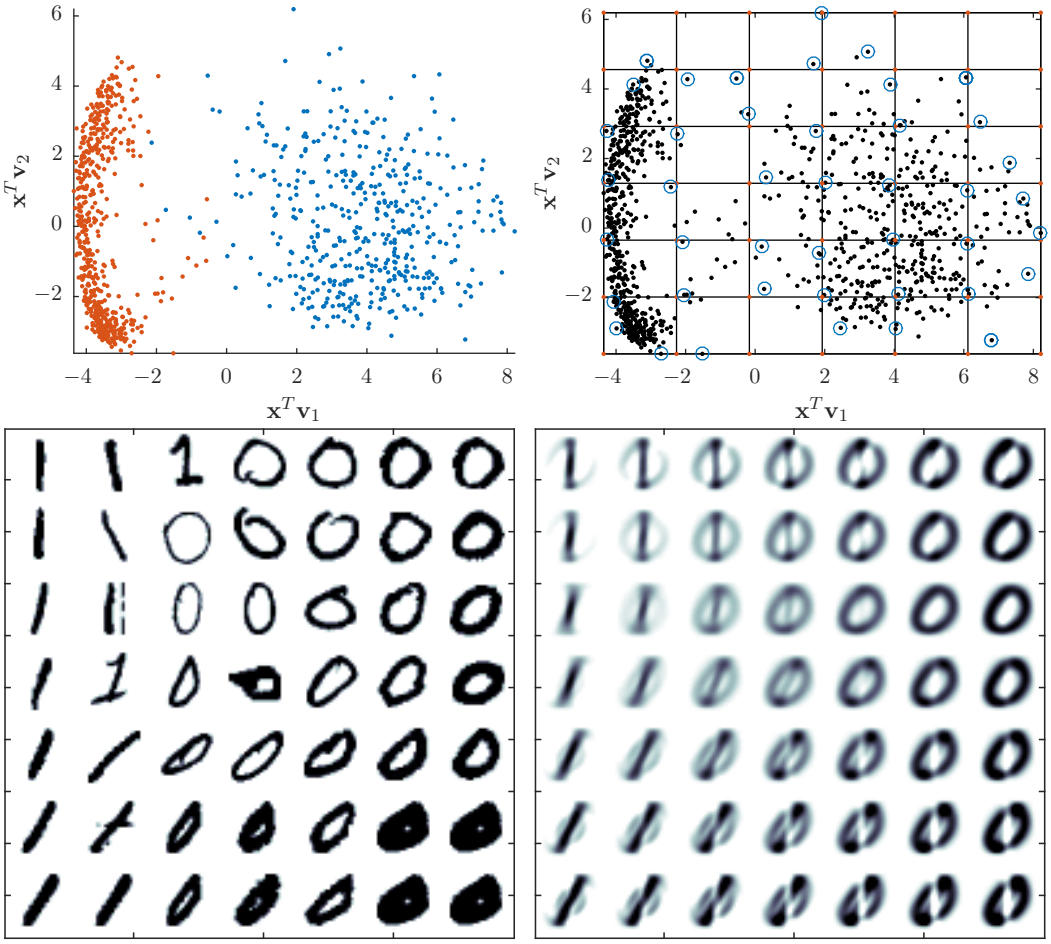


Fig. 3.10: $N = 1000$ observations from the MNIST dataset of the digits 0 and 1 are projected onto the first 2 principal directions as shown in the top-left pane. The red dots correspond to the digit 1 and we see the first principal direction easily allows us to distinguish the digits. If we select the 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 seperates 1's from 0's 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.

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.

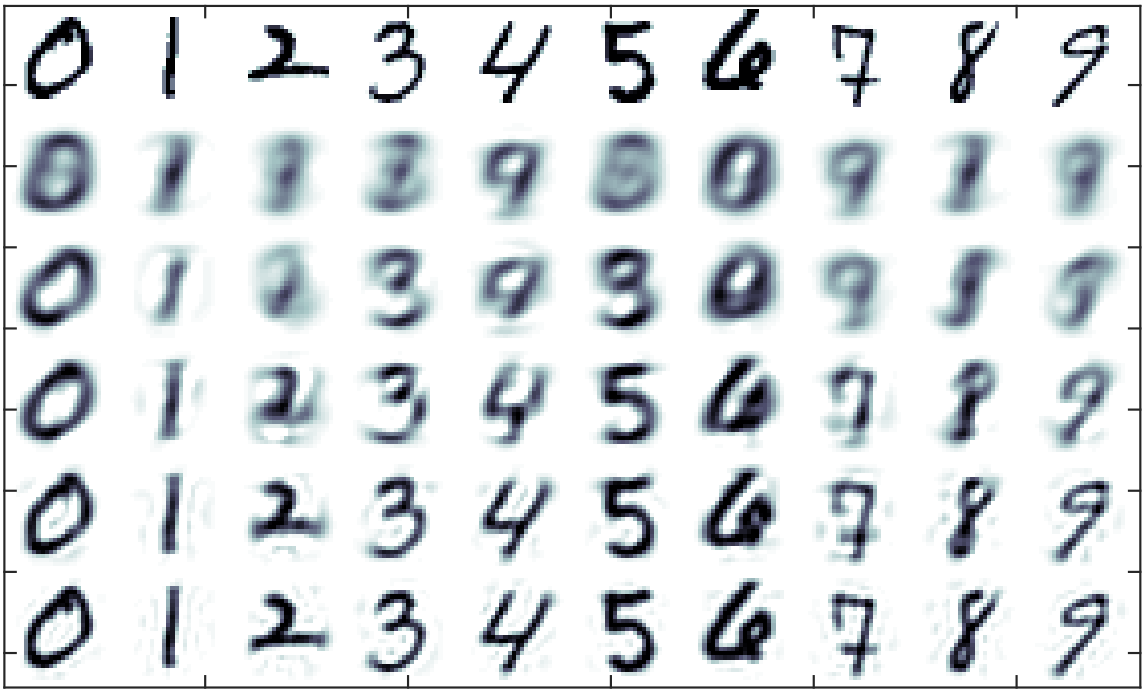


Fig. 3.11: Top row: 10 randomly chosen digits from the MNIST dataset. They are then reconstructed from $n = 2, 5, 20, 50, 100$ PCA components in the next 5 rows. We see that about 50 principal directions is enough to make a fairly good reconstruction of the images corresponding to a compression factor of about 16.

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 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 scale the normalization step is important. For instance, suppose one dataset record the height of a person in meters as well as the persons 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 variance will ensure the height and income vary with roughly the same amount and PCA will easier pick out potential correlations.

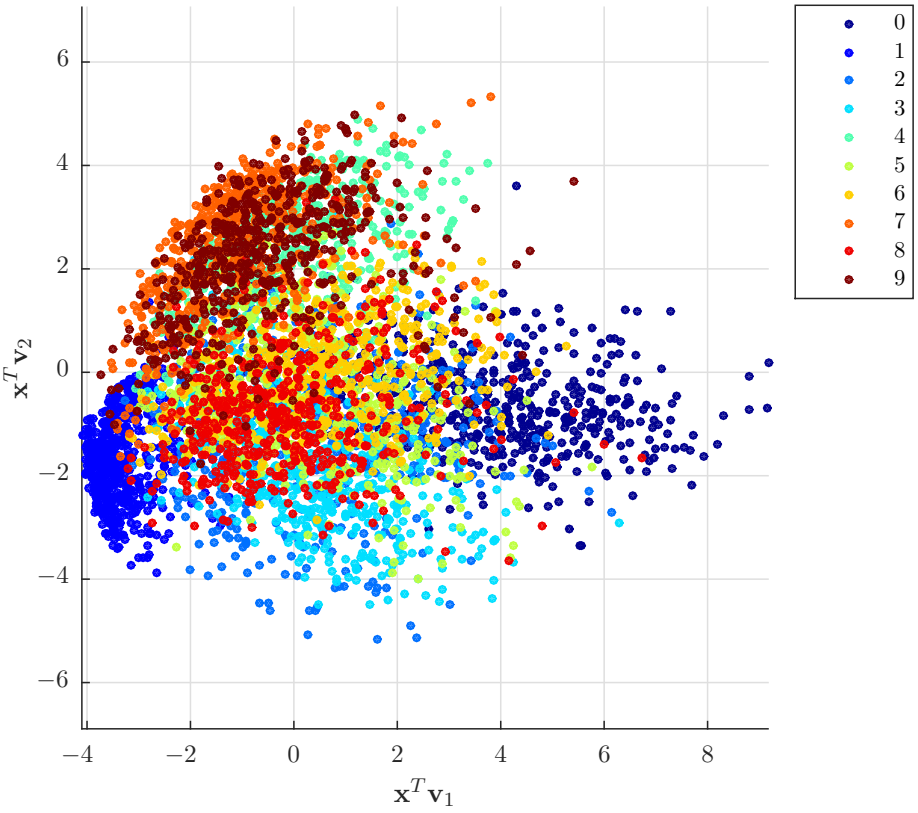


Fig. 3.12: $N = 2000$ observations of the MNIST dataset projected onto principal direction $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$

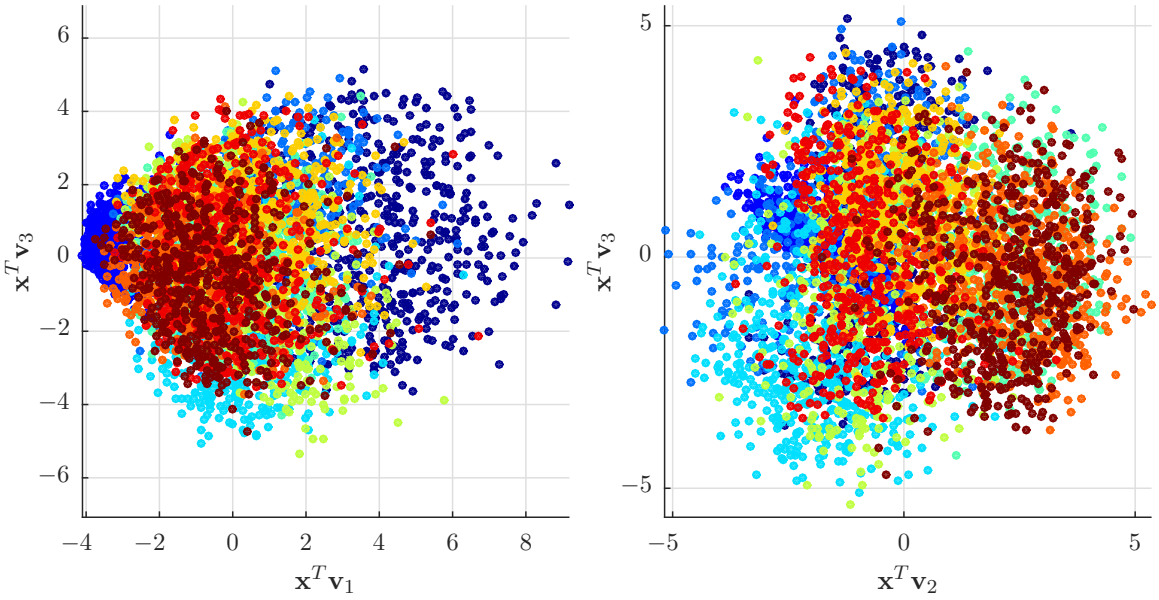


Fig. 3.13: $N = 2000$ observations of the MNIST dataset projected onto principal direction $\boldsymbol{v}_1, \boldsymbol{v}_3$ (left pane) and $\boldsymbol{v}_2, \boldsymbol{v}_3$ (right pane)

Problems

3.1. Spring 2012 question 4: The first and second principal components directions of the data in the RAT dataset in Table 3.1 are:

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

and in Figure 3.14 the data projected onto the first two principal components is plotted against the average consumer ratings (RAT). Which of the following statements is *correct*?

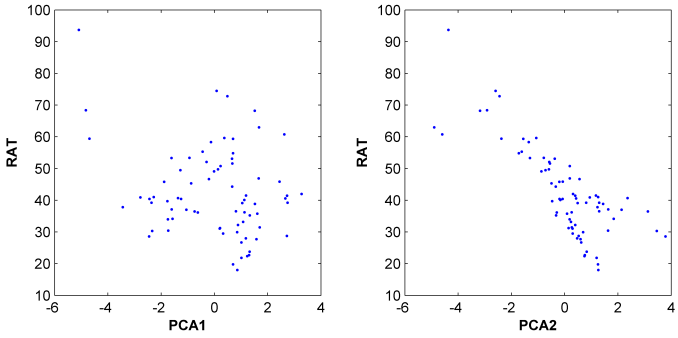


Fig. 3.14: The output RAT plotted against the first and second principal component respectively.

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%, 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. Fall 2011 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:

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

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. Fall 2013 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 $\boldsymbol{S}$ and $\boldsymbol{V}$:

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

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

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. Fall 2014 question 3: A principal component analysis is applied to a dataset composed of 1000 data points. After applying the principal component analysis, the datapoints (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. 3.15 corresponds to this description?

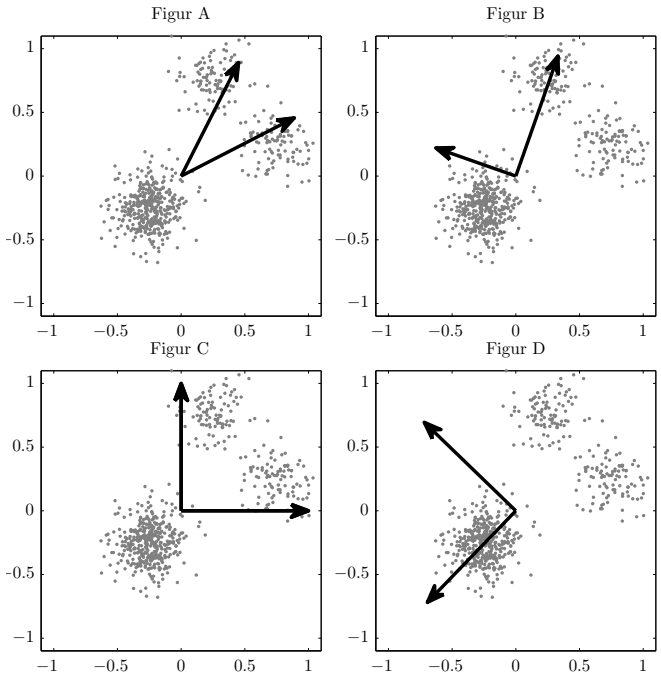


Fig. 3.15: Dataset of 1000 data points along with four possible choices of right singular vectors

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

3.5. Fall 2014 question 4: A principal component analysis is carried out on a dataset comprised of three data points $\boldsymbol{x}_1, \boldsymbol{x}_2$ and $\boldsymbol{x}_3$ collected in a $N \times M$ matrix $\boldsymbol{X}$ such that each row of the matrix is a data point. Suppose the matrix $\tilde{\boldsymbol{X}}$ corresponds to $\boldsymbol{X}$ with the mean of each columns substracted i.e.

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

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

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

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

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

A $\begin{bmatrix} -3.06 & 0.31 \end{bmatrix}^\top$
B $\begin{bmatrix} -0.78 & 0.85 \end{bmatrix}^\top$
C $\begin{bmatrix} -1.07 & 0.21 \end{bmatrix}^\top$
D $\begin{bmatrix} -3.16 & 0.23 \end{bmatrix}^\top$
E Don't know.

3.6. Spring 2014 question 3: A principal component analysis is carried out on the wholesale data based on x_1–x_6. The mean is subtracted 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 $\boldsymbol{S}$ and $\boldsymbol{V}$:

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

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

We note that both $\boldsymbol{S}$ and $\boldsymbol{V}$ 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 $x_1, \ldots, x_N$ 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:

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

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

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

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, $x_1, x_2, \ldots, x_N$ in ascending order, i.e. as $x'_1 \leq x'_2 \leq \cdots \leq x'_N$. 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:

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

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: The p'th percentile of a sample x 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 $i = 1, \ldots, N$. 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 has a grade average *less* than $x_{p=90\%}$ and 20 students has a grade average *greater* than $x_{p=90\%}$. If we use the notation introduced for the median we might reasonably expect $x_{p=90\%} \approx x'_{\lceil Np \rceil}$ [3] compare to the median with $p = 50\%$. However we had 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 in different computation environments and the details need not concern us here [4].

[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] A popular way is linear interpolation. Suppose the dataset is sorted as $x'_1 \leq x'_2 \leq \cdots \leq x'_N$ 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 $i = 1, \ldots, N$ and the function $X(z)$:

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

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

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

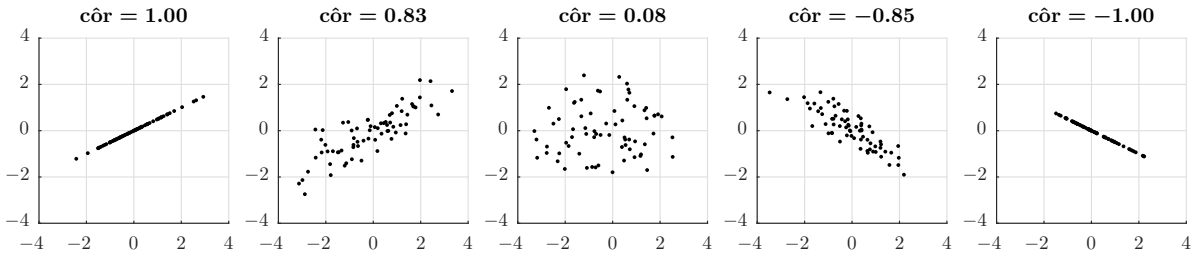


Fig. 4.1: Five two-dimensional datasets and their correlation as estimated from eq. (4.7).

Mode

We also define the *mode* as the most frequently occurring value of $x_1, \ldots, x_N$. 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 change and visa-versa. Suppose we have a dataset containing two attributes x and y with recorded values $x_1, x_2, \ldots, x_N$ and $y_1, y_2, \ldots, y_N$. 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

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

Notice that $\text{cov}[x, x] = \text{Var}[x]$. Given a dataset of M attributes, $x_1, \ldots, x_M$, we can compute the pairwise covariance between any two attributes $\text{cov}[x_i, x_j]$ and collect all these in a $M \times M$ matrix $\tilde{\boldsymbol{\Sigma}}$ 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:

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

A correlation of 0 means that x tell 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 and a negative correlation tells us that if x is large then y is small.

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:

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

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

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 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

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

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 $\boldsymbol{X}$ (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 matrix [5]. For instance the first column of $\boldsymbol{X}$ is $\begin{bmatrix}0 & 0 & 1\end{bmatrix}^T$ because only the last document contains the word "assembled".

[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 $\boldsymbol{X}$, and we have decided to re-use the terminology.

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 $\boldsymbol{x}, \boldsymbol{y}$ 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 three rules:

$$\begin{aligned} \textit{non-negativity} \quad & d(\boldsymbol{x}, \boldsymbol{y}) \geq 0, && (4.8)\\ \textit{identity of indiscernibles} \quad & d(\boldsymbol{x}, \boldsymbol{y}) = 0 \text{ if and only if } \boldsymbol{x} = \boldsymbol{y}, && (4.9)\\ \textit{symmetry} \quad & d(\boldsymbol{x}, \boldsymbol{y}) = d(\boldsymbol{y}, \boldsymbol{x}), && (4.10)\\ \textit{triangle inequality} \quad & d(\boldsymbol{x}, \boldsymbol{y}) \leq d(\boldsymbol{x}, \boldsymbol{z}) + d(\boldsymbol{z}, \boldsymbol{y}). && (4.11) \end{aligned}$$

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 all three rules is called a *metric*. A common way to define distances is as the magnitude of the difference of observations, $\boldsymbol{x} - \boldsymbol{y}$. Naturally this presume we can subtract the two observations $\boldsymbol{x}$, $\boldsymbol{y}$ from each other, however, this is nearly always possible. In a vector space such a measure of magnitude is called a *norm* and is denoted $\|\boldsymbol{x}\|$. It must in turn obey:

$$\begin{aligned} \textit{non-negativity} \quad & \|\boldsymbol{x}\| > 0 \text{ if } \boldsymbol{x} \neq \boldsymbol{0}, && (4.12)\\ \textit{scaling} \quad & \|a\boldsymbol{x}\| = |a|\|\boldsymbol{x}\| && (4.13)\\ \textit{triangle inequality} \quad & \|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|. && (4.14) \end{aligned}$$

Then we can define the distance from the norm as

$$d(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|. \tag{4.15}$$

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

$$\|\boldsymbol{x}\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_M^2}. \tag{4.16}$$

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

$$\|\boldsymbol{x}\|_p = \left(|x_1|^p + |x_2|^p + \cdots + |x_M|^p\right)^{\frac{1}{p}}. \tag{4.17}$$

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

$$\|\boldsymbol{x}\|_\infty = \max\{|x_1|, |x_2|, \ldots, |x_M|\}, \tag{4.18}$$

and the corresponding distance is denoted $d_p(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|_p$. For technical reasons the $p < 1$ case is more difficult. In general, we define the p-distance when $0 < p < 1$ as:

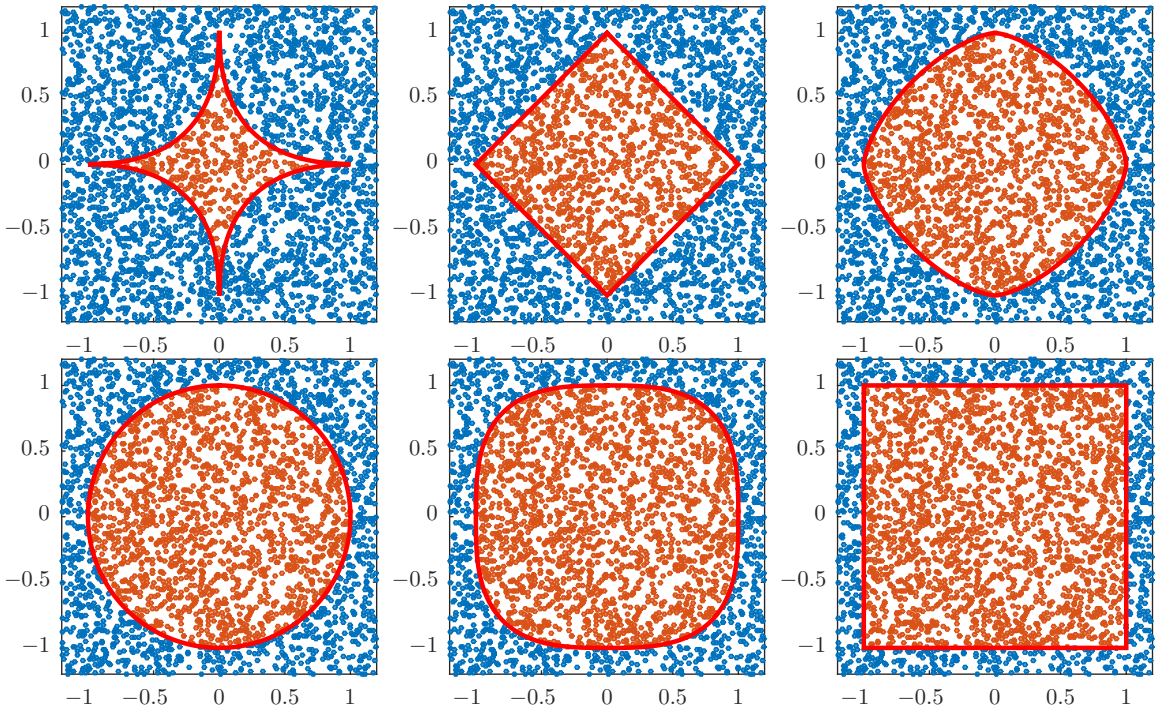


Fig. 4.2: Illustration of the p-distance for various values of p. A point $\boldsymbol{x}$ is colored red if it's p distance to the center $\mathbf{0}$ is less than 1, $d_p(\boldsymbol{x}, \mathbf{0}) \leq 1$. Top row: $p = \frac{1}{2}, 1, \frac{3}{2}$ and bottom row: $p = 2, 3, \infty$. The red line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

$$d_p(\boldsymbol{x}, \boldsymbol{y}) = |x_1|^p + |x_2|^p + \cdots + |x_M|^p, \tag{4.19}$$

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

$$\|\boldsymbol{x}\|_0 = |x_1|^0 + |x_2|^0 + \cdots + |x_M|^0, \tag{4.20}$$

which counts the number of non-zero coordinates of $\boldsymbol{x}$ the $p = 0$ norm[6]. 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. $d_p(\boldsymbol{x}, \mathbf{0}) \leq 1$, for 6 different values of p. For completeness sake we should also mention the Fröbenius norm which we encountered earlier in chapter 3:

$$\|\boldsymbol{X}\|_F = \sqrt{\sum_{i=1}^{N}\sum_{j=1}^{M} X_{ij}^2} = \sqrt{\text{trace}(\boldsymbol{X}^T\boldsymbol{X})}. \tag{4.21}$$

4.3.1 The Mahalanobis Distance

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

[6] This is really a misnomer from a mathematical standpoint since this function does not obey the mathematical properties of a norm, however it has become common terminology within machine learning.

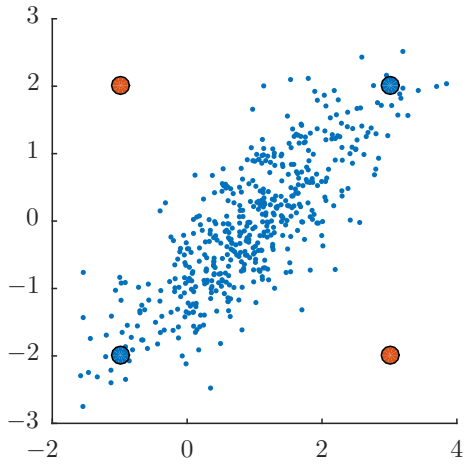


Fig. 4.3: A simple 2D dataset to illustrate the Mahalanobis distance. If we estimate the covariance matrix from the dataset, the Mahalanobis distance between the red points is 13 but only 4.15 between the blue points.

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

Notice if $\boldsymbol{\Sigma} = \boldsymbol{I}$ the Mahalanobis distance reduce to the Euclidian distance: $d_M(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{(\boldsymbol{x} - \boldsymbol{y})^T \boldsymbol{I} (\boldsymbol{x} - \boldsymbol{y})} = \|\boldsymbol{x} - \boldsymbol{y}\|_2$. 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 $\boldsymbol{x}, \boldsymbol{y}$ as input and is large when $\boldsymbol{x}$ is very similar to $\boldsymbol{y}$. Obviously, a measure of similarity can be constructed from a distance measure by a simple mapping. The most simple way is to define $s(\boldsymbol{x}, \boldsymbol{y}) = -d(\boldsymbol{x}, \boldsymbol{y})$, 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:

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

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 $\boldsymbol{x}$ is binary, i.e. $x_i = 0, 1$ for all i. Suppose we define

$$f_{11} = \text{Number of entries } i \text{ where } x_i = 1 \text{ and } y_i = 1 \tag{4.22}$$
$$f_{10} = \text{Number of entries } i \text{ where } x_i = 1 \text{ and } y_i = 0 \tag{4.23}$$
$$f_{01} = \text{Number of entries } i \text{ where } x_i = 0 \text{ and } y_i = 1 \tag{4.24}$$
$$f_{00} = \text{Number of entries } i \text{ where } x_i = 0 \text{ and } y_i = 0. \tag{4.25}$$

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

$$\textit{Simple Matching Coefficient} \quad \text{SMC}(\boldsymbol{x}, \boldsymbol{y}) = \frac{f_{11} + f_{00}}{M} \tag{4.26}$$
$$\textit{Jaccard Similarity} \quad \text{J}(\boldsymbol{x}, \boldsymbol{y}) = \frac{f_{11}}{f_{11} + f_{10} + f_{01}} \tag{4.27}$$
$$\textit{Cosine similarity} \quad \cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{f_{11}}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|} \tag{4.28}$$

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

$$\textit{Extended Jaccard Similarity} \quad \text{EJ}(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x}^T\boldsymbol{y}}{\|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2 - \boldsymbol{x}^T\boldsymbol{y}} \tag{4.29}$$
$$\textit{Cosine similarity} \quad \cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x}^T\boldsymbol{y}}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|} \tag{4.30}$$

Why do we need these different definitions? It is useful to consider these definitions in relationship with the term-document example of table 4.1. We 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: $\text{SMC}(1-\boldsymbol{x}, 1-\boldsymbol{y}) = \text{SMC}(\boldsymbol{x}, \boldsymbol{y})$. 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 smokes and a "one" if he does not smoke (or visa-versa).

However, for some datasets what is a zero and what is a one is asymmetric. 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 typically be large even for documents that have nothing in common: According to the SMC 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 suppose we compare two documents $\boldsymbol{x}$ and $\boldsymbol{y}$ which are on the same topic, but $\boldsymbol{x}$ is much larger than $\boldsymbol{y}$. In this case $\boldsymbol{x}$ will (likely) contain many words not found in $\boldsymbol{y}$ even though all words in $\boldsymbol{y}$ is in $\boldsymbol{x}$. Normalizing by the document length, as is done in the Cosine similarity, will somewhat correct for this problem and is therefore more suitable if some vectors has far more 1s than others and this difference is not in itself considered very informative.

Problems

4.1. Fall 2014 question 10:
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 $d(o_i, o_i) = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_1 = \sum_{k=1}^{M} |x_{ik} - x_{jk}|$, 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 $\text{J}(o_1, o_3) = 0.533$
C $\text{SMC}(o_1, o_3) = 0.533$
D There is insufficient information to draw specific conclusions.
E Don't know.

4.2. Spring 2013 question 18: 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. Fall 2013 question 18: 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*?

	YA_Y	YA_N	OA_Y	OA_N	PA_Y	PA_N
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 (YA_Y, YA_N), were operated after 1960 or not (OA_Y, OA_N), and had positive axillary nodes or not (PA_Y, PA_N).

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.

5

Probabilities and Bayes' theorem

Correct reasoning is central to many areas of intellectual human endeavor, including philosophy (how ought we reason?), cognitive science (how do we reason?), artificial intelligence (how do we build reasoning machines?), and science (what does reason tell us about theories given experimental evidence?). Considering the importance of reasoning correctly, it is perhaps surprising a large class of correct reasoning can be described with a simple equation the reader is likely familiar with, Bayes' theorem. Bayes' theorem was originally discovered by Thomas Bayes who considered a problem very akin to the binomial distribution example we will consider in section 5.5 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]. A reader who is only interested in an operational understanding of Bayes' theorem, or is familiar with basic probability theory (probabilities and densities) can consider skipping the sections marked with an asterisk.

5.1 The problem with logic*

The most famous attempt to describe reasoning formally is classical logic, originally developed by Aristotle in the 4th century BCE classical logic considers binary true/false propositions such as

$$\begin{aligned} A &: \textit{My bicycle is stolen.} \\ B &: \textit{My bicycle is not where I left it.} \\ \overline{A} &: \textit{The negation of A, "Not A".} \\ A \to B &: \textit{If } A \textit{ is true then } B \textit{ is true.} \end{aligned}$$

The aim being to provide rules for how the truth of some propositions can be deduced from the truth of other propositions. For instance, suppose we know that A is true and that A implies B (written as $A \to B$), the rule in classical logic known as *modus ponens* allow us to conclude that B is true, i.e. that it is true that my bicycle is not where I left it. Put formally

$$\text{If } A, \text{ and } A \to B, \text{ then } B \text{ is true.}$$

Alternatively, if we let $\overline{A}$ denote the negation of A, the following is also valid:

$$\text{If } \overline{B}, \text{ and } A \to B, \text{ then } \overline{A} \text{ is true}$$

or to put the argument in words, if my bicycle is where I left it then it is not stolen.

Classical logic provides a model of how humans ought to reason and being logical is often considered equivalent to reasoning correctly. It is therefore tempting to consider building a robot that reasons using logic and indeed this was a major approach to artificial intelligence during the 1960s. The problem with this approach is the world is fundamentally uncertain and in nearly any situation, there will be a range of logically possible explanations making logical inference useless. Consider for instance that B is true (the bicycle is missing). We cannot logically conclude that A is true (the bicycle is stolen) because A does not follow from B: After all, it *might* be the case the bicycle is *not* stolen, but simply moved by accident or because someone had to use the space where the bicycle was parked. In reality, outside our four senses there are nearly no non-trivial relationships or things we can know with absolute certainty. This does not mean we don't *learn* something by observing the bicycle is missing: When we learn our bicycle is missing, we become *more certain* that our bicycle is stolen than we would be before checking. When a scientist experimentally examine a theory she becomes *more confident* the theory is true and evidence in a trial will move the jury to be *more* or *less convinced* of the guilt of the accused. In all these cases we must (and do!) reason under uncertainty which mean we must reason using different rules than in classical logic. As we will see, probability theory is the uniquely optimal way to reason under uncertainty and it is for this reason probabilities play such a central role in machine learning.

5.2 Introduction to Bayes' theorem

Most readers are familiar with probabilities; however, probabilities are often confused with other concepts and will therefore be introduced anew here. To begin, we will consider binary propositions such as A, B or C similar to the bicycle example. We then define the *probability* of a proposition as our degree-of-confidence the proposition is true. For instance, the probability A is true or the probability that A and B are both true may be written as

$$P(A) \quad \text{and} \quad P(AB)$$

respectively and the probability A is true given C is true or that A and B are both true given C may be written as the *conditional probability*:

$$P(A|C) \quad \text{and} \quad P(AB|C).$$

All probabilities are numbers between 0 and 1 such that $P(A) = 0$ corresponds to the case we are absolutely certain A is false and $P(A) = 1$ the case we are absolutely certain A is true. In the example with the bicycle, we can express the idea that finding the bicycle missing increases our confidence that it is stolen:

$$P(A|B) > P(A).$$

Now to some good news: There are only two rules in probability theory and everything follows from these rules [1]. They are

[1] For completeness we should mention this is only true for finite sets of propositions. In general the sum rule is $\sum_{i=1}^{\infty} P(A_i) = 1$ assuming A_i's are mutually exclusive and exhaustive. This is however irrelevant for our purpose.

$$\textit{The sum rule:} \qquad P(A|C) + P(\overline{A}|C) = 1 \tag{5.1}$$

$$\textit{The product rule:} \qquad P(AB|C) = P(B|AC)P(A|C) \tag{5.2}$$

(we may at times omit C for simplicity). 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. In the following sections, we will explore some of the way these rules can be applied.

5.2.1 Bayes' theorem

Let's begin by deriving Bayes' theorem as an example of how non-trivial results can be obtained from just the sum and product rule. Notice how the following are just applications of the sum and product rule:

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

If we then apply the product rule twice we obtain

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

The two right hand expressions can thus be set equal, and we can divide both expressions by $P(B|C)$ to obtain:

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

Example 1: The taxicab accident

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

> 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?

Try to make your best guess at the solution before reading on. To solve the problem, lets define two relevant variables:

$$B : \textit{The delinquent was a Blue cab.}$$
$$W : \textit{The Witness reported the car was blue.}$$

Since cabs can only be green and blue, $\overline{B}$ is the event the cab is blue. We are interested in computing the probability the cab was Blue given the witness said it was blue $P(B|W)$. Using Bayes theorem this is

$$P(B|W) = \frac{P(W|B)P(B)}{P(W|B)P(B) + P(W|\overline{B})P(\overline{B})} \tag{5.3}$$
$$= \frac{0.8 \times 0.15}{0.8 \times 0.15 + 0.2 \times 0.85} \approx 41\% \tag{5.4}$$

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

5.2.2 Mutually exclusive events

Probability as considered so far is only defined for binary events and to expand its use to events with more outcomes, for instance numbers, requires some work. Fortunately, all this work can be accomplished only using the basic rules of probability theory! Suppose we roll an ordinary die. There are then six possible outcomes corresponding to the six events

$$\begin{aligned} A_1 &: \textit{The side } \boxdot \textit{ face up.} & A_2 &: \textit{The side } \boxdot \textit{ face up.} \\ A_3 &: \textit{The side } \boxdot \textit{ face up.} & A_4 &: \textit{The side } \boxdot \textit{ face up.} \\ A_5 &: \textit{The side } \boxdot \textit{ face up.} & A_6 &: \textit{The side } \boxdot \textit{ face up.} \end{aligned} \tag{5.5}$$

Obviously no two of these events can happen at once and they are said to be *mutually exclusive*, simply meaning A_iA_j can be known to be false if $i \neq j$. The sum rule takes a particular simple form for mutually exclusive events given in eq. (5.7) and the proof is given below for completeness.

Derivation of the sum rule*

Continuing with the dice example, let us consider the probability *either* A_1 or A_2 occurs and for simplicity we will write it using the symbols $A = A_1$ and $B = A_2$. That either A or B occurs will be written as the logical "or" $A + B$, and this is equivalent to

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

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

$$\begin{aligned} P(A + B) &= 1 - P(\overline{A}\ \overline{B}) && = 1 - P(\overline{A}|\overline{B})P(\overline{B}) \\ &= 1 - \left[1 - P(A|\overline{B})\right] P(\overline{B}) && = P(B) + P(A\overline{B}) \\ &= P(B) + P(\overline{B}|A)P(A) && = P(B) + \left[1 - P(B|A)\right] P(A) \\ &= P(A) + P(B) - P(AB). \quad . \end{aligned} \tag{5.6}$$

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 (mutually exclusive) $P(A_1A_2) = 0$ this simplifies to

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

And in general, for n mutually exclusive events we have

$$P(B_1 + B_2 + \cdots + B_n|C) = P(B_1|C) + P(B_2|C) + \cdots + P(B_n|C). \tag{5.7}$$

As a special case, consider the case the events are also *exhaustive*, i.e. $A_1 + \cdots + A_n = 1$. This is the case for the die where we know one of the six options $A_1, \ldots, A_6$ are true because one side is always facing up. In this case:

$$1 = P(A_1|C) + P(A_2|C) + \cdots + P(A_6|C)$$

This is actually a powerful result. Let B be any other event and consider the probability $P(B|C)$. We can always write:

$$B = B(A_1 + \cdots + A_n) = BA_1 + \cdots + BA_n$$

(since $A_1 + \cdots + A_n = 1$) and since BA_1 and BA_2 are still mutually exclusive we get:

$$P(BA_1 + \cdots + BA_n|C) = P(BA_1|C) + \cdots + P(BA_n|C)$$

Collecting this we have shown that

$$P(B|C) = P(BA_1|C) + \cdots + P(BA_n|C),$$

assuming $A_1, \ldots, A_n$ are mutually exclusive and exhaustive. This general procedure which we will use again and again is known as *marginalization*. Our first use of marginalization will be to generalize Bayes theorem: Suppose $A_1, \ldots, A_n$ is a set of mutually exclusive and exhaustive hypothesis and B is some piece of evidence, we then have

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

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]:

> 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 the following reasoning: *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 argument is compelling. Do you think it is true? To examine it, let us define the four variables:

$$A_1, A_2, A_3 : \textit{The car is behind door 1, 2 and 3 respectively}$$
$$R_{g3} : \textit{The host reveals a goat behind door 3}$$

The problem boils down to calculating $P(A_1|R_{g3})$, the probability the car is behind door 1 given we initially selected door 1 and the host revealed the goat behind door 3. Using the version of Bayes theorem with mutually exclusive hypothesis we get:

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

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

So we now 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.

5.3 Probability densities

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.

Let us consider a simple example. Suppose we denote by r the amount of rain that falls per day in Denmark, then clearly r is a random variable (different amounts of rain fall on different days). However, the problem is that it does not strictly speaking make sense to say the probability of r

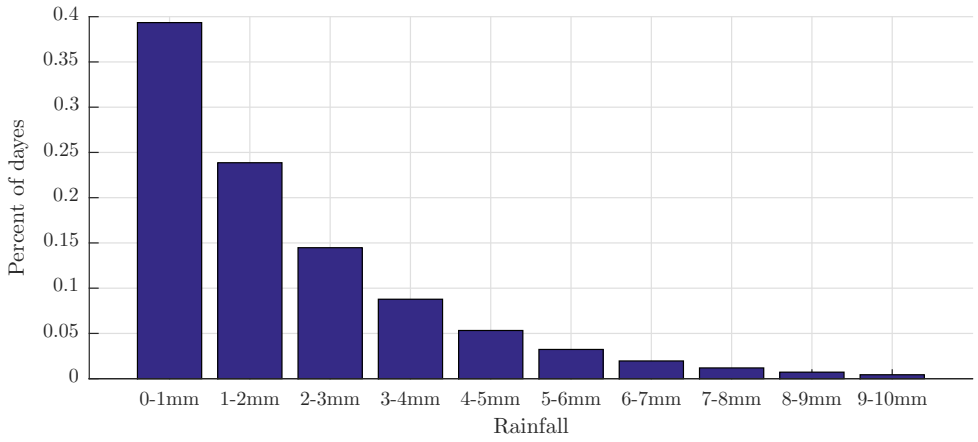


Fig. 5.1: The probability of rainfall per day illustrated as a histogram. Each bar denotes the event that on a given day there are between $0-1mm$ of rain, $1-2$mm of rain, $2-3$mm of rain etc. These well-defined binary events can be estimated from historical rainfall. records.

is anything specific – after all, suppose we ask: What is the probability there will be $r = 2.3$mm of rain a given day? The problem is why not ask for 2.31 or 2.3001 mm of rain? The chance there will be *exactly* $r = 2.3000000$mm of rain must be zero. The way we overcome this in practice is to rather ask *"what is the probability there will be between 2 and 3mm of rain"*? This is a proper question that can be formulated by introducing a general variable:

$$A_{[a,b]} : \textit{There will be between } a \textit{ and } b \textit{ mm of rain},$$

and then simply write $P(A_{[2,3]})$ for the probability which will be non-zero. In fig. 5.1 the probability of having different intervals of rainfall a given day in Denmark is tabulated and their respective probability is shown.

However, keeping track of these histograms are difficult. After all, suppose some ask for $P(A_{[2.5,3.5]})$? We can perhaps make a qualified guess at this variable (around 13%), however, we would like a way to represent *all* binary variables. This can be accomplished using a *probability density*. A probability density is simply a function p that is non-negative and integrates to one. We can then *define* the probability of a particular event such as $A_{[a,b]}$ as the integral

$$P(A_{[a,b]}) = \int_a^b p(x)dx. \tag{5.8}$$

In fig. 5.2 is shown the probability of the events $A_{[2,3]}, A_{[0,1]}$ and $A_{[5,\infty[}$ with their respective probabilities.

Let us now 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. 5.3)

$$P(A_{[x,x+dx]}) \approx p(x)dx.$$

In general, suppose we have two variables x, y. In direct generalization of the 1d case, the probability

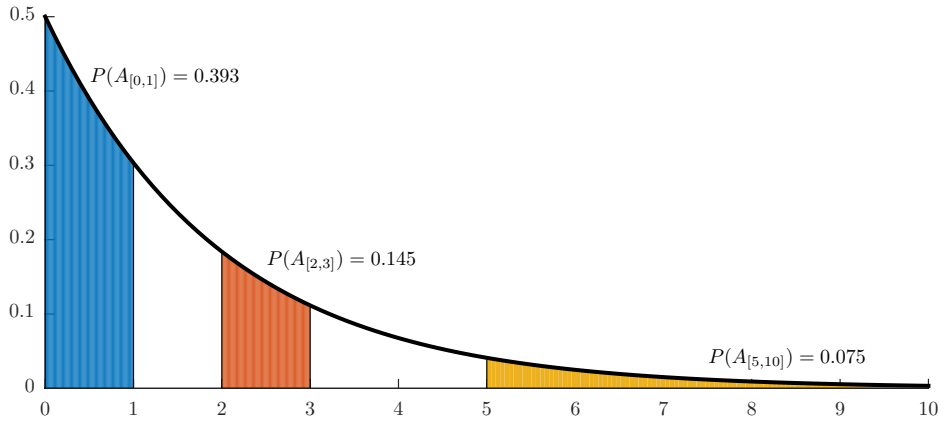


Fig. 5.2: The rainfall example continued. If we introduce a function $p(r)$, we can define the probability of the event $A_{[a,b]}$ (that there was between a and b mm of rainfall a given day) as $p(A_{[a,b]}) = \int_a^b p(r)dr$. The three colored regions thus correspond to three events, and each *area* corresponds to their probability.

that x, y both fall within some 2d subset D of $\mathbb{R}^2$ is then

$$P((x,y) \in D) = \int_{(x,y)\in D} p(x,y)dxdy \tag{5.9}$$

However similar to the 1d case, we can consider the event x lies in the interval $[x, x+dx]$ and y in the interval $[0, y+dx]$, $A_{[x,x+dx]}$ and $B_{[y,y+dy]}$, see fig. 5.4 where this corresponds to the red area. The ordinary product rule is

$$P(A_{[x,x+dx]}B_{[y,y+dy]}) = P(B_{[y,y+dy]}|A_{[x,x+dx]})P(A_{[x,x+dx]}) \tag{5.10}$$

However using that dx, dy are both small we can again approximate this as

$$P(A_{[x,x+dx]}B_{[y,y+dy]}) \approx dxdyp(x,y), \tag{5.11}$$
$$P(B_{[y,y+dy]}|A_{[x,x+dx]}) \approx dyp(y|x), \tag{5.12}$$
$$P(A_{[x,x+dx]}) \approx dxp(x), \tag{5.13}$$

5.3.1 Multiple continuous parameters

where $p(y|x)$ is a new function of two parameters. If we plug these definitions into eq. (5.10) and divide by $dxdy$ this can be written in the more familiar form:

$$p(x,y) = p(y|x)p(x)$$

We say that $p(x,y)$ is the *joint* distribution of x, y and $p(y|x)$ is the *conditional* distribution of y given x. In general, we can define the sum and product rules for continuous densities:

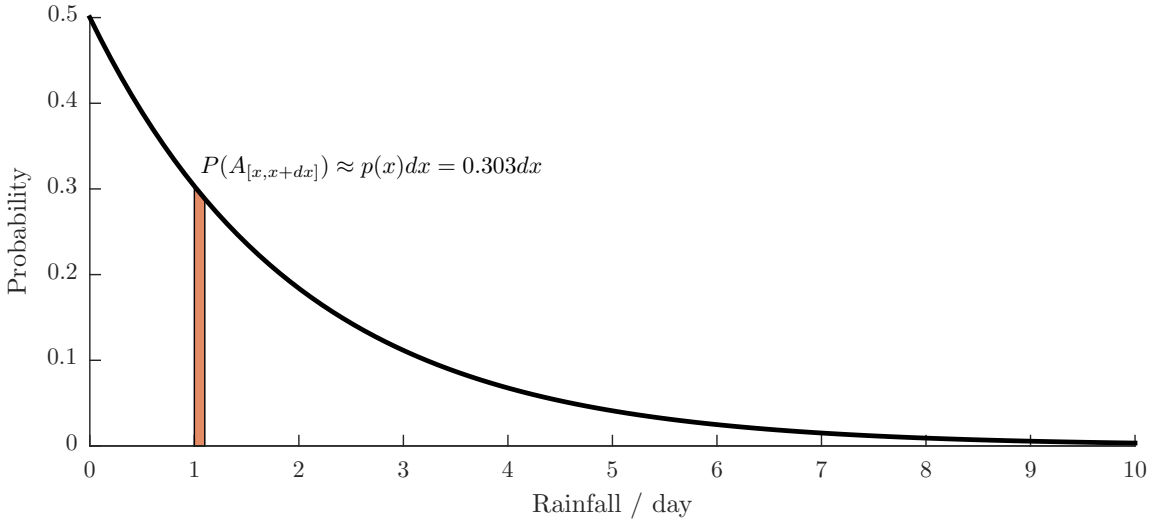


Fig. 5.3: Continuing further the rainfall example, rather than talking about the probability there will be exactly x mm of rain, we can talk about the probability there will be between x and $x + dx$ mm of rain. This can be approximated as $p(A_{[x,x+dx]}) \approx p(x)dx$ which becomes more and more exact when dx approaches 0.

$$\textit{The sum rule:} \qquad \int dx\; p(x|z) = 1, \tag{5.14}$$

$$\textit{The product rule:} \qquad p(x, y|z) = p(y|x, z)p(x|z), \tag{5.15}$$

where again we will often omit z for simplicity. 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:

$$\textit{Marginalization:} \qquad \int dx\; p(x, y|z) = p(y|z) \tag{5.16}$$

$$\textit{The product rule:} \qquad p(x, y|z) = p(y|x, z)p(x|z) \tag{5.17}$$

$$\textit{Bayes theorem:} \qquad p(x|y, z) = \frac{p(y|z)p(x|y, z)}{\int p(y|x', z)p(x'|z)dx'}. \tag{5.18}$$

5.4 Distributions and the central limit theorem

In the previous section we saw how the basic rules of probability could naturally be expanded to encompass densities, however at this point the theory is still fairly "content-less" since we have not yet seen examples of actual distributions or uses of probability theory. The first important concept is that of averages. Let's suppose x denotes a continious quantity of interest for instance the temperature one week from now and $p(x)$ is the probability density of x. As a shorthand we will

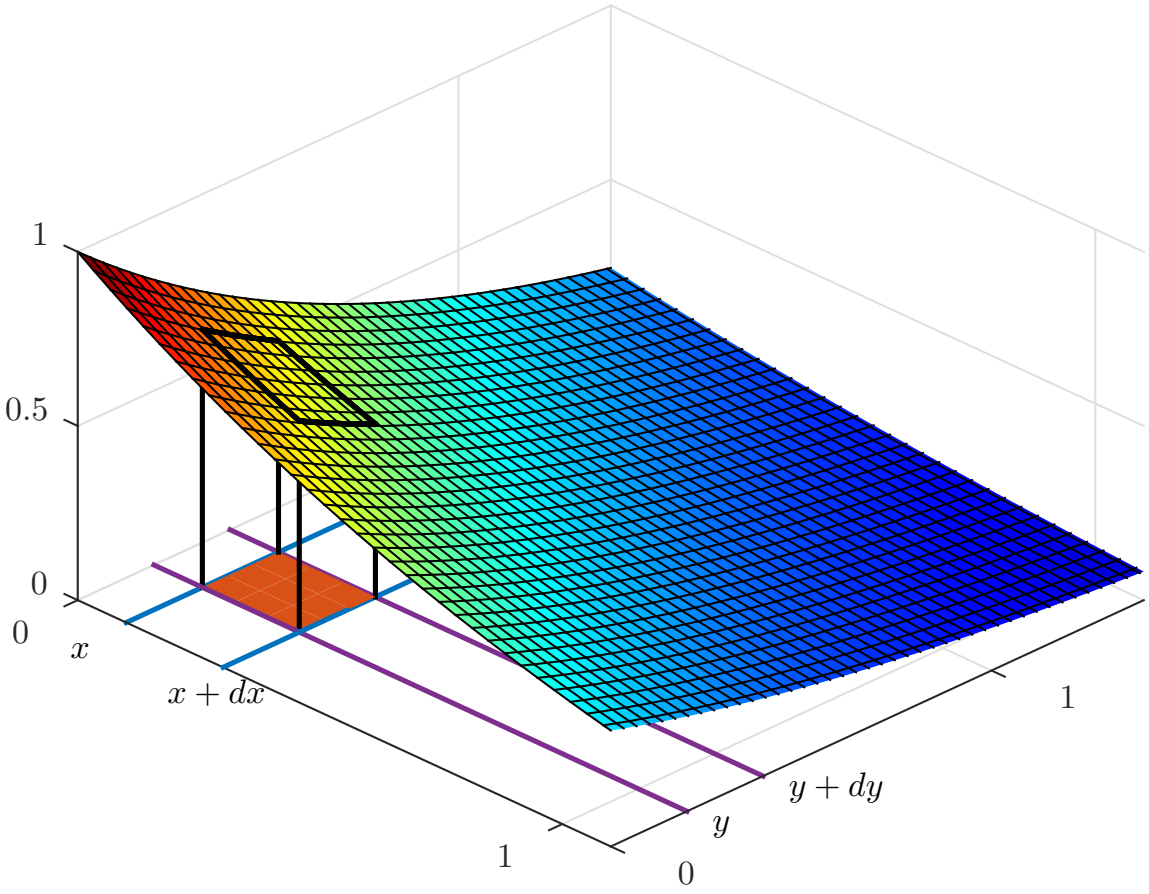


Fig. 5.4: Suppose we have a 2d density $p(x, y)$. For a subset $D \subset \mathbb{R}^2$ we can define the probability (x, y) lies in D as $p(A_D) = \int_{(x,y)\in D} p(x, y)dxdy$. For small but non-zero values of dx and dy, and the case where D is the rectangle $[x, x + dx] \times [y, y + dy]$ (the red area), the probability of D (the volume indicated by the black lines) can be approximated as $p(A_D) \approx p(x, y)dxdy$

sometimes write, say, $p(x = 3)$ for the probability (density) that x takes the value 3. If we consider any function f of x we can then define the expectation of a function of x as:

$$\mathbb{E}[f] = \int f(x)p(x)dx \tag{5.19}$$

If x is discrete and takes possible values $x_1, \ldots, x_n$ then the expectation is the sum:

$$\mathbb{E}[f] = \sum_{i=1}^{n} f(x_i)p(x_i)$$

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:

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

Similar equations can be obtained for the case where x is continuous by replacing the sum with an integral. These definitions are somewhat abstract so they are worth illustrating with a few 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 let's consider the *silly dice*. Suppose we take an ordinary dice, but instead of the numbers $1, \cdots, 6$ on the six sides we paint the sides with the numbers $-2, 1, 1, 4, 4$ and 10. When we roll the dice, it generates a random number (let's call it x) which takes four different values $(-2, 1, 4, 10)$ and the probability it comes out with any particular value is

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

It is a bit hard to visualize the density of a discrete variable but we have made an attempt in fig. 5.5. We can then compute the mean and variance of the silly dice as:

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

The following properties of the mean and variance are easy to show and will come in handy later:

$$\mathbb{E}[ax + b] = \sum_{i=1}^{n} (ax_i + b)p(x_i) = a\mathbb{E}[x] + b, \quad \text{Var}[ax + b] = a^2 \,\text{Var}[x].$$

Multidimensional expectations

Lets consider the more general case of a distribution of several variables, say, x, y. In this case the concept of expectation generalizes (we will assume x, y are continuous):

$$\mathbb{E}_{x,y}[f(x, y)] = \iint f(x, y)p(x, y)dxdy$$

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:

$$\begin{aligned} \mathbb{E}_{x,y}[x + y] &= \iint (x + y)p(x, y)dxdy = \iint xp(x, y)dxdy + \iint yp(x, y)dxdy = \int xp(x)dx + \int yp(y)dy \\ &= \mathbb{E}_x[x] + \mathbb{E}_y[y] \end{aligned} \tag{5.20}$$

This naturally generalize to the case where we have many variables.

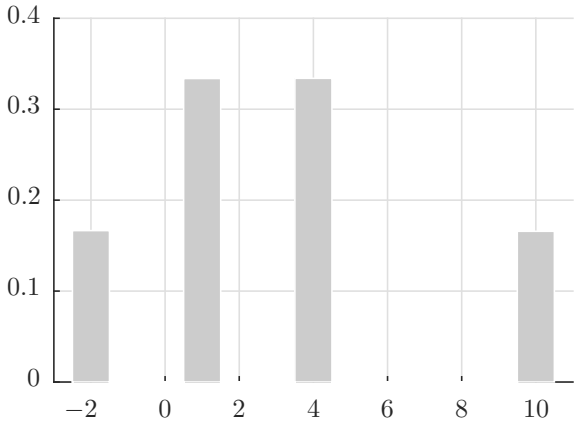


Fig. 5.5: The silly dice. The silly dice has four outcomes, $-2, 1, 4$ and 10 where $1, 4$ has been painted on two sides and therefore occurs twice as often as the other outcomes.

5.4.1 The Bernoulli and binomial distributions

In the case of the silly dice we knew (by symmetry of the dice) the probability it came out with any particular value. In general we want to *learn* the probability events happen and that involve treating probabilities as parameters. We will introduce this concept with the simplest example imaginable: binary events.

Suppose we let the variable b denote the outcome of a binary event such that $b = 0$ if the event did not take place and $b = 1$ if the event did take place. For instance the binary event could be a coin flip where $b = 0$ denote the event the coin landed tails and $b = 1$ the event the coin landed heads (we assume the coin might be *weighted* such that the chance it lands on head can be something else than 0.5) or alternatively b 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 event b happens ($b = 1$) with a probability which we will denote by θ and correspondingly the event does not happen with probability $1 - \theta$. Thus we can write the density of b 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 (the chance the event occurs) then the probability of b (which has two outcomes) is the expression on the right. Let's check this makes sense: $p(b = 0|\theta) = \theta^0(1-\theta)^{1-0} = 1 - \theta$ and $p(b = 1|\theta) = \theta^1(1-\theta)^{1-1} = \theta$. This distribution, which depends on the single parameter θ, is known as the *Bernoulli distribution*. This distribution may appear too simple to be worth any interest, however as we will see more elaborate distributions can be build from the Bernoulli distribution and it turns out to serve an important role in classification. For now let's simply compute the mean and variance of the Bernoulli distribution:

$$\mathbb{E}[b] = \sum_{b=0}^{1} bp(b|\theta) = 0 \times (1-\theta) + 1 \times \theta = \theta, \quad \text{Var}[b] = (0-\theta)^2 \times (1-\theta) + (1-\theta)^2 \times \theta = \theta(1-\theta).$$

Repeated events

Let's turn to a more elaborate example of 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 $b_1, \ldots, b_N$ each which occurs with a probability θ. What's the chance a particular sequence occured? Since all we know is the basic rules of probability this is where we should look to answer the question. For simplicity let's assume that $N = 3$ and we can then apply the product rule two times:

$$p(b_1, b_2, b_3|\theta) = p(b_3|b_2, b_1, \theta)p(b_2, b_1|\theta) = p(b_3|b_2, b_1, \theta)p(b_2|b_1, \theta)p(b_1|\theta)$$

Notice this is *only* applying the basic rules of probability with no "thinking". Now let's turn to particular information we have been given about the problem. Let's consider $p(b_2|b_1, \theta)$: This is *"the probability of* b_2 ***given*** *we know* θ ***and*** b_1*"*. But if we know θ (the chance b_2 occurs), b_1 does not give us any extra information and we can reason $p(b_2|b_1, \theta) = p(b_2|\theta)$. Similarly $p(b_3|b_2, b_1, \theta) = p(b_3|\theta)$ and we are thus left with: $p(b_1, b_2, b_3|\theta) = p(b_1|\theta)p(b_2|\theta)p(b_3|\theta)$. In general if we consider N events:

$$p(b_1, \cdots, b_N|\theta) = \prod_{i=1}^{N} p(b_i|\theta) = \prod_{i=1}^{N} \theta^{b_i}(1-\theta)^{1-b_i} = \theta^{\sum_{i=1}^{N} b_i}(1-\theta)^{N-\theta^{\sum_{i=1}^{N} b_i}}$$

When $p(b_2|b_1, \theta) = p(b_2|\theta)$ we say that b_2 is *independent* of b_1 given θ and the above equation is called a factorization of $p(b_1, \cdots, b_N|\theta)$.

Let's return to the case where b_i denoted if patient i was cured or not. In this setting, a doctor is more likely interested in *how many* of the N patients was cured (or not) than if any individual patient was cured. Suppose we introduce m as the number of positive outcomes of $b_1, \ldots, b_N$ then the probability of a sequence is:

$$p(b_1, \cdots, b_N|\theta) = \theta^m(1-\theta)^{N-m}, \quad m = b_1 + b_2 + \cdots + b_N \tag{5.21}$$

and we see the probability of a particular sequence of outcomes only depend on N and m. Suppose further we are only interested in the number of positive outcomes m given we know N and θ, i.e. $p(m|N, \theta)$. This can be computed using the sum rule and in particular eq. (5.7) for mutually exclusive events as the sum of all sequences with m positive outcomes, however the derivation is slightly complicated. It has been reproduced below for completeness but a reader may skip to eq. (5.23) on a first reading.

Derivation of the binomial distribution*

Suppose $N = 3$ and $m = 2$. Then the chance there are $m = 2$ positive outcomes is the sum of the three sequences with $m = 2$ positive outcomes:

$$\begin{aligned} p(m = 2|N, \theta) &= p(b_1 = 1, b_2 = 1, b_3 = 0|\theta) + p(b_1 = 1, b_2 = 0, b_3 = 1|\theta) + p(b_1 = 0, b_2 = 1, b_3 = 1|\theta) \\ &= 3\theta^2(1-\theta)^{3-2} = 3\theta^2(1-\theta) \end{aligned}$$

In general this can be computed to be:

$$\begin{aligned} p(m|N,\theta) &= \{\text{Sum of the probability of all sequences of length } N \text{ with } m \text{ positive outcomes}\} \\ &= \begin{Bmatrix}\text{Number of sequences of length } N \\ \text{with } m \text{ positive outcomes}\end{Bmatrix} \times \begin{Bmatrix}\text{Probability of a sequence} \\ \text{with } m \text{ positive outcomes}\end{Bmatrix} \\ &= \begin{Bmatrix}\text{Number of sequences of length } N \\ \text{with } m \text{ positive outcomes}\end{Bmatrix} \times \theta^m(1-\theta)^{N-m} \end{aligned} \tag{5.22}$$

What is the number of sequences of length N with m positive outcomes? Suppose we consider a different problem of counting all the ways we could arrange the N variables $b_1, \cdots, b_N$ in a sequence. I.e. for $N = 4$:

$$(b_1, b_2, b_4, b_3),\ (b_4, b_3, b_2, b_1),\ (b_2, b_3, b_1, b_4), \cdots$$

There are N ways to select which b_i should go in the first place, then $N-1$ ways to select which of remaining $N-1$ b_i's should go in the second place and so on. All in all there are $N \times (N-1) \times (N-2) \times \cdots \times 2 \times 1 = N!$ such sequences. Suppose that m of the b_i's are 1 and $N-m$ are 0 and we denote the number of ways to select which m are 1 by $\binom{N}{m}$ (this is the number we are interested in), we can count the number of sequences in a different manner: The total number of sequences can be obtained by first selecting which m of the b_i's should go in the first m places in the sequence, then multiply this by the number of ways the first m elements can be permuted and finally the number of ways the last $N-m$ elements can be permuted. Accordingly:

$$\begin{aligned} \{\text{ Total sequences of } N\ b_i\text{'s}\} = \{&\text{ Ways to select which } b_i\text{'s go in first } m \text{ places}\} \\ &\times \{\text{Ways to arrange first } m \text{ selected } b_i\text{'s}\} \\ &\times \{\text{Ways to arrange last } N-m \text{ selected } b_i\text{'s}\} \end{aligned}$$

Plugging in this gives:

$$N! = \binom{N}{m} \times m! \times (N-m)!$$

And this implies that $\binom{N}{m} = \frac{N!}{m!(N-m)!}$. As a sanity check we get $\binom{3}{2} = \frac{3!}{2!1!} = 3$. If we plug this result into eq. (5.22) we get:

$$\text{Binomial distribution:}\quad p(m|N,\theta) = \binom{N}{m}\theta^m(1-\theta)^{N-m} \tag{5.23}$$

which is known as the *Binomial distribution*. This derivation is no doubt quite complicated, however it is important to stress it essentially only relied on the sum rule and a counting argument. In fig. 5.6 we have shown the probability density function of m computed from eq. (5.23) when $\theta = 0.7$ and $N = 4, 10$ and 60.

Properties of the binomial distribution

The mean and variance of the binomial distribution can be computed explicitly from eq. (5.23) however we can also just apply the sum rule of expectations eq. (5.20) to get that $\mathbb{E}[m] = \mathbb{E}[\sum_i b_i] = \sum_i \mathbb{E}[b_i]$ (and a similar relation hold for the variance). All in all this gives:

$$\mathbb{E}[m] = \sum_{m=0}^{N} m p(m|N,\theta) = N\theta, \quad \text{Var}[m] = \sum_{m=0}^{N} (m - N\theta)^2 p(m|N,\theta) = N\theta(1-\theta) \tag{5.24}$$

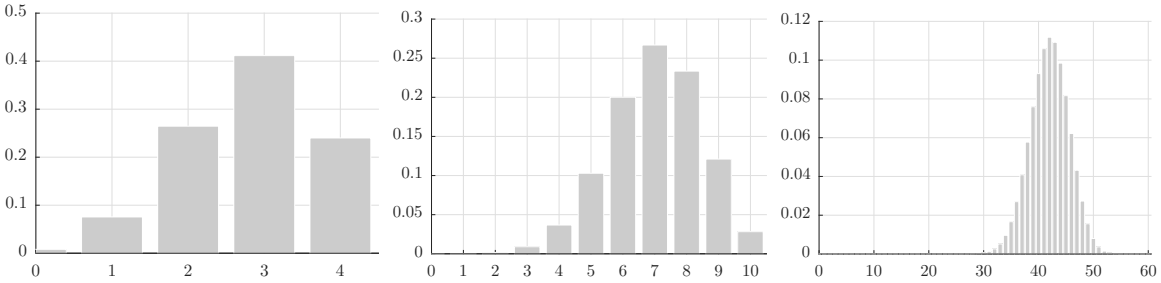


Fig. 5.6: The binomial distribution $p(m|N,\theta)$ for $N = 4, 10, 60$ and $\theta = 0.7$.

5.4.2 The central limit theorem*

As before, lets suppose we flip N weighted coins and count the number of heads m. The distribution of m is the Binomial distribution eq. (5.23) 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 this plot from arms length we see the mean value and variance grows as N and the square root of N respectively as dictated by eq. (5.24), however it also seems that m takes the form of a continuous smooth curve centered around the mean value $\mathbb{E}[m] = N\theta$. Suppose we are interested in what happens with the distribution of m when N is very large. Since the mean grows unbounded, let's instead consider the distribution of $\nu = \frac{m}{N}$. If $N = 4$ this can take 5 different values:

$$\nu: \quad 0, \ \frac{1}{4}, \ \frac{1}{2}, \ \frac{3}{4}, \ 1$$

corresponding to sequences of length $N = 4$ with $m = 0$ to $m = 4$ positive outcomes. Naturally the mean of ν must be θ because $\mathbb{E}[\nu] = \mathbb{E}[\frac{m}{N}] = \frac{1}{N}\mathbb{E}[m] = \theta$ and the variance $\mathrm{Var}[\nu] = \frac{1}{N^2}\mathrm{Var}[m] = \frac{\theta(1-\theta)}{N}$. In fig. 5.7 is shown histograms of ν when $\theta = 0.7$ and $N = 4, 10, 60$ as before. The red lines is the density of the normal distribution with mean θ and variance $\sigma^2 = \frac{\theta(1-\theta)}{N}$

$$\mathcal{N}(x|\theta,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\theta)^2}{2\sigma^2}}$$

We see that as we compute the average of longer and longer sequences of Bernoulli variables the probability that the average number of positive outcomes takes a particular value can be approximated by a normal distribution. In other words, if N is really large we can use the normal distribution as a replacement for the Bernoulli distribution.

We might imagine this result has to do with the particulars of the Binomial distribution, but it is much more general. The reason why we encountered the normal distribution in the example is not specific to the binomial distribution, but rather it occurred because in the binomial distribution the variable m is a *sum of independent* variables and therefore μ is an *average* of independent variables. As we will see, whenever we encounter averages of N variables, and when N is large enough, the average will be approximately normally distributed a central fact of statistics known as the *central limit theorem*. This theorem is so important it is worth checking the normal distribution does indeed occur when we consider something else than Binomial distribution. Let's therefore return to the example of the silly coin from fig. 5.5. Suppose we roll a silly dice N times and compute the mean value of the roll ν. For instance if $N = 3$ and we consider two example roll sequences:

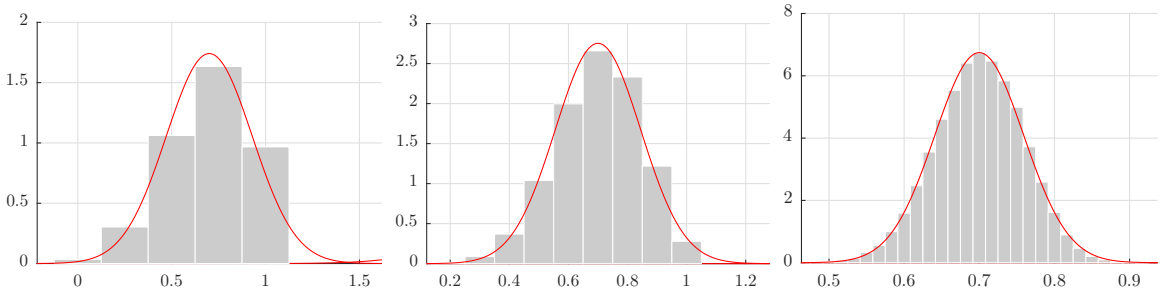


Fig. 5.7: The distribution of $\nu = \frac{m}{N}$ where m follows a binomial distribution $p(m|N,\theta)$ for $N = 2, 10, 60$ and $\theta = 0.7$. The inserted red curve is the normal approximation.

$$\text{Roll sequence 1: } \begin{bmatrix}10 & -1 & 4\end{bmatrix}, \qquad \text{mean: } \nu = \frac{13}{3} \tag{5.25}$$

$$\text{Roll sequence 2: } \begin{bmatrix}4 & 2 & 1\end{bmatrix}, \qquad \text{mean: } \nu = \frac{7}{3} \tag{5.26}$$

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 let's suppose we simulate many realizations of ν and use these draw $p(\nu|N)$ for different N. In fig. 5.8 the distribution is shown the distribution for $N = 2, 10, 60$ along with the normal densities:

$$\mathcal{N}(x|\mu,\sigma^2), \text{ with } \mu = \{\text{Mean of a single dice}\} = 3, \quad \sigma^2 = \frac{\{\text{Variance of a single dice}\}}{N} = \frac{14}{N}$$

That the normal densities come out with such nice formulas for the mean and variance is no surprise. In fact that this exactly the content of the central limit theorem: Suppose we have a random variable (a single dice) x with mean μ and variance σ^2. If we then generate N realizations of this random variable, $x_1, \ldots, x_N$, and compute the mean:

$$m = \frac{1}{N}\sum_{i=1}^{N} x_i$$

then the new random variable m will be approximately normally distributed: $p(m) \approx \mathcal{N}\left(m|\mu, \frac{\sigma^2}{N}\right)$. 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 op in all kinds of circumstances because most quantities are (in one way or another) representative of average effects.

5.4.3 The normal and multivariate normal distribution

We already encountered the normal probability density in the previous section as the density

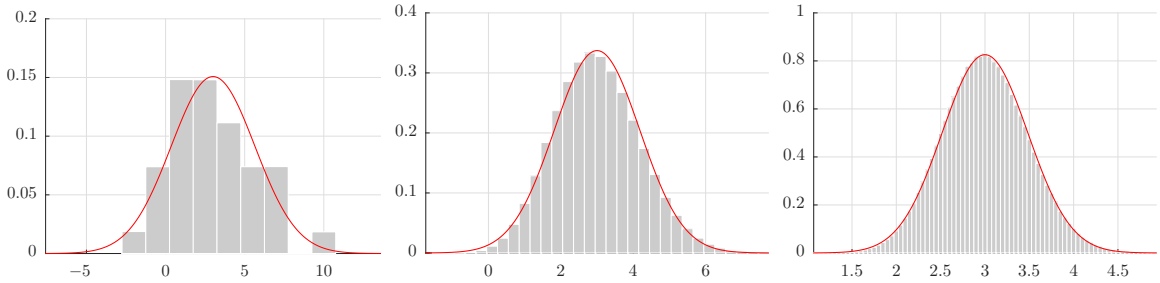


Fig. 5.8: The average of N rolls of the silly dice for $N = 2, 10, 60$ and inserted normal approximations. Notice average of the rolls converge rapidly to the normal distribution.

$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

The normal distribution has the familiar bell curve we already encountered in fig. 5.7 and fig. 5.8 and has mean and variance $\mathbb{E}[x] = \mu$, $\text{Var}[x] = \sigma^2$. The normal distribution can be generalized to the multivariate normal distribution which is a distribution over d-dimensional vectors

$$\boldsymbol{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_d \end{bmatrix}^T$$

and which is defined by the density:

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^d |\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$$

where $\boldsymbol{\Sigma}$ is known as the covariance matrix which must be symmetric and positive definite and $|\boldsymbol{\Sigma}|$ is the determinant of $\boldsymbol{\Sigma}$. The determinant quantifies the volume of the column-vectors of $\boldsymbol{\Sigma}$ and in 2d it corresponds to the cross-product, i.e. $\left|\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right| = ad - bc$. $\boldsymbol{\mu}$ is known as the mean of the multivariate normal distribution. In fig. 5.9 is shown the same multivariate normal distribution corresponding to

$$\boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix} \quad \text{and} \quad \boldsymbol{\mu} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Notice the distribution for this choice of $\boldsymbol{\mu}$ is centered on $(0, 0)$; this is no accident. For a general probability density p we can define the covariance matrix $\boldsymbol{C}$ as:

$$C_{ij} = \text{cov}(x_i, x_j) = \mathbb{E}_{\boldsymbol{x}}\left[(x_i - \mathbb{E}[x_i])(x_j - \mathbb{E}[x_j])\right] = \int (\, x_i - \mathbb{E}[x_i])(x_j - \mathbb{E}[x_j]) p(\boldsymbol{x}) d\boldsymbol{x}.$$

For the special case of the multivariate normal distribution, it holds that

$$\boldsymbol{\mu} = \mathbb{E}[\boldsymbol{x}] \quad \text{and} \quad \boldsymbol{\Sigma} = \boldsymbol{C}.$$

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. 5.10 where in each instance the mean is chosen as $\boldsymbol{\mu} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$ and

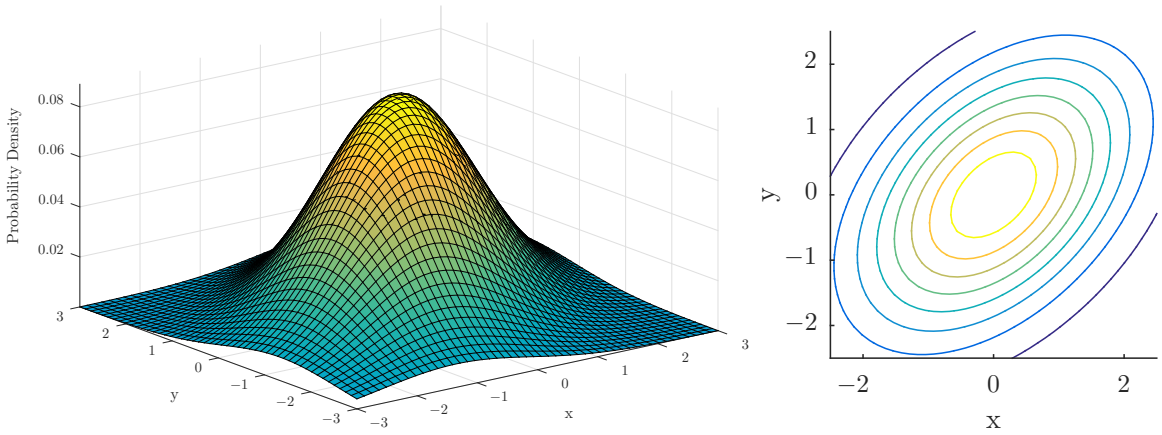


Fig. 5.9: Example of the probability density function of a 2d multivariate normal distribution. In left it is plotted as a function of $\boldsymbol{x} = [x\ y]^T$, i.e. $p(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$, whereas on the right the same distribution is shown as a contour plot.

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 0.2 & 0 \\ 0 & 1 \end{bmatrix} \qquad \boldsymbol{\Sigma}_3 = \begin{bmatrix} 1 & 0 \\ 0 & 0.2 \end{bmatrix} \tag{5.27a}$$

$$\boldsymbol{\Sigma}_2 = \begin{bmatrix} 1 & 0.7 \\ 0.7 & 1 \end{bmatrix} \qquad \boldsymbol{\Sigma}_4 = \begin{bmatrix} 1 & -0.7 \\ -0.7 & 1 \end{bmatrix} \tag{5.27b}$$

Let us consider one final example where we vary the off-diagonal elements (see fig. 5.11) corresponding to

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \boldsymbol{\Sigma}_2 = \begin{bmatrix} 1 & 0.45 \\ 0.45 & 1 \end{bmatrix} \qquad \boldsymbol{\Sigma}_3 = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}. \tag{5.28}$$

Notice, the distribution becomes more "skewed" when the off-diagonal elements are increased. As one more example, consider the case where $\boldsymbol{\Sigma}$ is diagonal

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_d^2 \end{bmatrix},$$

then

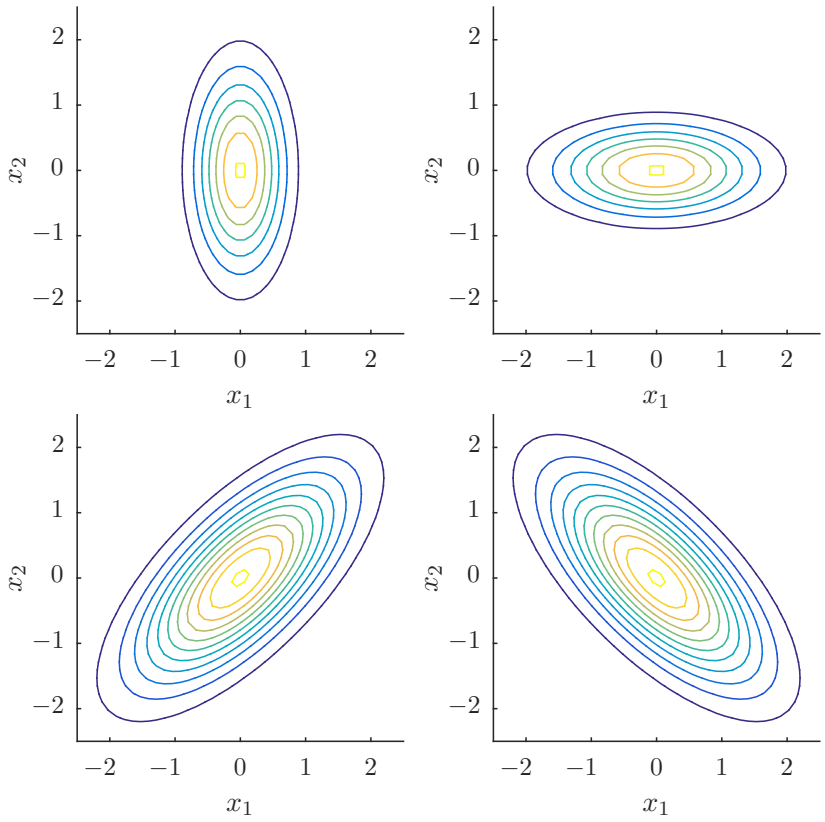


Fig. 5.10: Example of multivariate Gaussians when the covariance matrix is given as $\boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2, \boldsymbol{\Sigma}_3$ and $\boldsymbol{\Sigma}_4$ in eq. (5.27). When the covariance matrix is diagonal, the multivariate Gaussians are "cigars" oriented along either of 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.

$$
\begin{aligned}
p(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) &= \frac{1}{\sqrt{(2\pi)^d |\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})} \\
&= \frac{1}{\sqrt{(2\pi)^d \prod_{i=1}^d \sigma_i^2}} e^{-\sum_{i=1}^d \frac{1}{2}(x_i-\mu_i)\sigma_i^{-2}(x_i-\mu_i)} \\
&= \prod_{i=1}^d \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{(x_i-\mu_i)^2}{2\sigma_i^2}} \\
&= \prod_{i=1}^d p(x_i|\mu_i, \sigma_i^2)
\end{aligned}
$$

Thus, in this case the multivariate normal distribution is just a product of normal univariate normal distributions, i.e. the x_i's are independent.

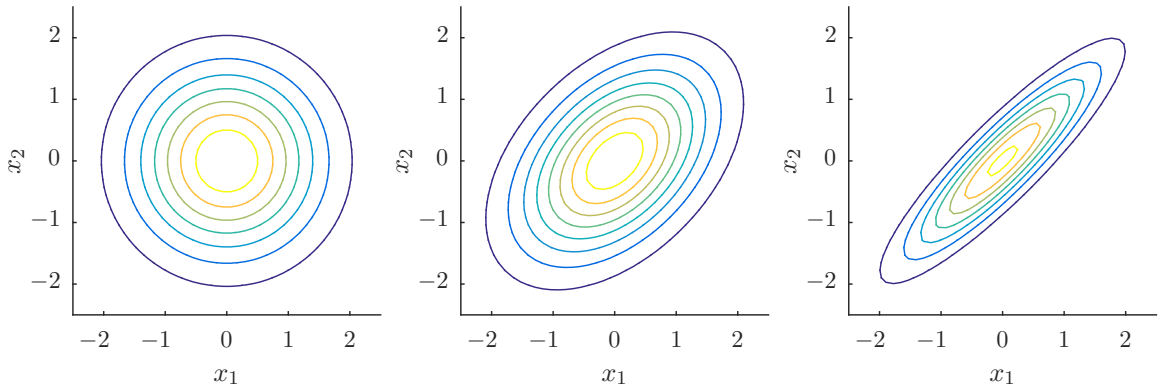


Fig. 5.11: Example of multivariate Gaussians when the covariance matrix is given as $\boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2$ and $\boldsymbol{\Sigma}_3$ in eq. (5.28). Increasing the off-diagonal elements increases the covariance.

5.5 Bayesian probabilities and machine learning

In the previous section, we derived the basic rules of probability theory. However, it is not obvious why these are useful for actual machine learning tasks. 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 three 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 don'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. Suppose we let $b_i = 0$ be the event the coin come up tails in flip i and $b_i = 1$ the event the coin came up heads in flip i. We can then represent the sequence *heads, tails, heads, heads* as

$$b_1 = 1,\ b_2 = 0,\ b_3 = 1,\ b_4 = 1,$$

and so on. For convenience we will collect a sequence of flips into a vector $\boldsymbol{b} = \begin{bmatrix} b_1 & b_2 & b_3 & b_4 \end{bmatrix}$. Phrased in this way what we are interested in is learning the chance the coin comes up heads in a given flip and we thus introduce the variable θ which goes from 0 to 1:

$$\theta : \text{The probability the coin comes up heads}$$

Let's make some preliminary observations. Firstly, what would be a common-sense response to this task? Well, there are four flips and it came up heads three times. So perhaps we could guess that

$\theta = \frac{3}{4}$. In general, suppose we observe N flips and it come up heads m times, then a good guess would be $\theta = \frac{m}{N}$.

We might object that this response does not take into account the number of flips are limited. For instance, suppose that θ was actually 0.5 (i.e. it was not a magic coin at all!), then the sequence $\boldsymbol{b}$ of four flips would still not be very unexpected, and our answer should take this into account. We are in other words *uncertain* about what θ is to begin with, and we remain somewhat *uncertain* even after observing N flips. However if we are uncertain about θ we should represent this with a probability. We therefore follow the program from before and write up a probability distribution for both θ and $\boldsymbol{b}$: $p(\boldsymbol{b}, \theta)$. We don't know what this probability distribution is yet, however we can apply the product rule to this probability distribution to get:

$$p(\boldsymbol{b}, \theta) = p(\boldsymbol{b}|\theta)p(\theta).$$

Lets first focus on the first term $p(\boldsymbol{b}|\theta)$. This probability is the chance of getting the sequence $\boldsymbol{b} = \begin{bmatrix} b_1 & b_2 & \cdots & b_N \end{bmatrix}$ **given** the probability the coin will come up heads is θ. We already encountered this expression in our discussion of the binomial distribution eqs. (5.21) and (5.23) and know:

$$p(\boldsymbol{b}|\theta) = \prod_{i=1}^{N} p(b_i|\theta) = \theta^m \theta^{N-m} \tag{5.29}$$

Let's turn our attention to the second term, $p(\theta)$. This is the probability density of θ when we don't know what sequence came up, i.e. our a-priori knowledge about what θ might be. One line of reason to determine this value could be this: Suppose we don't know what θ is. Then the chance θ is in (say) the interval $A = [0.3, 0.3 + 0.0001[$ and $B = [0.8, 0.8 + 0.0001[$ of the same length should be the same and so $P(A) = P(B)$. But by definition of the density $P(A) \approx p(0.3)dx, P(B) \approx p(0.8)dx$ for $dx = 0.0001$ and so $p(\theta = 0.3) = p(\theta = 0.8)$. This have to hold for any intervals and the only density that satisfy this is $p(\theta) = 1$. We can plug in these expressions and get:

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

What we are interested in is θ given we observe the particular sequence *heads, tails, heads, heads*. This is simply given by Bayes' theorem:

$$p(\theta|\boldsymbol{b}) = \frac{p(\boldsymbol{b}|\theta)p(\theta)}{p(\boldsymbol{b})} = \frac{p(\boldsymbol{b}|\theta)p(\theta)}{\int_0^1 p(\boldsymbol{b}|\theta')p(\theta')d\theta'}.$$

The integral in the denominator can be computed to be[2]:

$$p(\boldsymbol{b}) = \int_0^1 p(\boldsymbol{b}|\theta)p(\theta)d\theta = \int_0^1 p(\boldsymbol{b}|\theta)\theta^m(1-\theta)^{N-m}d\theta = \frac{m!(N-m)!}{N!} \tag{5.30}$$

if we plug this in we obtain

$$p(\theta|\boldsymbol{b}) = \frac{N!}{m!(N-m)!}\theta^m(1-\theta)^{N-m}. \tag{5.31}$$

[2] See for instance `https://en.wikipedia.org/wiki/Beta_function`

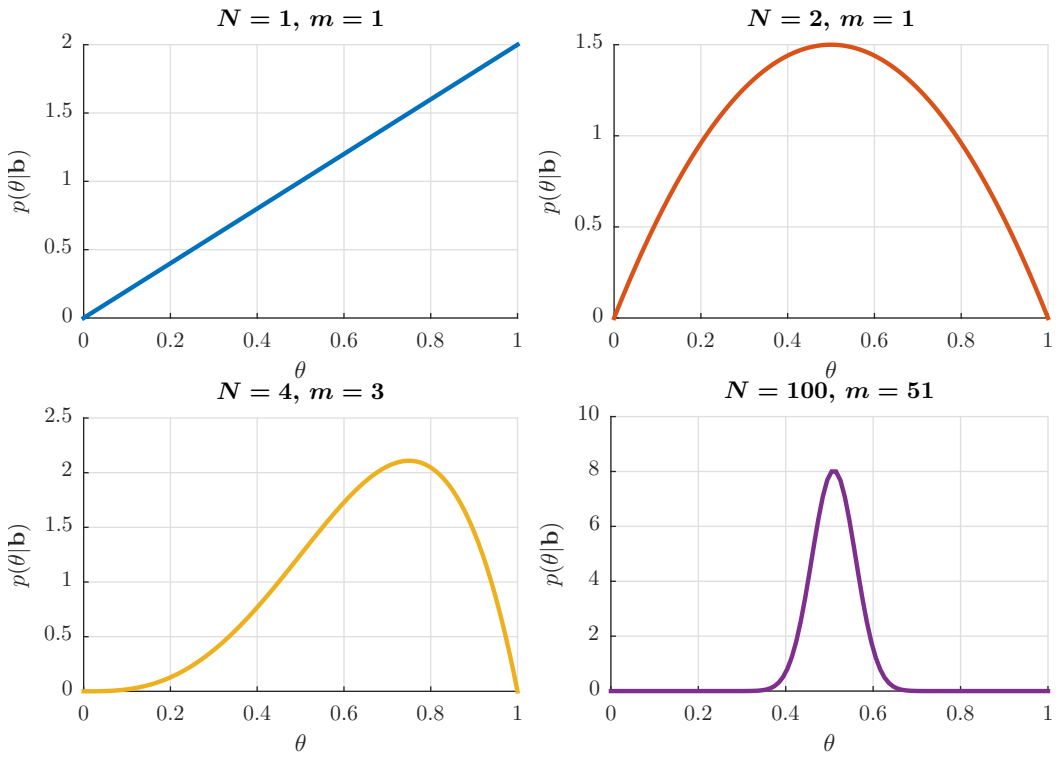


Fig. 5.12: Examples of the posterior probability $p(\theta|\boldsymbol{b})$ from eq. (5.31) for different numbers of flips N and different number of heads m. The top left figure corresponds to *heads*, the top-right to *heads, tails*, bottom left to *heads, tails, heads* and bottom right to $N = 100$ flips where $m = 51$ came up heads.

In fig. 5.12 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:

$$P(\theta \text{ is between } 0.4 \text{ and } 0.6|N = 4, m = 3) = \int_{0.4}^{0.6} p(\theta|\boldsymbol{b}) \approx 0.25$$

whereas for $N = 100, m = 51$ this chance is more than 0.95!. We might not feel this answer is as satisfying as a single number ("θ is really 0.5"), or an interval as usually seen in ordinary 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 include these statements as special cases.

In general, learning some parameters $\boldsymbol{w}$ in Bayesian probability theory from data $\boldsymbol{X}$ is thus nothing more than writing up the probability density:

$$p(\boldsymbol{X}, \boldsymbol{w})$$

and then compute the probability density of $\boldsymbol{w}$ given $\boldsymbol{X}$ using Bayes' theorem,

$$p(\boldsymbol{w}|\boldsymbol{x}) = \frac{p(\boldsymbol{X}|\boldsymbol{w})p(\boldsymbol{w})}{\int p(\boldsymbol{X}|\boldsymbol{w}')p(\boldsymbol{w}')d\boldsymbol{w}'},$$

and in the remainder of this book many of the models we will see will do this either explicitly or approximately.

5.5.1 Deciding between different models*

Let us suppose your friend and you disagree about whether the coin is magic or not and you have narrowed it down to two options. Your friend says that the coin is a magic coin with a probability θ of coming up heads (which he nevertheless do not know), but you are skeptical and believe the coin is just an ordinary coin that come up heads half the time. The two hypothesis are therefore:

h_1 : The coin come up heads with an (unknown) probability θ
h_2 : The coin come up heads half the time

How likely is h_1 given some sequence $\boldsymbol{b}$ of coin flips? We can easily compute this symbolically at least as

$$p(h_1|\boldsymbol{b}) = \frac{p(\boldsymbol{b}|h_1)p(h_1)}{p(\boldsymbol{b}|h_1)p(h_1) + p(\boldsymbol{b}|h_2)p(h_2)}$$

Suppose you agree to be initially unbiased against either two hypothesis $p(h_1) = p(h_2) = \frac{1}{2}$. We already computed $p(\boldsymbol{b}|h_1)$ in eq. (5.30), and $p(\boldsymbol{b}|h_2) = \frac{1}{2^N}$, because either outcome happens with probability $\frac{1}{2}$. This gives:

$$p(h_1|\boldsymbol{b}) = \frac{\frac{m!(N-m)!}{N!}}{\frac{m!(N-m)!}{N!} + \frac{1}{2^N}}. \tag{5.32}$$

To gain insight in this expression we generate a long sequences of up to $N = 200$ flips using a fair coin, thus conforming to hypothesis h_2. The value of $\frac{m}{N}$ as well as $p(h_1|\boldsymbol{b})$ computed from eq. (5.32) is shown in fig. 5.13 (left pane). We see that as more coin flips are included the probability h_1 is true, the coin is a magic coin, decreases because the coin is actually fair – thus the more parsimonious answer wins out. On the other hand, in the right pane we repeated the same experiment but where the sequence was generated from a "magic" coin with probability 0.65 of coming up heads. In this case h_1 eventually wins because, even though the flips *could* come from a fair coin, it eventually becomes more probable they came from a magic coin. Notice there is significant fluctuations in both cases when N is low due to random fluctuations in $\frac{m}{N}$.

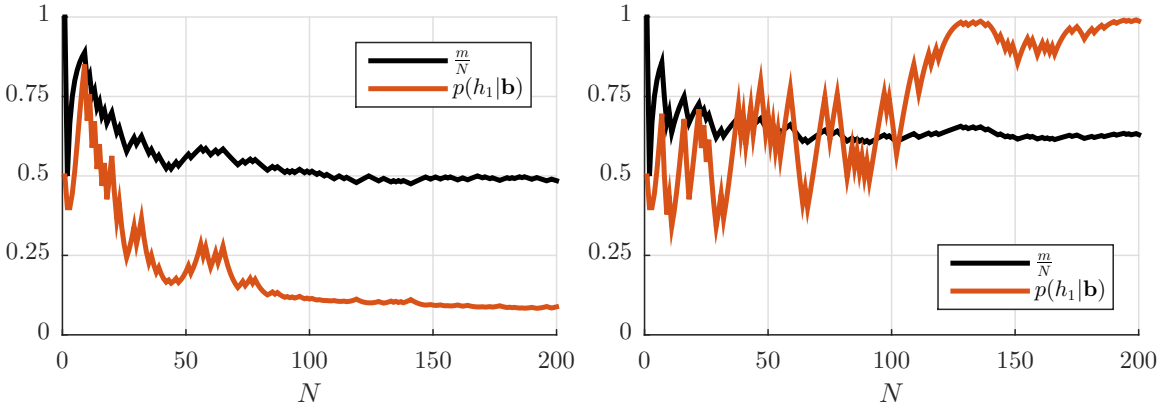


Fig. 5.13: (left:) For a simulated dataset of up to $N = 100$ coin flips from a fair coin we show $\frac{m}{N}$ (as expected converging to $\frac{1}{2}$) and the probability the coin is magic given a limited number of observations computed using eq. (5.32). As seen, with more and more samples we can rule out the "magic coin" hypothesis. (Right:) Same example but where the dataset is generated from a magic coin.

Problems

5.1. Fall 2014 question 8: 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.

5.2. Spring 2013 question 17: 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.

5.3. Fall 2013 question 15: Based on Haberman's Survival Data found in Table 5.1 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 5.1: 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.

6

Data Visualization

6.1 Visualization techniques

Visualization plays three important roles in machine learning and data mining

Exploratory data analysis Where the goal is to visualize the data in order to discover new patterns or relationships in the data.

Dissemination Where the goal is to communicate some form of result or point to a reader through a graphical element. This is used in technical writing.

Debugging Where often it can be very helpful when implementing a machine-learning algorithm to let it plot intermediate results or other measures for what it does. This often allows much faster discovery of issues with the implementation.

In this section, we will primarily focus on the second item, dissemination of results. However, the various plots we will go over will be useful for the other tasks as well.

6.1.1 The basic plots

To begin our discussion of plots we will briefly go over a few common types of visualization.

Visualization of a single attribute

Suppose we wish to visualize a single attribute. As an example, we will consider each of the four attributes of the Fisher Iris dataset. Likely, the most well-known way of visualizing a single attribute is the histogram. We construct the histogram in two steps. The first step is to divide the entire range of value of the variable into a series of intervals, 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 rectangle. 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. 6.1 is shown the histograms of all four attributes of the Iris dataset where we have scaled the x-axis to have the same length. 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

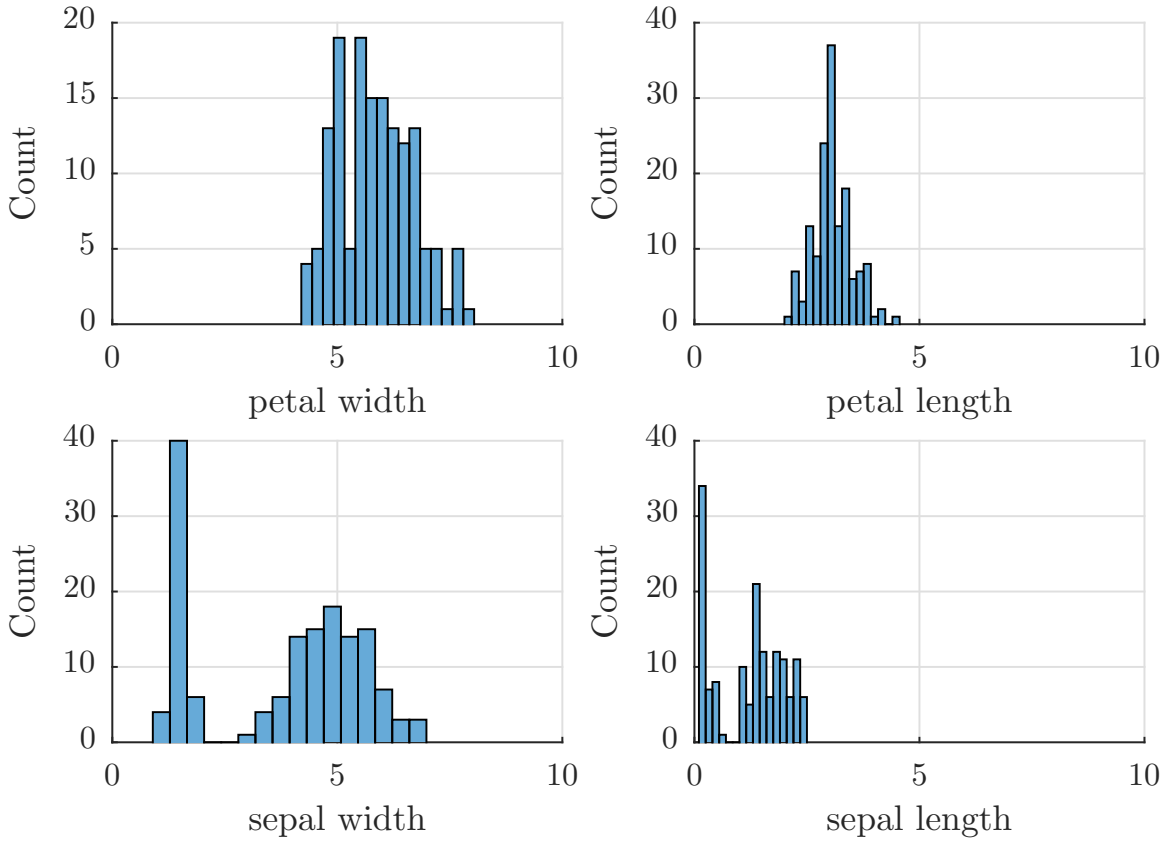


Fig. 6.1: Histograms based on $N = 16$ bins of the four features in the Iris dataset.

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 way to summarize a histogram is the box plot.

The boxplot is shown in fig. 6.2, 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:

$$\textit{upper whisker:}\ \min(l_{75} + \frac{3}{2}(l_{75} - l_{25}), v_N), \tag{6.1}$$

$$\textit{lower whisker:}\ \max(l_{25} - \frac{3}{2}(l_{75} - l_{25}), v_1), \tag{6.2}$$

where v_N detotes the value of the largest observation, and v_l 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.

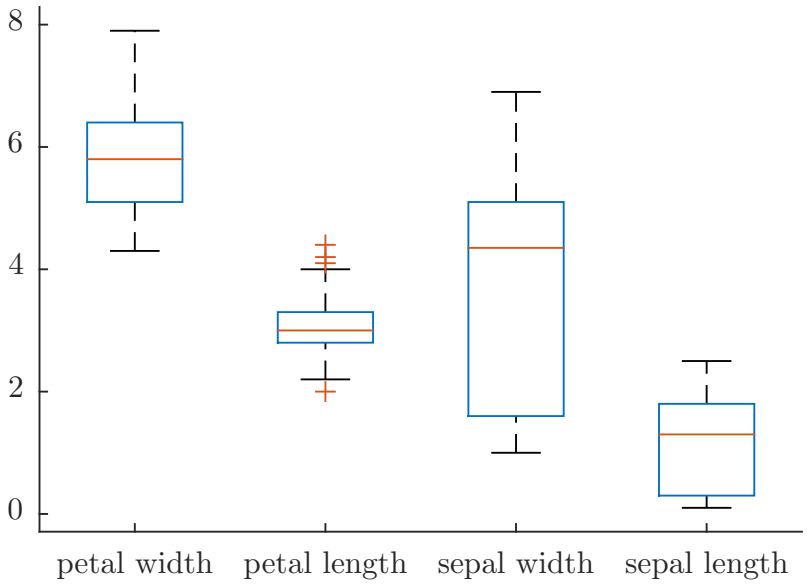


Fig. 6.2: Boxplots of the four attributes of the Iris dataset. Compare to the histograms in fig. 6.1 and notice how it is for instance possible to see how the bimodality of the sepal width creates an asymmetric boxplot.

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. 6.3 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 tends 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 have also increased the line width slightly. In all four charts, we use grid lines to guide the readers eye making it easier to compare the first months to the last.

Many one-dimensional series

Suppose we tests four different models on eight datasets and for each model we obtain a performance rating (the accuracy). We believe our method (Method 1) is the better. How do we best communicate

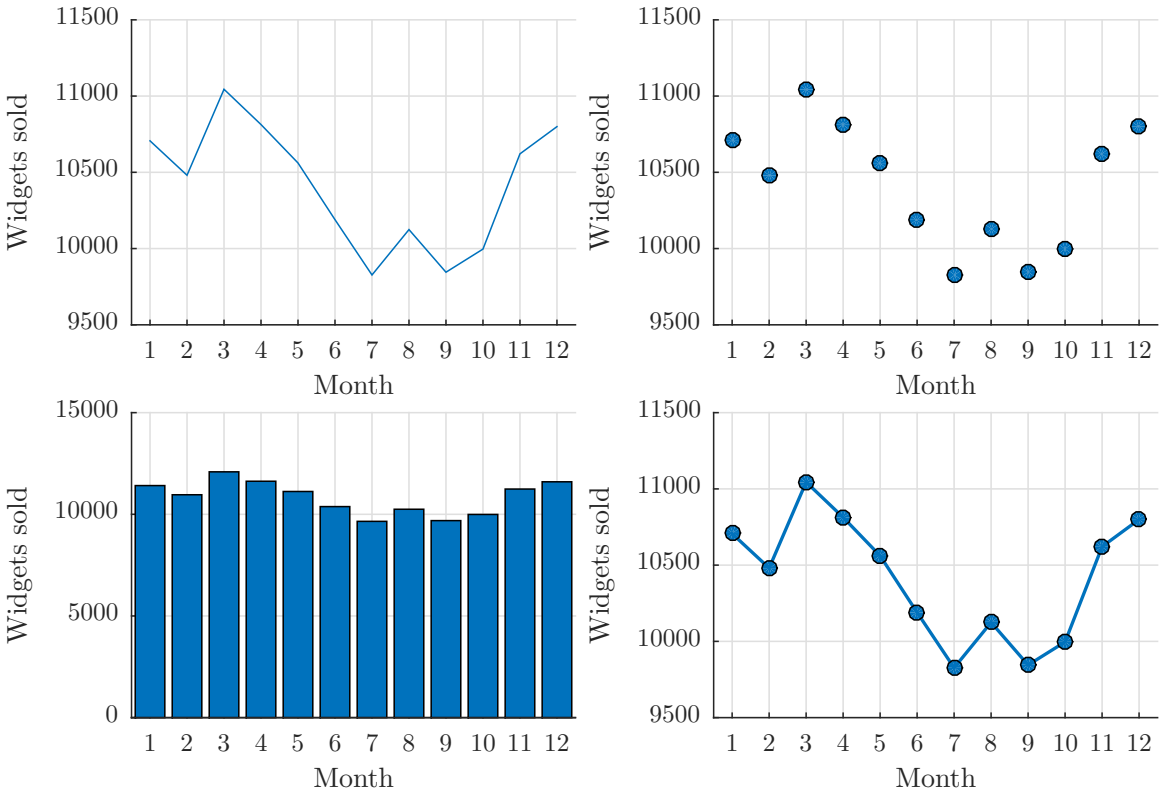


Fig. 6.3: Different attempts to visualize a simple 1D dataset corresponding to widget sale over months. Top row is the line plot and the dot-plot. Bottom left is the bar plot (notice the y-axis start at 0) and an attempt to combine the line and dot plot to better guide the readers eye.

this? Of course, we should include a table in the report. However, suppose we wanted 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. 6.4 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 a 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 benefit 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 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.

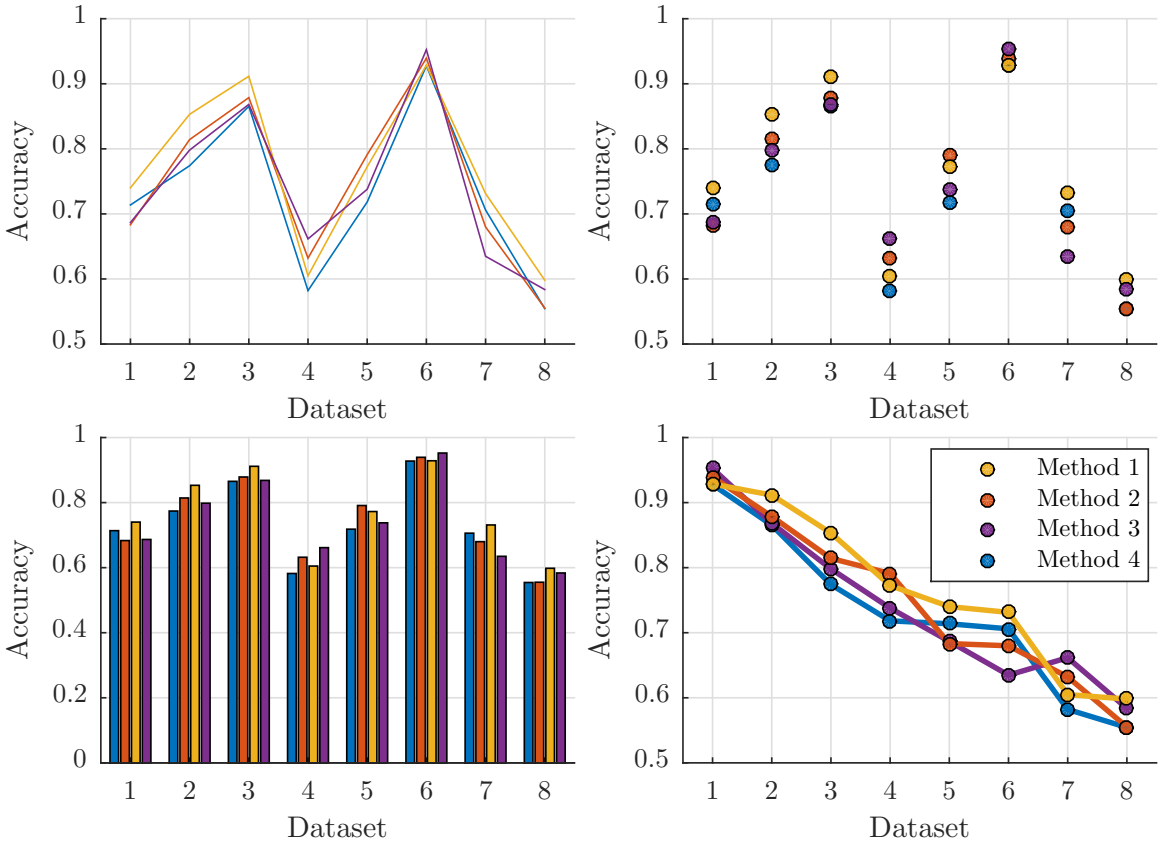


Fig. 6.4: Illustration of four 1D datasets corresponding to the performance of four machine-learning methods on eight datasets. In the bottom right pane we have tried to sort the datasets and use lines to connect related datasets. Which are easier to read?

Visualizing 2D data

Likely, the best and certainly the simplest way to visualize 2D data is the scatter plot. In fig. 6.5 we have plotted two coordinates of the 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 in what is known as a matrix plot, see fig. 6.6. 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

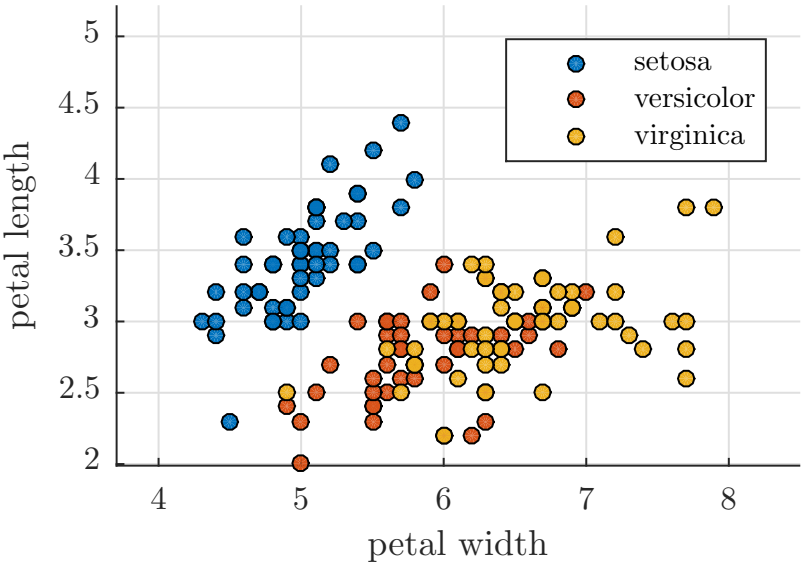


Fig. 6.5: Scatter plot of two attributes of the Fisher Iris data. Colors are used to visualize the three classes and from this plot we might expect it would be easy to tell setosa from the two other classes of flowers.

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 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.

Finally, one can perform a 3D scatter plot of the data where we consider three features together as shown in fig. 6.7. 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. As a rule, one should be critical of 3D plots for technical work and only include it if it has a tangible benefit. In the current 3D plot we have tried to preserve some of the 3D effect by drawing "shadows" of the points projected onto the 3 axis-oriented planes.

6.2 What is a good plot?

Having introduced the basic plots a natural question to ask is when to use which plot type and for what. Unfortunately, there is no single optimal question. However, there are useful guidelines [1].

A useful analogy is to consider technical writing. Suppose we are writing a paragraph in, say, a book about machine learning or a report. Which relevant questions should we keep in mind for this paragraph? Arguably, the first, and most important, question is what the point of the paragraph is. What particular *question* are we going to answer with this paragraph? Without such a question, there is no need for the paragraph and we would be much better off just avoiding it.

[1] The guidelines illustrated here are adopted from `http://junkcharts.typepad.com/junk_charts/`

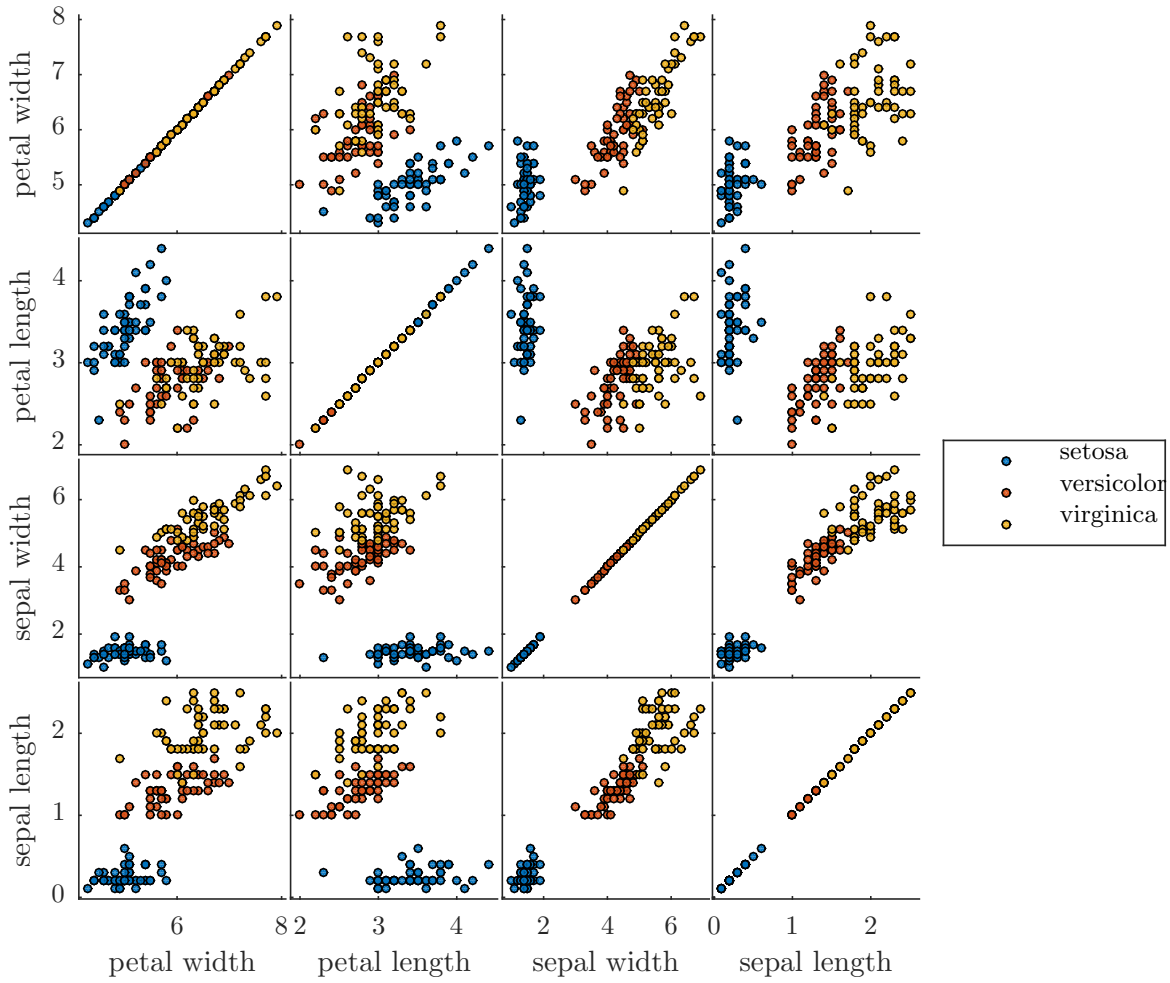


Fig. 6.6: A matrix plot in which the four attributes are plotted pairwise against each other.

When we have zoomed in on the point we wish to convey the next element is *how* the paragraph should address the question — should we use examples? An abstract definition first and then illustrations? Draw on other parts of the text? Consider to first explain why this is an important question?

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 writing [2] it is certainly important to ensure the text is enjoyable or, as a bare minimum, readable.

For plots (or visualizations) we should imagine a similar thought-process:

[2] Or so we hope...

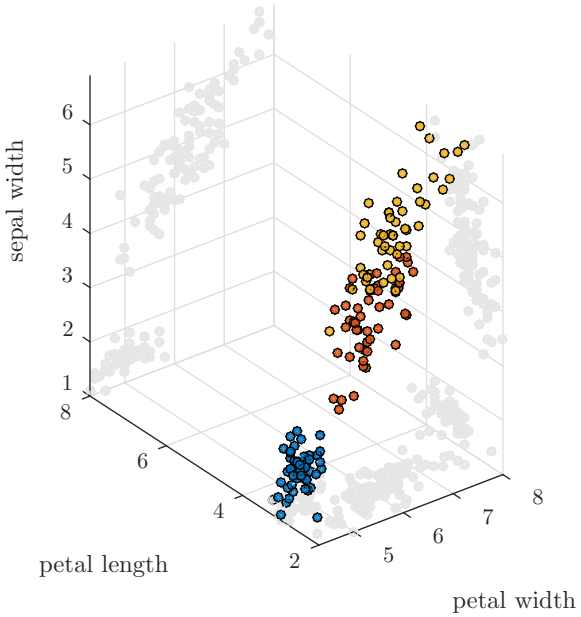


Fig. 6.7: A 3D scatter plot where three attributes are plotted against each other. 3D plots are best when one can interactively move the camera around since their 2D projection onto paper necessarily ruins much of the information that can be extracted. The gray shadows indicate the axis-aligned projections and can help to maintain some of the 3D effect.

The question: The first (and most important) aspect of plotting is the question the graphical element will address. A graphical element should convey one (or a very few) interesting facts to the reader and what those facts are should determine everything that subsequently goes on — it should be aligned with the main conclusions of the text and be interesting enough to warrant the space. A plot that is only supposed to "show what output we got" can and should be discarded.

The data: Next we should determine what data is useful to answer the question (as a rule, go with the bare minimum amount) and possibly what transformations should be applied to the data to (say) reduce noise, change scale (again as a rule, go with the bare minimum).

The visuals: Lastly (and least importantly), what visual element (the proceeding plot types) should be used to represent the data and answer the question. Preference should be given to simplicity. Consider (if possible) to make the important visual feature stand out, that correct labels are used, etc.

There are many answers as to how these steps, in particular the last, should be best carried out leading to "plot checklists". However more important than any set of rules is that *some thought is given to the process of making graphics*. 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 literally impossible to read; this type of errors (and the associated error of not writing labels to axis where it is non-obvious) is easily avoided with a bare minimum of thought. Simply becoming aware graphics are easy to get wrong is an important first step. 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.

Problems

6.1. Fall 2012 question 2: 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 6.8 a boxplot of the standardized data is given. Which of the following statements is *correct*?

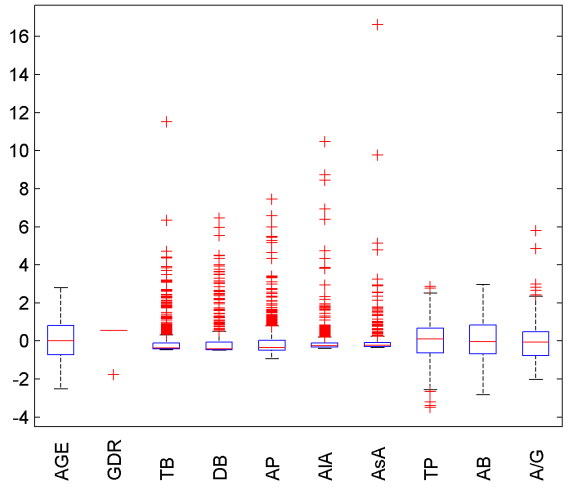


Fig. 6.8: Boxplots of the 10 attributes x_1–x_{10} where the data has been standardized.

A The value of the 50th and 75th percentiles of the attribute DB coincides.

B Even though the distribution of AlA 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.

6.2. Fall 2014 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. 6.9 is a boxplot of the dataset?

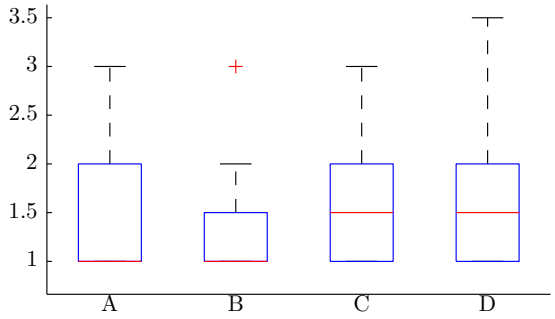


Fig. 6.9: Boxplots

A Boxplot A

B Boxplot B

C Boxplot C

D Boxplot D

E Don't know.

6.3. Spring 2014 question 1: 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 6.1.

In Figure 6.10 is given a boxplot of the six input attributes of the data. Which one of the following statements is *correct*?

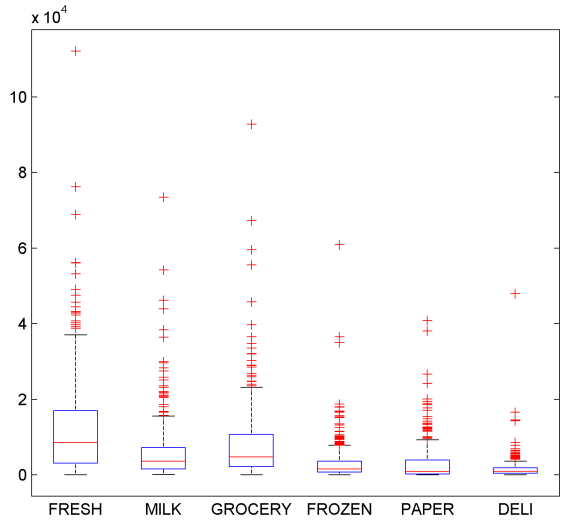


Fig. 6.10: Boxplot of the six input attributes x_1–x_6 of the wholesale data.

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 6.1: 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

7

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, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ and N targets $y_1, \ldots, y_N$ and we wish to come up with a way to predict y from $\boldsymbol{x}$:

$$y = f(\boldsymbol{x}, \boldsymbol{w}) + \varepsilon, \tag{7.1}$$

where $\boldsymbol{w}$ is a vector of tunable parameters and ε represents a noise term. Learning then consists of selecting the parameters $\boldsymbol{w}$ based on the training data $\boldsymbol{X}, \boldsymbol{y}$. If y is a continuous parameter, for instance the price of a stock, we will say that the model (detnoed $\mathcal{M}$) is a regression model. On the other hand if y is discrete, i.e. $y = 1, 2, \ldots, C$ 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 $f(\boldsymbol{x}, \boldsymbol{w})$ looks like and then show how probabilities can be used treat the noise term ε.

7.1 Linear models

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 it's origin with the discovery of the logistic function by Pierre François Verhulst and Adolphe Quételetin in 1838 [Garnier and Quételet, 1838] where it was originally applied to growth curves of populations. 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. (7.1) is modelled as a *linear combination* of the input features:

$$f(\boldsymbol{x}, \boldsymbol{w}) = w_0 + w_1 x_1 + \cdots + w_M x_M, \quad \text{and} \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{bmatrix}. \tag{7.2}$$

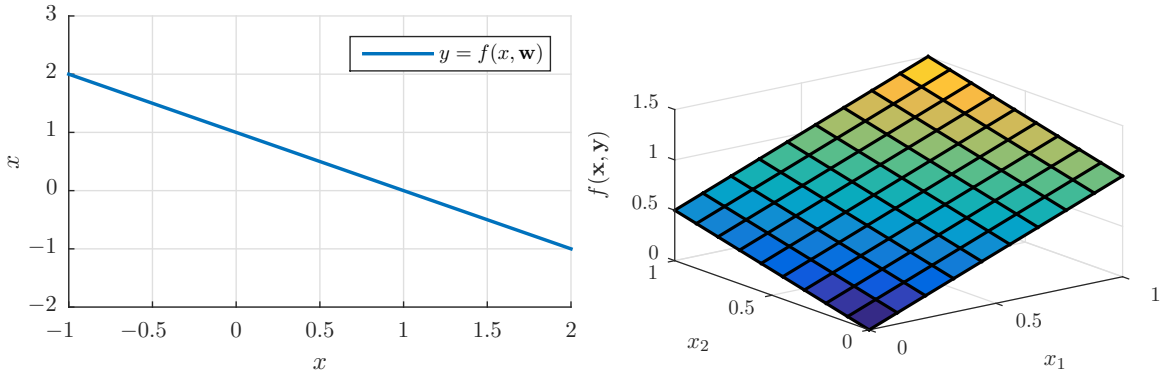


Fig. 7.1: The linear regression models prediction is a linear combination of the features $f(\boldsymbol{x}, \boldsymbol{w}) = w_0 + w_1 x_1 + \cdots + W_M x_M$. This allows for lines (left pane), $y = w_0 + w_1 x$, planes (right pane) $y = w_0 + w_1 x_1 + w_2 x_2$ and in general hyperplanes.

The reason this is known as a linear model is that it is a *linear function* of the input features $\boldsymbol{w}$. To consider a very simple example, consider the linear model with $w_0 = 1, w_1 = -1$ shown in fig. 7.1 as the blue line

$$y = f(x, \boldsymbol{w}) = 1 - x.$$

This naturally also extends to multiple input features. In the right-hand pane of fig. 7.1 is shown the two-dimensional regression example with $w_0 = 0, w_1 = 1, w_2 = \frac{1}{2}$.

$$y = f(\boldsymbol{x}, \boldsymbol{w}) = 0 + x_1 + \frac{1}{2} x_2 = x_1 + \frac{1}{2} x_2.$$

More generally, we can consider a feature transformation of $\boldsymbol{x}$ such that

$$y = f(\boldsymbol{x}, \boldsymbol{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\boldsymbol{x})$$

where $\phi_1(\boldsymbol{x}), \ldots, \phi_{M-1}(\boldsymbol{x})$ are $M - 1$ *basis functions*. If we define

$$\boldsymbol{\phi}(\boldsymbol{x}) = \begin{bmatrix} 1 \\ \phi_1(\boldsymbol{x}) \\ \vdots \\ \phi_{M-1}(\boldsymbol{x}) \end{bmatrix}$$

(that is, the first basis function is just a constant) we can write the linear regression model more compactly as simply

$$y = f(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{w} \quad \text{and} \quad \boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{M-1} \end{bmatrix}. \tag{7.3}$$

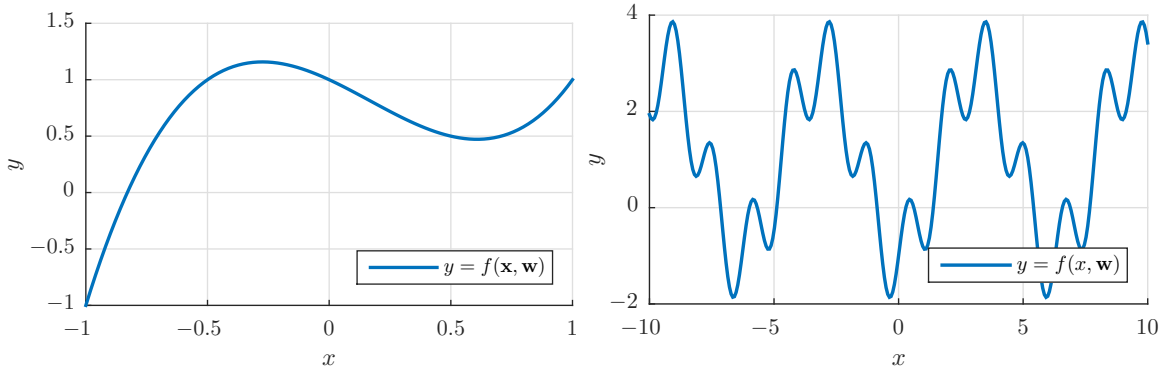


Fig. 7.2: Applying a non-linear transformation to the input vector x allows much more complicated curves to be fitted by the linear regression model. In the left-hand pane is shown a polynomial $y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$ and in the right-hand pane a sinusoidal model $y = w_0 + w_1 \cos(x) + w_2 \sin(4x)$.

The use of non-linear basis functions allow the linear regression model to model non-linear features. In fig. 7.2 is shown two such examples, the first is a 3rd degree polynomial corresponding to

$$\boldsymbol{\phi}(x) = \begin{bmatrix} 1 & x & x^2 & x^3 \end{bmatrix}^T \quad \text{and} \quad \boldsymbol{w} = \begin{bmatrix} 1 & -1 & -1 & 2 \end{bmatrix}^T. \tag{7.4}$$

The second example corresponds to two trigonometric functions suitable for a periodic signal

$$\boldsymbol{\phi}(x) = \begin{bmatrix} 1 & \cos(x) & \sin(4x) \end{bmatrix}^T \quad \text{and} \quad \boldsymbol{w} = \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}^T. \tag{7.5}$$

Since the transformation by a basis function does not change the linearity in $\boldsymbol{w}$ the discussion in this chapter will be independent on the choice of basis functions. In practical terms, applying a basis functions to a dataset $\boldsymbol{X}$ to obtain the transformed dataset $\tilde{\boldsymbol{X}}$ is equivalent to applying feature transformations such as the ones we encountered in chapter 2

$$\tilde{\boldsymbol{x}}_i = \phi(\boldsymbol{x}), \quad \text{and} \quad \tilde{\boldsymbol{X}} = \begin{bmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1)^T \\ \boldsymbol{\phi}(\boldsymbol{x}_2)^T \\ \vdots \\ \boldsymbol{\phi}(\boldsymbol{x}_N)^T \end{bmatrix}$$

and we can write the prediction of observation i as $y_i = \tilde{\boldsymbol{x}}_i^T \boldsymbol{w}$.

7.1.1 Training the linear regression model

The linear regression model eq. (7.3) is an ideal relationship between the input $\boldsymbol{x}$ and the target y. An actual observation will be noisy which we will capture with a noise parameter ε:

$$y = f(\boldsymbol{x}, \boldsymbol{w}) + \varepsilon$$

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. The probability density of y is then a normal distribution centered around $f(\boldsymbol{x}, \boldsymbol{w})$

$$p(y|\boldsymbol{x}, \boldsymbol{w}, \sigma) = \mathcal{N}(y|f(\boldsymbol{x}, \boldsymbol{w}), \sigma^2). \tag{7.6}$$

Given all observations $\boldsymbol{X}$ and targets $\boldsymbol{y}$ the likelihood of the entire dataset is (using the noise terms are independent):

$$p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w}) = \prod_{i=1}^{N} \mathcal{N}(y_i|f(\boldsymbol{x}_i, \boldsymbol{w}), \sigma^2).$$

How should we find $\boldsymbol{w}$? Let's simply start by applying Bayes' theorem

$$p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{y}) = \frac{p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w})p(\boldsymbol{w})}{p(\boldsymbol{y}|\boldsymbol{X})}. \tag{7.7}$$

We now make two simplifying assumptions: The first simplification is that we assume we are only interested in the *most probable* value of $\boldsymbol{w}$, that is, the value of $\boldsymbol{w}$ that maximizes the above expression written as $\boldsymbol{w}^* = \arg\max_{\boldsymbol{w}} p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{y})$. The second simplification is that we assume the prior of $\boldsymbol{w}$ is flat, $p(\boldsymbol{w}) = 1$, which is also known as the *uniform* or *improper* prior[1]. Later in section 12.1 we will consider another prior term later. Since maximizing eq. (12.5) with respect to $\boldsymbol{w}$ is equal to maximizing the logarithm and the term $p(\boldsymbol{y}|\boldsymbol{x})$ is constant and does not depend on $\boldsymbol{w}$ and therefore does not affect the maximum we are left with maximizing:

$$\begin{aligned}
\boldsymbol{w}^* = \arg\max_{\boldsymbol{w}} \left[p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{y})\right] = \arg\max_{\boldsymbol{w}} \left[\frac{p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w})p(\boldsymbol{w})}{p(\boldsymbol{y}|\boldsymbol{X})}\right] = \arg\max_{\boldsymbol{w}} \left[\log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w})\right] \\
\text{where } \log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w}) = \log \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_i - f(\boldsymbol{x}_i, \boldsymbol{w}))^2}{2\sigma^2}} \\
= -\frac{1}{2\sigma^2} \sum_{i=1}^{N} \|\boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{w}) - \boldsymbol{y}_i\|^2 - \frac{N}{2} \log(2\pi\sigma^2).
\end{aligned}$$

Maximizing the likelihood of the observed data is equivalent to maximizing the first part of this expression and is independent of σ^2. We therefore consider *minimizing* the sum-of-squares error function

$$\begin{aligned}
\arg\max_{\boldsymbol{w}} p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{y}) = \arg\max_{\boldsymbol{w}} \left[\log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w})\right] = \arg\min_{\boldsymbol{w}} E(\boldsymbol{w}) \\
\text{where } E(\boldsymbol{w}) = \frac{1}{2} \sum_{i=1}^{N} \left(f(\boldsymbol{x}_i, \boldsymbol{w}) - y_i\right)^2.
\end{aligned} \tag{7.8}$$

Maximizing the probability of the data is also known as the likelihood principle, however as we saw it is simply Bayes' theorem with some simplifications. When we train the linear regression model we thus try to find the value of the weights $\boldsymbol{w}^*$ as:

[1] Notice the choice $p(\boldsymbol{w}) = 1$ strictly speaking does not make sense since the density is no longer normalized: $\int p(\boldsymbol{w}) d\boldsymbol{w} = \infty$. The prior can however be understood as saying that no particular value of $\boldsymbol{w}$ is "preferred" over another or more formally it can be understood as using the prior $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}\delta)$ throughout the derivation and then taking the limit $\delta \to \infty$.

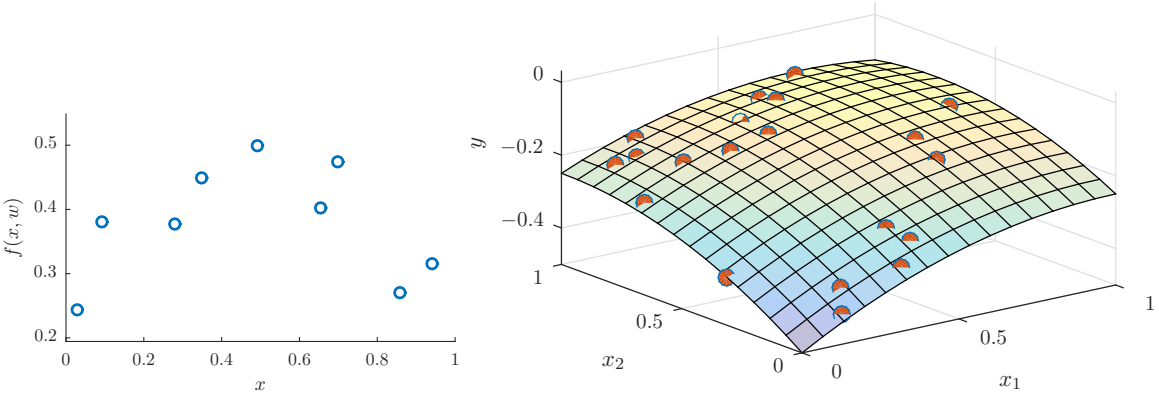


Fig. 7.3: Examples of two datasets for which we will apply linear regression. In the left-hand pane is a 1-d dataset comprised of x, in the right-hand side a 2d dataset comprised of red dots lying on a curved plane.

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} E(\boldsymbol{w}).$$

As we know from analysis, this can be accomplished by setting the derivative of E equal to zero and solving. If we differentiate eq. (7.8) with respect to a weight w_j we obtain:

$$\frac{\partial}{\partial w_j} E(\boldsymbol{w}) = \sum_{i=1}^{N} (y_i - \tilde{\boldsymbol{x}}_i^T \boldsymbol{w}) \tilde{X}_{ij} = \sum_{i=1}^{N} y_i \tilde{X}_{ij} - \left[\sum_{i=1}^{N} X_{ij} \tilde{\boldsymbol{x}}_i^T \right] \boldsymbol{w} \tag{7.9}$$

Thus the gradient is

$$\nabla E(\boldsymbol{w}) = \begin{bmatrix} \frac{\partial E(\boldsymbol{w})}{\partial w_1} \\ \vdots \\ \frac{\partial E(\boldsymbol{w})}{\partial w_M} \end{bmatrix} = \tilde{\boldsymbol{X}}^T \boldsymbol{y} - (\tilde{\boldsymbol{X}}^T \tilde{\boldsymbol{X}}) \boldsymbol{w}. \tag{7.10}$$

Setting the gradient equal to zero and solving we obtain

$$\boldsymbol{w}^* = (\tilde{\boldsymbol{X}}^T \tilde{\boldsymbol{X}})^{-1} \tilde{\boldsymbol{X}}^T \boldsymbol{y} = (\tilde{\boldsymbol{X}}^T \tilde{\boldsymbol{X}}) \backslash \tilde{\boldsymbol{X}}^T \boldsymbol{y}. \tag{7.11}$$

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 to 1d dataset

Consider the 1d dataset shown in the left-pane of fig. 7.3. 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.

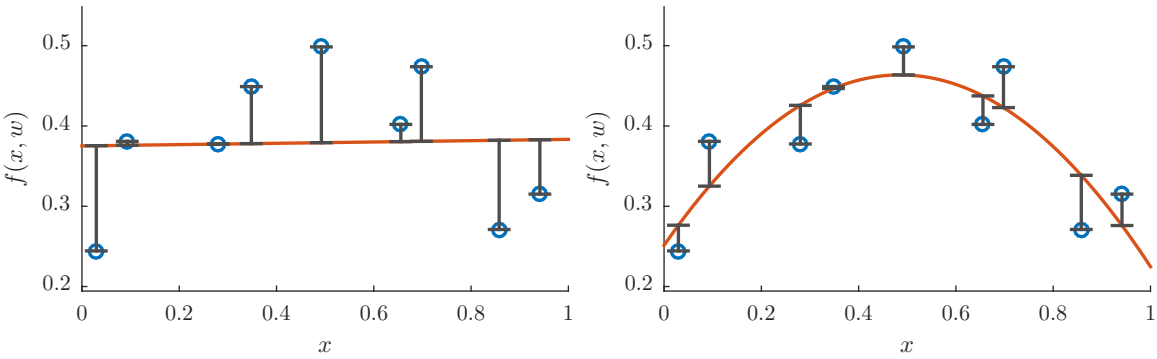


Fig. 7.4: Examples of applying the linear regression model to the dataset shown in the left-hand pane of fig. 7.3. The two panes show a basic linear regression model $\boldsymbol{y} = \tilde{\boldsymbol{X}}_{(1)}\boldsymbol{w}(1)$ without feature transformations, and the right-hand pane we make a feature transformation by adding the feature x^2 to produce a second-polynomial curve, $\boldsymbol{y} = \tilde{\boldsymbol{X}}_{(2)}\boldsymbol{w}_{(2)}$. See text for details.

For the first order polynomial linear regression case, this is accomplished by applying the (identity) feature transformation:

$$\tilde{\boldsymbol{X}}_{(1)} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix} \tag{7.12}$$

then we compute $\boldsymbol{w}_{(1)} = (\tilde{\boldsymbol{X}}_{(1)}^T\tilde{\boldsymbol{X}}_{(1)})\backslash\tilde{\boldsymbol{X}}_{(1)}^T\boldsymbol{y}$ and and the red curve for an arbitrary point x can be predicted as $y = f(x, \boldsymbol{w}_{(1)}) = \begin{bmatrix} 1 & x \end{bmatrix}\boldsymbol{w}_{(1)}$. In the right-hand pane of fig. 7.4 we illustrate the second-degree polynomial linear regression corresponding to the feature transformation:

$$\tilde{\boldsymbol{X}}_{(2)} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \\ 1 & x_N & x_N^2 \end{bmatrix} \tag{7.13}$$

then we compute $\boldsymbol{w}_{(2)} = (\tilde{\boldsymbol{X}}_{(2)}^T\tilde{\boldsymbol{X}}_{(2)})\backslash\tilde{\boldsymbol{X}}_{(2)}^T\boldsymbol{y}$ and the red curve for an arbitrary point x can be predicted as $y = f(x, \boldsymbol{w}_{(2)}) = \begin{bmatrix} 1 & x & x^2 \end{bmatrix}\boldsymbol{w}_{(2)}$.

Example 2: Linear regression to 2d dataset

In the second example, we will consider the 2d dataset shown in the right-hand pane of fig. 7.3. 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. 7.5 we illustrate the second-degree polynomial linear regression corresponding to the feature transformation:

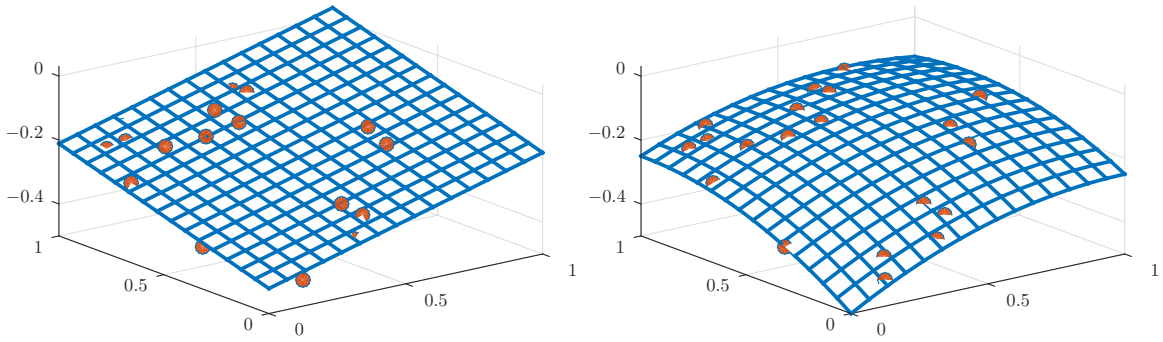


Fig. 7.5: Examples of applying the linear regression model to the dataset shown in the righ-hand pane of fig. 7.3. The left-hand pane shows the basic linear regression model $\boldsymbol{y} = \tilde{\boldsymbol{X}}_{(1)}\boldsymbol{w}$, and in the right-hand pane we make a feature transformation to include second order terms corresponding to $\boldsymbol{y} = \tilde{\boldsymbol{X}}_{(2)}\boldsymbol{w}$. See text for details.

$$\tilde{\boldsymbol{X}}_{(1)} = \begin{bmatrix} 1 & X_{11} & X_{12} \\ 1 & X_{22} & X_{22} \\ \vdots & \vdots & \vdots \\ 1 & X_{N1} & X_{N2} \end{bmatrix} \tag{7.14}$$

then we compute $\boldsymbol{w}_{(1)} = (\tilde{\boldsymbol{X}}_{(1)}^T\tilde{\boldsymbol{X}}_{(1)})\backslash\tilde{\boldsymbol{X}}_{(1)}^T\boldsymbol{y}$ (notice $\boldsymbol{w}$ is three-dimensional) and for an arbitrary point $\boldsymbol{x}$ we can predict $y = f(\boldsymbol{x}, \boldsymbol{w}_{(1)}) = \begin{bmatrix} 1 & x_1 & x_2 \end{bmatrix}\boldsymbol{w}_{(1)}$. This is used to generate the plane shown in the left-hand pane of fig. 7.5.

In the second example, we will attempt to fit a second-order expansion to the same dataset. This is accomplished by the feature transformation:

$$\tilde{\boldsymbol{X}}_{(2)} = \begin{bmatrix} 1 & X_{11} & X_{12} & X_{11}^2 & X_{12}^2 & X_{11}X_{12} \\ 1 & X_{21} & X_{22} & X_{21}^2 & X_{22}^2 & X_{21}X_{22} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & X_{N1} & X_{N2} & X_{N1}^2 & X_{N2}^2 & X_{N1}X_{N2} \end{bmatrix} \tag{7.15}$$

Again, learning $\boldsymbol{w}$ (which is now 7-dimensional!) can be accomplished as $\boldsymbol{w}_{(2)} = (\tilde{\boldsymbol{X}}_{(2)}^T\tilde{\boldsymbol{X}}_{(2)})\backslash\tilde{\boldsymbol{X}}_{(2)}^T\boldsymbol{y}$ and predictions, as shown in the right-hand pane of fig. 7.5, for a new point $\boldsymbol{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T$ can be made as

$$y = f(\boldsymbol{x}, \boldsymbol{w}_{(2)}) = \begin{bmatrix} 1 & x_1 & x_2 & x_1^2 & x_2^2 & x_1x_2 \end{bmatrix}\boldsymbol{w}_{(2)}.$$

In the later case the found value of $\boldsymbol{w}_{(2)}$ is

$$\boldsymbol{w}_{(2)} = \begin{bmatrix} -0.5 & 0.5 & 0.5 & -0.25 & -0.25 & -0.125 \end{bmatrix}^T,$$

which is exactly (at this precision at least) equal to the value of $\boldsymbol{w}$ used to generate the data.

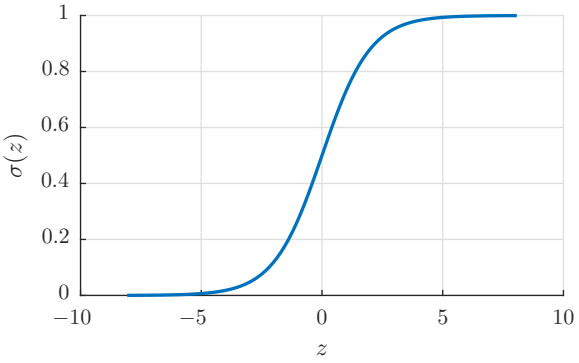


Fig. 7.6: The logistic sigmoid $\sigma(z) = (1 + e^{-z})^{-1}$. Notice as $z \to -\infty$ then $\sigma(z) \to 0$ and when $z \to \infty$ then $\sigma(z) \to 1$.

7.2 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. Suppose we ask for the probability an observation $\boldsymbol{x}$ belongs to class $y = 1$; since we are asking for a probability our immediate hunch should be to apply Bayes' theorem:

$$p(y = 1|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|y = 1)p(y = 1)}{p(\boldsymbol{x}|y = 0)p(y = 0) + p(\boldsymbol{x}|y = 1)p(y = 1)} \tag{7.16}$$

$$= \frac{1}{1 + \frac{p(\boldsymbol{x}|y=0)p(y=0)}{p(\boldsymbol{x}|y=1)p(y=1)}} = \frac{1}{1 + e^{-z}} = \sigma(z). \tag{7.17}$$

Where we have defined the two quantities

$$z = \log \frac{p(\boldsymbol{x}|y = 1)p(y = 1)}{p(\boldsymbol{x}|y = 0)p(y = 0)} \quad \text{and} \quad \sigma(z) = \frac{1}{1 + e^{-z}}$$

the function $\sigma(z)$ is known as the *logistic sigmoid* and is shown in fig. 7.6. Notice in the above z may take both positive and negative values because of the logarithm and if z is large for a given $\boldsymbol{x}$ this implies a greater chance that $\boldsymbol{x}$ is classified in the positive class. The idea in logistic regression is then simply to let z be the output obtained by linear regression. That is, we consider a model where

$$z = \boldsymbol{X}\boldsymbol{w}, \quad p(y = 1|\boldsymbol{x}) = \sigma\left(\boldsymbol{X}\boldsymbol{w}\right) \tag{7.18}$$

and $\boldsymbol{X}$ may be transformed by adding non-linear features as in the previous section for linear regression. However, in this section we will simply assume it has been transformed by adding an extra column of ones to give a basic linear prediction model. Now that we know the probability that y belongs to the positive class, and y is binary, we can also conclude that y just follows a Bernoulli distribution with parameter $\hat{y} = \sigma(\boldsymbol{X}\boldsymbol{w})$. The probability that a datapoint $\boldsymbol{x}_i$ is in class y_i is therefore

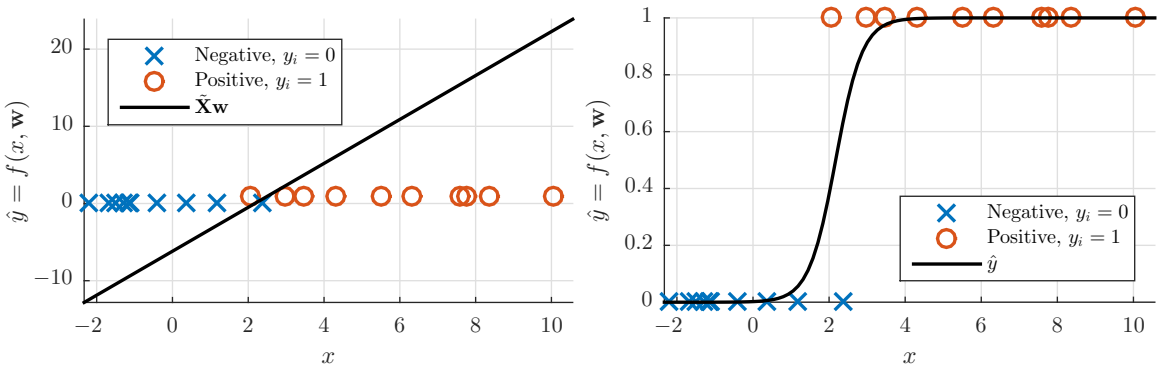


Fig. 7.7: A one-dimensional logistic regression example. The left-hand pane shows the intermediate (linear) output $\boldsymbol{X}\boldsymbol{w}$ and the right-hand pane the true logistic-regression decision boundary $\hat{y} = \sigma(\boldsymbol{X}\boldsymbol{w})$.

$$p(y_i|\boldsymbol{x}_i, \boldsymbol{w}) = p_{\text{Bernoulli}}(y_i|\hat{y}_i) = \hat{y}_i^{y_i}(1-\hat{y}_i)^{1-y_i}$$

The probability of the full dataset can then be obtained by multiplying the probability of the N (independent) observations together:

$$p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w}) = \prod_{i=1}^{N} p(y_i|\boldsymbol{x}_i, \boldsymbol{w}) = \prod_{i=1}^{N} \hat{y}_i^{y_i}(1-\hat{y}_i)^{1-y_i}. \tag{7.19}$$

We can now repeat the argument we saw for linear regression from eq. (12.5) onwards and find that the most probable $\boldsymbol{w}$ is the one which makes the logarithm of the above expression as large as possible. We therefore define the error function to be *minimized*:

$$E(\boldsymbol{w}) = -\log\left[p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w})\right] = -\sum_{i=1}^{N}\left[y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i))\right], \quad \hat{y}_i = \sigma\left[f(\boldsymbol{x}_i, \boldsymbol{w})\right], \tag{7.20}$$

and exactly as was the case for linear regression the optimal weights $\boldsymbol{w}^*$ are obtained by maximizing the probability of the data or equivalently minimizing the above error:

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} E(\boldsymbol{w}).$$

However, at this point our analysis departs from linear regression: there is no closed-form analytical solution for the minimum of the error function. How we find $\boldsymbol{w}^*$ 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 $\boldsymbol{X}$ also "works" for logistic regression and we will therefore only consider two simple examples where $\boldsymbol{X}$ has not had feature-transformations applied to it asides being pre-fixed with 1s as is required for any regression problem. In fig. 7.7 is shown a basic logistic regression example for a simple 1-dimensional dataset. In the left-and pane is shown the intermediate linear regression $z = \boldsymbol{X}\boldsymbol{w}$ and in the right-hand pane the predicted class-membership probability $\hat{y} = \sigma(\boldsymbol{X}\boldsymbol{w})$. As expected for such a simple problem the logistic regression

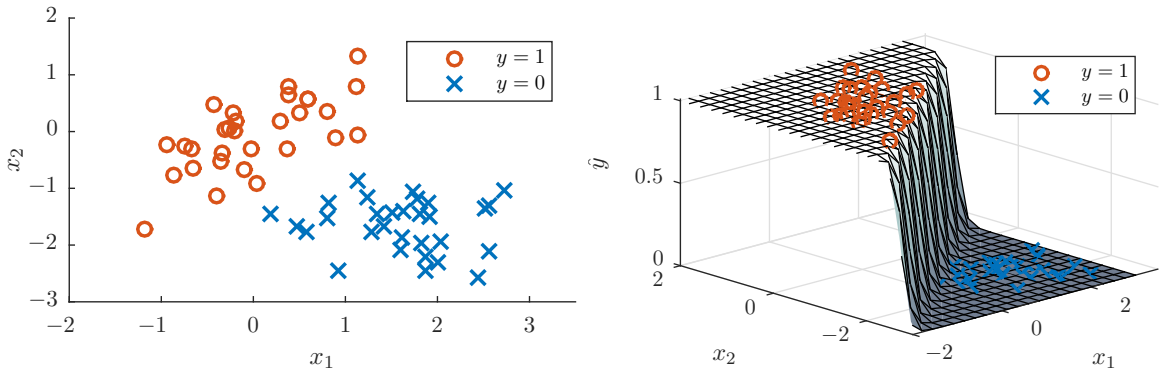


Fig. 7.8: 2d logistic regression example. The dataset shown in the left-hand pane is fitted with a logistic regression model and the class-membership prediction $\hat{y}$ is shown in the right-hand pane.

learns how to separate the two classes. In fig. 7.8 is shown a 2d example, where the two classes are fitted with a logistic regression model and the decision surface $\hat{y}$ is shown.

7.2.1 The confusion matrix

While the error function $E(\boldsymbol{w})$ could be used to evaluate a logistic regression model it is important to keep in mind the error function measure the *probability* of the data and, at the end of the day, we are probably more interested in how often the logistic regression model classify 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 9) and predict that observation i belongs to the positive class if $\hat{y}_i > \frac{1}{2}$ and the negative class if $\hat{y}_i \leq \frac{1}{2}$.

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:

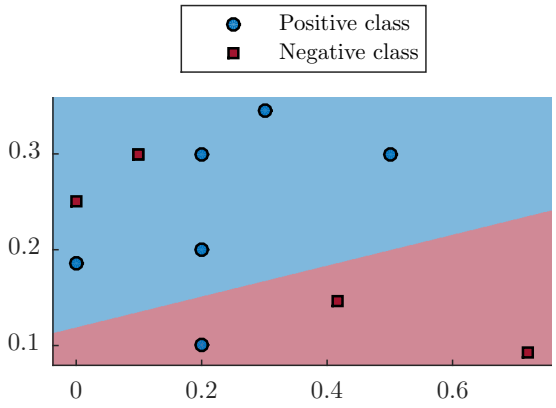
True Positives, TP: Number of observations which are in fact positive $y_i = 1$ which the classifier correctly labels as positive $\hat{y}_i > \frac{1}{2}$

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. 7.9. 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 9, 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*):



$N = 10$	Predicted **Positive**	Predicted **Negative**
Actually **Positive** $N^+ = 6$	TP $= 5$ (True Positive)	FN $= 1$ (False Negative)
Actually **Negative** $N^- = 4$	FP $= 2$ (False Positive)	TN $= 2$ (True Negative)

Fig. 7.9: (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.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{N}, \quad \text{Error rate} = \frac{\text{FP} + \text{FN}}{N} = 1 - \text{Accuracy}.$$

For instance the accuracy of the logistic regression model in fig. 7.9 is $\frac{5+2}{N} = \frac{7}{10}$.

Problems

7.1. Spring 2013 question 12: We fit a linear regression model to the PM10 data shown in Table 7.1. 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:

$$f(\boldsymbol{x}) = 3.27 + 0.36x_1 - 0.01x_2 - 0.19x_3 + 0.01x_4 + 0.05x_7.$$

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 7.1: 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.

7.2. Spring 2014 question 5: We consider the Wholesale dataset shown in Table 7.2 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 7.2: 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.

7.3. Fall 2013 question 6: We consider the Galapapos dataset shown in Table 7.3 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 7.10:

$$f(x_1, x_2, x_4, x_5, x_6, x_7) = 63.4 + 4.3x_1 - 34.7x_2 + 3.0x_4 - 7.2x_5 - 1.4x_6 - 0.5x_7$$

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 7.3: The seven attributes of the data on a selection of 29 of the Galápagos islands.

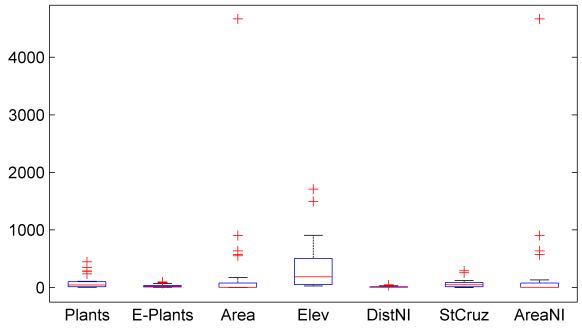


Fig. 7.10: Boxplot of the seven input attributes of the Galápagos data.

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.

7.4. Fall 2013 question 17: 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 7.4. Considering again the confusion matrices for the logistic regression and decision tree classifiers given in Figure 7.11, which one of the following statements is *correct*?

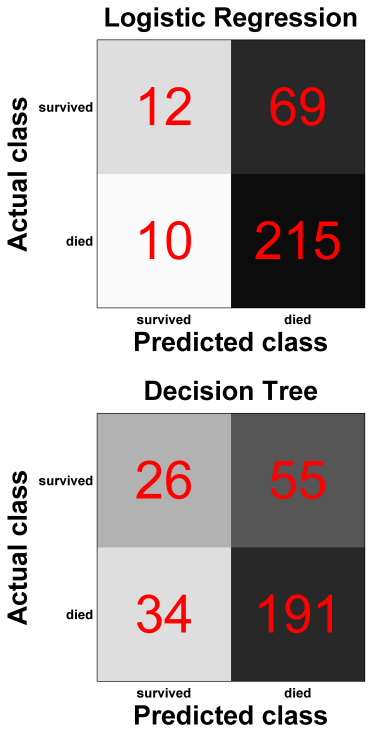


Fig. 7.11: The confusion matrix for the logistic regression and decision tree classifiers used to predict survival based on leave-one-out cross validation.

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 No, $x_3 = 0$ or Yes, $x_3 = 1$	PAN
y	Lived after 5 years No, $y = 0$ or Yes, $y = 1$	Surv

Table 7.4: 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.

7.5. Fall 2014 question 22:

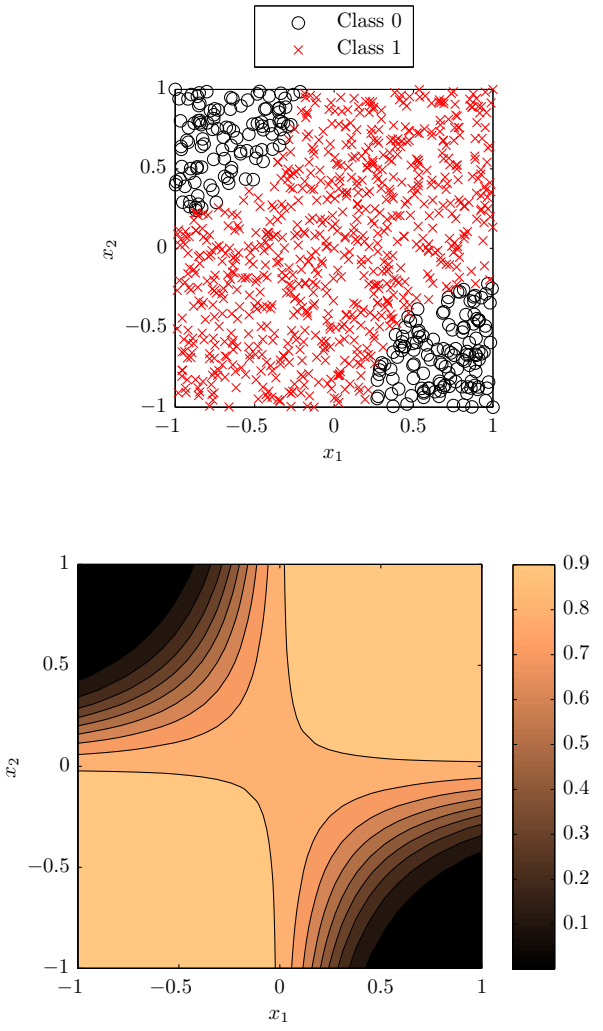


Fig. 7.12: top pane: Observed data. Bottom pane: Estimated probability of belonging to class 1 according to the logistic regression classifier.

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. 7.12 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

$$f(x_1, x_2) = \text{logistic}(w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2)$$

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.

8

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 $\boldsymbol{x}_i$, 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 $\boldsymbol{x}_i$ 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 8.1: Animals dataset. A dataset of $N = 15$ observations and $M = 3$ binary features where the goal is to predict if the animal is a Mammal or not

Name	x_1: Warm Blooded	x_2: Has Legs	x_3: Lay eggs	x_4:Has Fur	y: Mammal
Snake	-	-	yes	-	-
Starfish	-	-	yes	-	-
Bluebird	yes	yes	yes	-	-
Blackbird	yes	yes	yes	-	-
Earthworm	-	-	yes	-	-
Chameleon	-	-	yes	-	-
Ant	-	-	yes	-	-
Jellyfish	-	-	yes	-	-
Snail	-	-	yes	-	-
Sea Urchin	-	-	yes	-	-
Dolphin	yes	-	-	-	yes
Rat	yes	yes	-	yes	yes
Dog	yes	yes	-	yes	yes
Monkey	yes	yes	-	yes	yes
Lion	yes	yes	-	yes	yes

Algorithm 1: 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 $D_r = \{D_{v_1}, \ldots, D_{v_K}\}$ with the highest purity gain
Recursively call the method on $D_{v_1}, \ldots, D_{v_K}$
end if

8.1 Classification trees

Consider the dataset in table 8.1 comprised of $N = 15$ animals. For each animal, we have recorded $M = 4$ features (Warm blooded, Has legs and Lay eggs) 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 $\boldsymbol{X}$ and a vector $\boldsymbol{y}$, however for the moment we will limit ourselves to the case where $\boldsymbol{X}$ 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. 8.1. 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. 8.2 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 1.

8.1.1 Impurity measures and purity gains

So how do we determine when one question is better than another? In fig. 8.3 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. 8.3, and we denote the root of the tree by r and the two branches by v_1 and v_2, the left-most branch

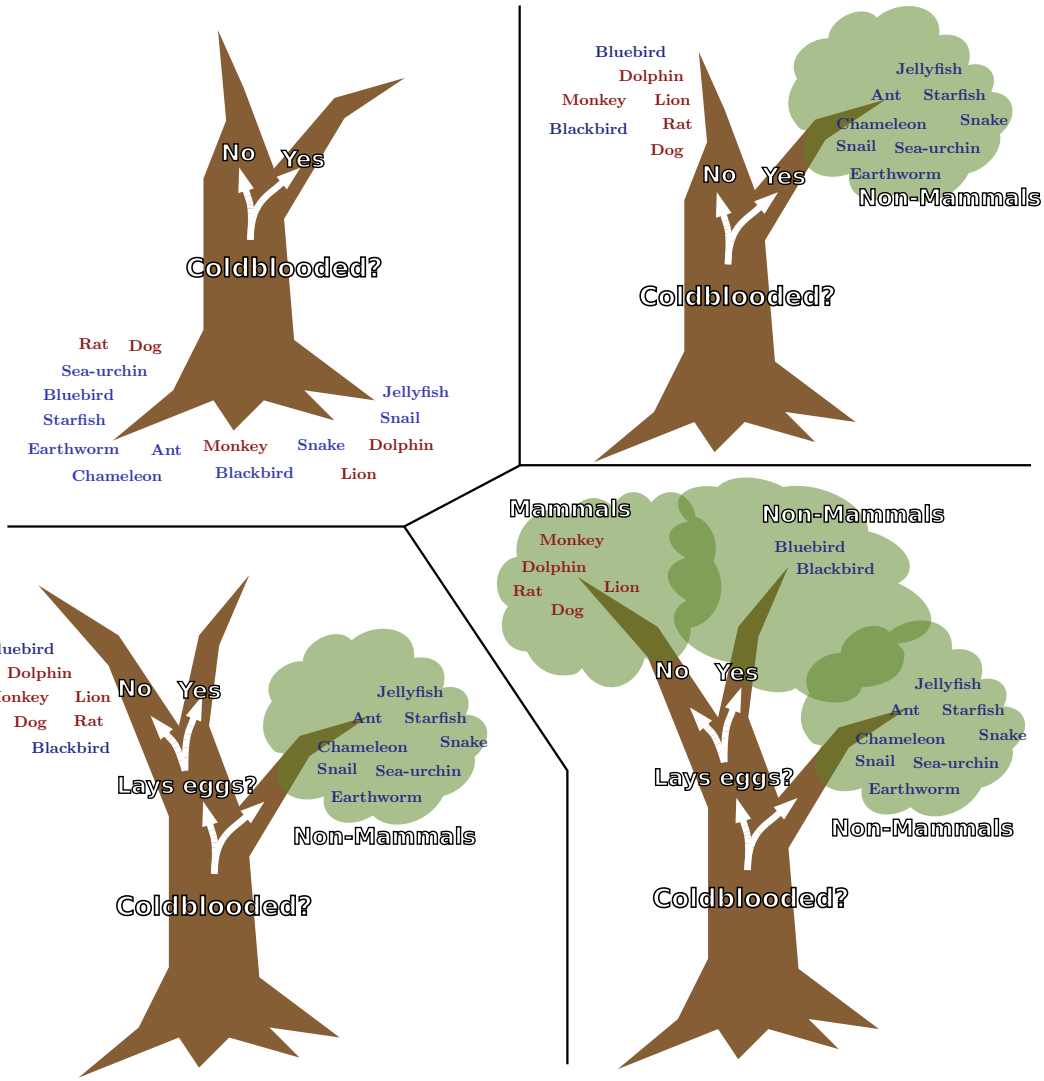


Fig. 8.1: Construction of a decision tree using Hunt's algorithm to classify whether an animal is a mammal or not. Initially, all observations are assigned to the root (top left pane) and we consider a question. The question divides the animals into two sets, if a particular set is pure, i.e. only contains 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 question at each branch, and select the best according to it's *purity gain* .

of the tree, v_1, contains $N(v_1) = 7$ animals whereas the right-most branch contains $N(v_2) = 8$

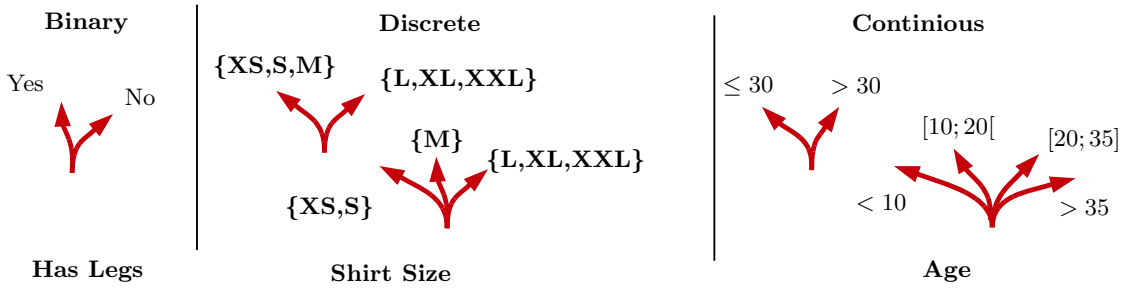


Fig. 8.2: Different types of attributes allow different splits. Binary attributes only allow binary (yes/no) splits, whereas discrete and continuous attributes allow either binary splits or many-way splits.

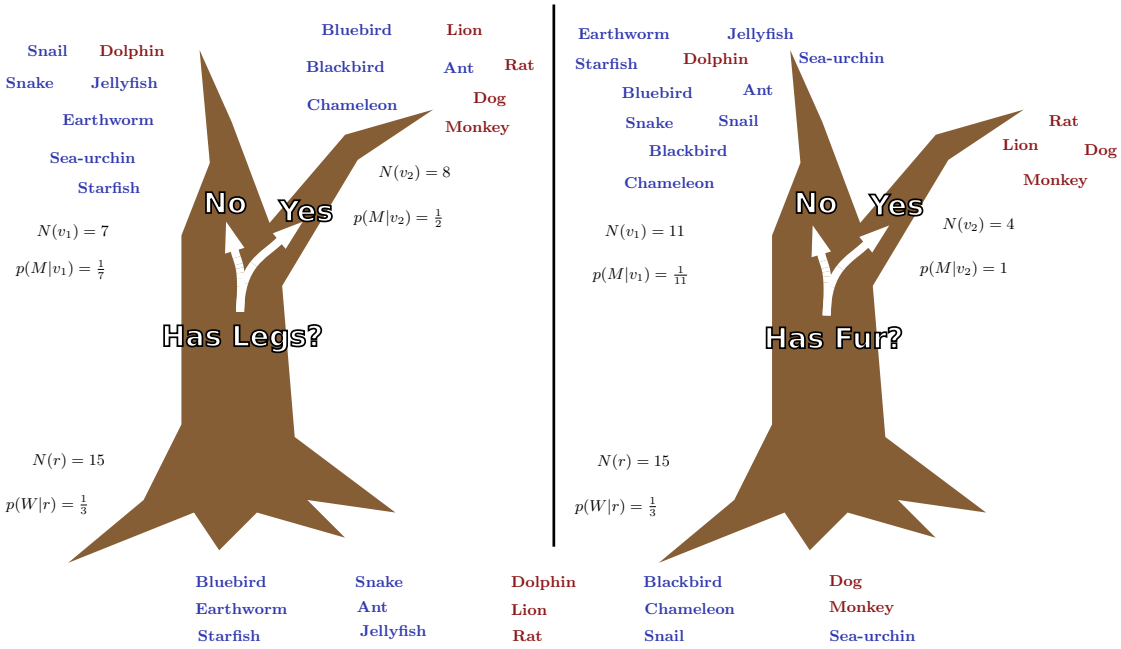


Fig. 8.3: Two different potential splits (left and right pane). The splits divide the animals at the root r into different groups corresponding to the left v_1 and right v_2 branch. When choosing between two splits, we are interested in how *balanced* they are (i.e. if $N(v_1) \approx N(v_2)$) and to 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.

animals. The question is thus fairly *balanced*. On the other hand for the right-most branch we have $N(v_1) = 11$ and $N(v_2) = 4$.

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. 8.3 the left-

branch v_1 contains only one mammal (i.e. the probability the animal is a mammal in this branch is $p(M|v_1) = \frac{1}{8}$) and in the right-branch $p(M|v_2) = \frac{4}{7}$. However, the fur-split in the right hand pane is much more balanced 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 $I(v_1), I(v_2), \ldots, I(v_K)$ 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

$$\Delta = I(r) - \sum_{k=1}^{K} \frac{N(v_k)}{N(r)} I(v_k). \tag{8.1}$$

A high purity gain indicates that the impurity of the individual splits, $I(v_1), \ldots, I(v_K)$ is low, i.e. the classes has 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 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 $p(c = 1|v), \ldots, p(c = C|v)$ (in the animals example we had two classes corresponding to $C = 2$ and $p(M|v), p(\text{not } M|v)$) and there are three popular choices for impurity function $I(v)$, entropy, Gini and classification error:

$$\text{Entropy}(v) = -\sum_{c=1}^{C} p(c|v) \log p(c|v) \tag{8.2}$$

$$\text{Gini}(v) = 1 - \sum_{c=1}^{C} p(c|v)^2 \tag{8.3}$$

$$\text{ClassError}(v) = 1 - \max_c p(c|v) \tag{8.4}$$

Suppose we consider the example in the right pane of fig. 8.3 and we use the ClassError impurity measure we obtain:

$$I(r) = 1 - \frac{2}{3} = \frac{1}{3}, \quad I(v_1) = 1 - \frac{10}{11} = \frac{1}{11} \quad \text{and} \quad I(v_2) = 1 - 1 = 0 \tag{8.5}$$

We can then compute the purity gain as

$$\Delta = I(r) - \frac{11}{15} I(v_1) - \frac{4}{15} I(v_2) = \frac{1}{3} - \frac{1}{15} = \frac{4}{15}. \tag{8.6}$$

As mentioned, the purity gain can easily be computed for many types of splits and having multiple classes. In fig. 8.4 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

$$\Delta = I(r) - \frac{5}{12} I(v_1) - \frac{1}{4} I(v_2) - \frac{1}{3} I(v_3). \tag{8.7}$$

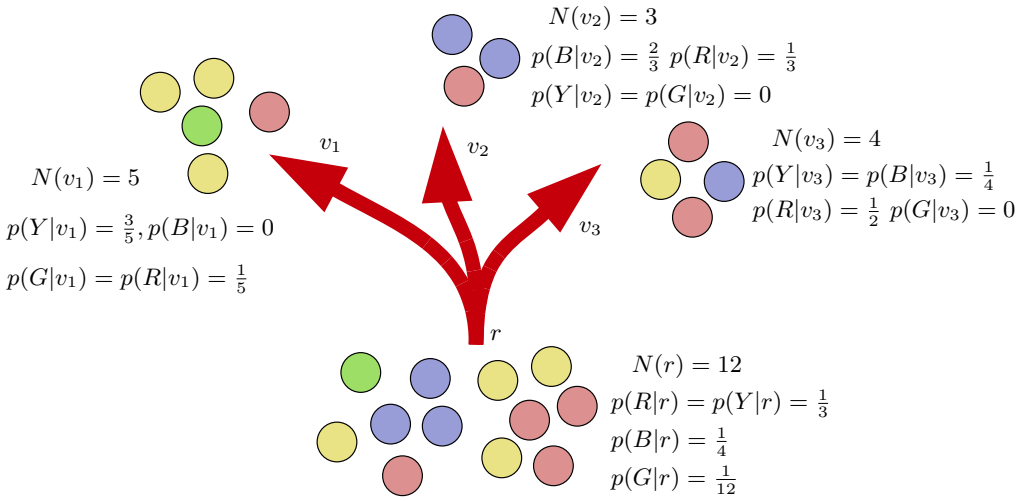


Fig. 8.4: A multi-way split where $N = 12$ observations belonging to $C = 4$ classes are split in a $K = 3$ way split. The classes are indicated by the colors. See text for details on how the purity gain Δ can be computed from the given numbers.

In practice, we are of course primarily interested in applying classification trees to the case where $\boldsymbol{X}$ contains general continuous features. The splits most often considered are binary, two-way splits obtained by considering each of the M features of $\boldsymbol{X}$ and then attempting different possible split-values:

$$x_m < x^* \tag{8.8}$$

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. 8.5. 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.

8.1.2 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. 8.5 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:

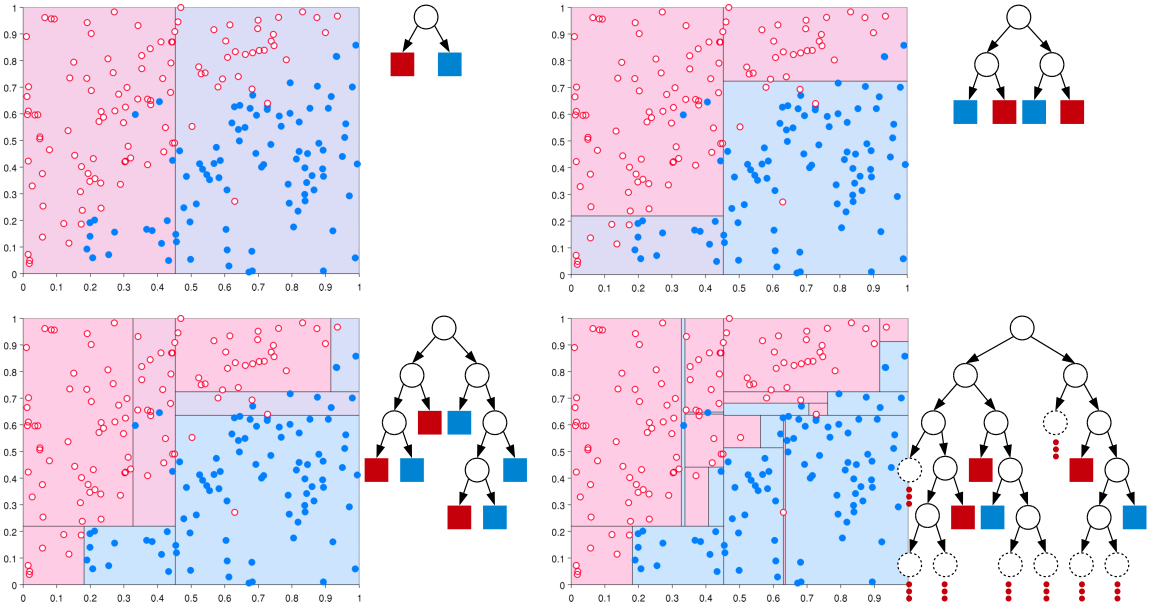


Fig. 8.5: Construction of a decision tree using Hunt's algorithm. We consider binary $K = 2$ splits where we as candidate splits consider if x_1 (and x_2) is less than or greater than a sequence of split-values. In the top-left pane we have selected an initial split based on the value of the x-axis. 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 leave 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, however in chapter 9 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 which are replaced by a single leaf (i.e. are 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 $T_0, T_1, \cdots, T_m$ 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 2: Cost complexity pruning of decision trees

Require: Initial full tree T_0 produced by Hunt's algorithm
Require: $T_0, T_1, \cdots, T_m$, a sequence of increasingly more pruned trees
  for $i = 1, 2, \cdots$ 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

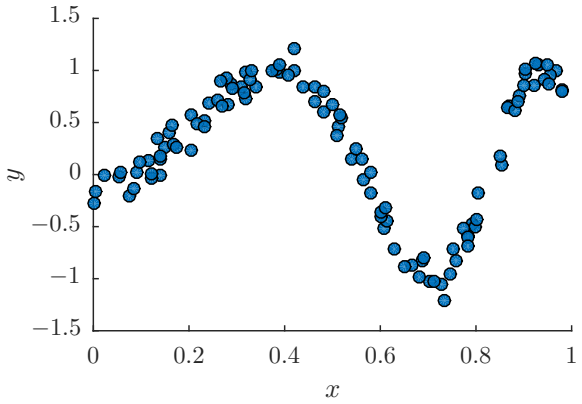


Fig. 8.6: A simple 1D example regression data set containing $N = 100$ observations.

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 2. Once the sequence $T_0, \cdots, T_m$ 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 9 as *two-layer cross-validation*.

8.2 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. 8.3, we would simply predict the mean value of the animals in that branch:

Algorithm 3: 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 $D_r = \{D_{v_1}, \ldots, D_{v_K}\}$ with the highest purity gain
Recursively call the method on $D_{v_1}, \ldots, D_{v_K}$
end if

$$\text{Predicted } y\text{-value in } v_2 \ : y(v_2) = \frac{y_{\text{Rat}} + y_{\text{Lion}} + y_{\text{Dog}} + y_{\text{Monkey}}}{4} \tag{8.9}$$

$$= \frac{\sum_{i \in v_2} y_i}{N(v_2)} \tag{8.10}$$

To evaluate the goodness of a new split, we can now simply compute the 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:

$$I(v) = \sum_{i \in v} (y_i - y(v))^2 \quad \text{where} \quad y(v) = \frac{1}{N(v)} \sum_{i \in v} y_i, \tag{8.11}$$

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 1 but for completeness it is listed in algorithm 3.

We will consider this method applied to the simple 1-d regression problem in fig. 8.6, the result can be seen in fig. 8.7. 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.

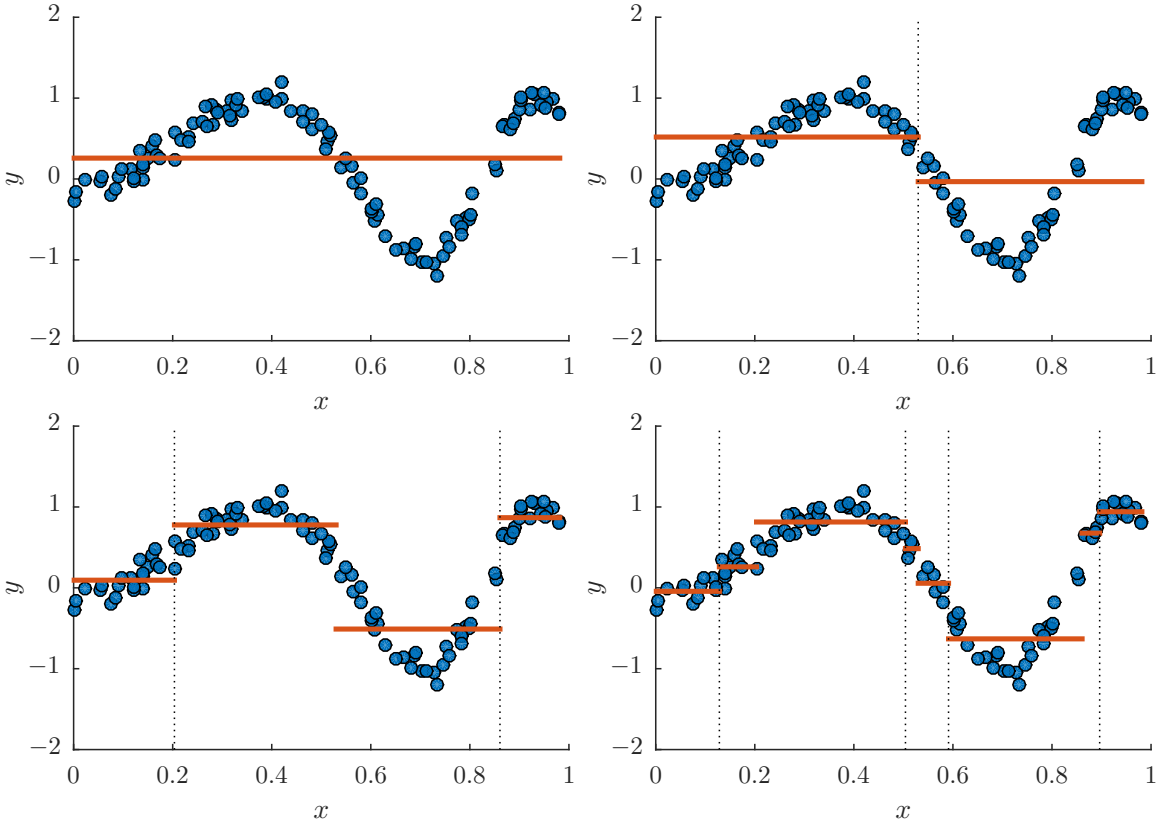


Fig. 8.7: Application of the regression tree method to the 1d example. First, all observations are assigned a constant y-value (top left pane). Then we consider various splits (along the x-axis) and select the one with the highest purity gain computed using the sum-of-squares impurity measure (top right). The method is applied recursively on each section until a stopping criterion is met (bottom row).

Problems

8.1. Fall 2013 question 14: 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 8.2. 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 No, $x_3 = 0$ or Yes, $x_3 = 1$	PAN
y	Lived after 5 years No, $y = 0$ or Yes, $y = 1$	Surv

Table 8.2: 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.

8.2. Fall 2013 question 13: We will use the decision tree given in Figure 8.9 to attempt to solve the classification problems given to the right of Figure 8.9 corresponding to the classification problem also considered in Figure 8.8. 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?

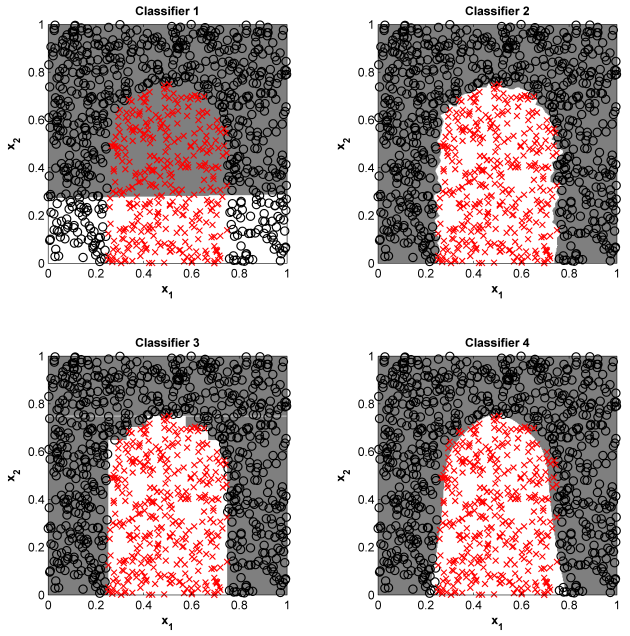


Fig. 8.8: The decision boundaries given in white and gray of four different classifiers used to separate red crosses from black circles.

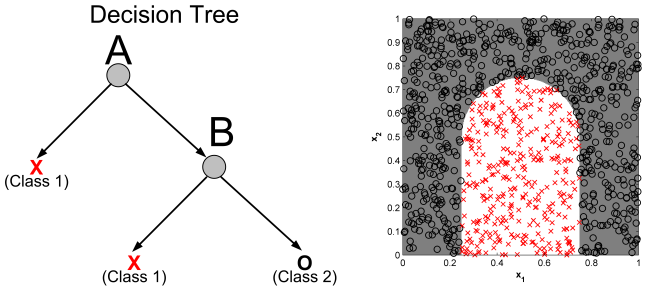


Fig. 8.9: A decision tree with two decisions denoted A and B that if given the right decisions can be used to perfectly separate the red crosses from black circles given in the classification problem to the right that was also considered in Figure 8.8.

A A=$\| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.25 \end{bmatrix} \|_\infty < 0.25$

B=$\| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \|_2 < 0.25$

B A=$\| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.25 \end{bmatrix} \|_1 < 0.25$

B=$\| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \|_2 < 0.25$

C A=$\| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.25 \end{bmatrix} \|_\infty < 0.25$

B=$\| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \|_1 < 0.25$

D A=$\| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.25 \end{bmatrix} \|_2 < 0.5$

B=$\| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \|_\infty < 0.5$

E Don't know.

8.3. Fall 2014 question 9: Consider a one-dimensional data set of features $\boldsymbol{X}$ and 3-class responses y shown in table 8.3; there are thus $N = 7$ observations.

$\boldsymbol{X}$	1	3	1	2	1	4	2
y	2	2	2	0	0	1	0

Table 8.3: Table of data and responses.

Suppose a decision tree is used to classify y on $\boldsymbol{X}$. Consider an attempted split at $\boldsymbol{X} = x_1 > 2.5$. 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.

8.4. Fall 2014 question 21:

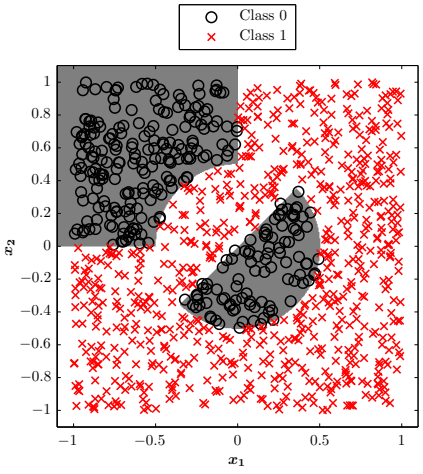


Fig. 8.10: Two-class classification problem

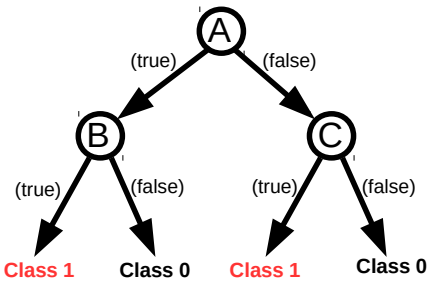


Fig. 8.11: Decision tree with 3 nodes A, B and C

Suppose we wish to solve the two-class classification problem in fig. 8.10 using a classification tree of the form fig. 8.11. What rules, acting on the coordinates $\boldsymbol{x} = (x_1, x_2)$, should be assigned to the three internal nodes A, B and C for the tree to give rise to the indicated decision boundary?

A $A : \|\boldsymbol{x}\|_2 \geq \frac{1}{2}$, $B : \left\| \boldsymbol{x} - \begin{bmatrix} -1 \\ 1 \end{bmatrix} \right\|_\infty > 1$
$C : \left\| \boldsymbol{x} - \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right\|_1 > 2$

B $A : \|\boldsymbol{x}\|_2 \geq \frac{1}{2}$, $B : \left\| \boldsymbol{x} - \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right\|_1 > 2$
$C : \left\| \boldsymbol{x} - \begin{bmatrix} -1 \\ 1 \end{bmatrix} \right\|_\infty > 1$

C $A : \left\| \boldsymbol{x} - \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right\|_1 > 2$, $B : \left\| \boldsymbol{x} - \begin{bmatrix} -1 \\ 1 \end{bmatrix} \right\|_\infty > 1$
$C : \|\boldsymbol{x}\|_2 \geq \frac{1}{2}$

D $A : \left\| \boldsymbol{x} - \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right\|_1 > 2$, $B : \|\boldsymbol{x}\|_2 \geq \frac{1}{2}$
$C : \left\| \boldsymbol{x} - \begin{bmatrix} -1 \\ 1 \end{bmatrix} \right\|_\infty > 1$

E Don't know.

9

Overfitting and performance evaluation

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 has S different models $\mathcal{M}_1, \ldots, \mathcal{M}_S$ 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 pre-occupied with. In this chapter, we will discuss common issues with model performance evaluation and provide the industry standard, cross-validation, for evaluating model performance and selecting between different models.

9.1 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 the three principal applications of cross-validation. We will use the simple linear regression model as a running example.

9.1.1 A simple example, linear regression

To provide a concrete example, consider the simple regression problem in fig. 9.1 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 and we wish to fit a regression model to the data. We assume we have access to three different models

$\mathcal{M}_1 = \{$1'st order polynomial, i.e. $\}f_{\mathcal{M}_1}(x, \boldsymbol{w}) = w_0 + w_1 x.$

$\mathcal{M}_2 = \{$2'nd order polynomial, i.e. $\}f_{\mathcal{M}_2}(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2.$

$\mathcal{M}_3 = \{$6'th order polynomial, i.e. $\}f_{\mathcal{M}_3}(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + w_5 x^5 + w_6 x^6.$

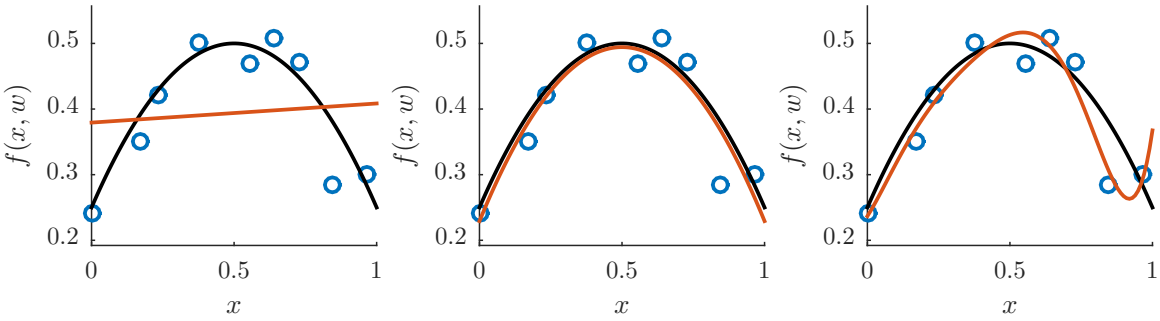


Fig. 9.1: A small dataset of nine observations generated from the true curve shown in the black line. The three red lines are three different linear regression models $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$ fitted to the dataset. Clearly the most complicated model $\mathcal{M}_3$ fits the dataset best, however, the second model $\mathcal{M}_2$ is better suited to account for the true black curve.

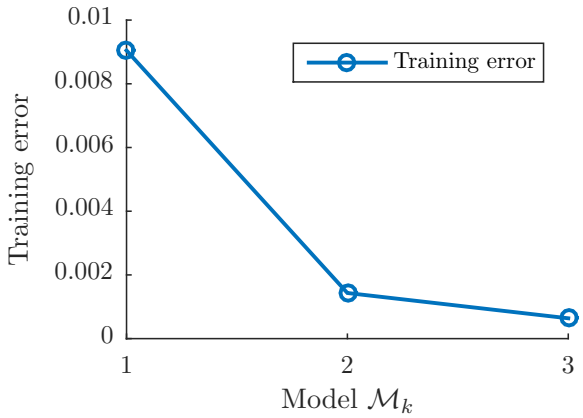


Fig. 9.2: Training error for each of the three models $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$ computed on the training data set. The most complicated model has the lowest training error.

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 $\mathcal{M}_k$ is

$$E_{\mathcal{M}_k}^{\text{train}} = \frac{1}{N^{\text{train}}} \sum_{i \in \mathcal{D}^{\text{train}}} (y_i - f_{\mathcal{M}_k}(x_i, \boldsymbol{w}))^2.$$

Here $f_{\mathcal{M}_k}$ is the model $\mathcal{M}_k$ 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. 9.2. 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 $\mathcal{M}_3$.

Nevertheless, we are not interested in a model like $\mathcal{M}_3$ because clearly it will not generalize well to new data. We say model $\mathcal{M}_3$ is *overfitting* the data. Clearly, 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:

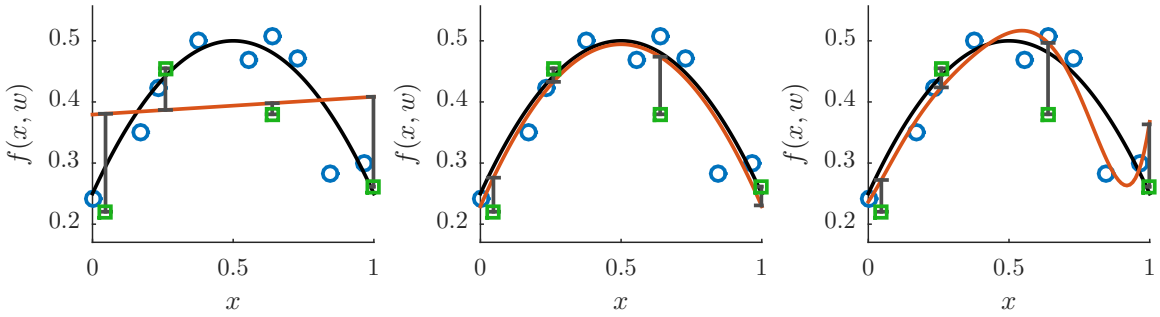


Fig. 9.3: The example from before with a test dataset of three new points. If the models are evaluated in terms of how the predict the *new* points $\mathcal{M}_2$ is preferred.

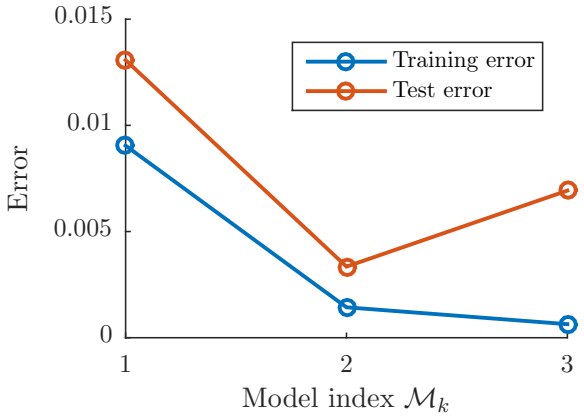


Fig. 9.4: Test error for each of the three models $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$ computed on the test data set. The test error correctly singles out model $\mathcal{M}_2$ as the better model.

> Never, ever should you estimate how well a model performs by its predictions on data it was trained upon.

However, lets assume we obtain access to some new data, the test data $\mathcal{D}^{\text{test}}$, indicated by the green squares in fig. 9.3. 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

$$E^{\text{test}}_{\mathcal{M}_k} = \frac{1}{N^{\text{test}}} \sum_{i \in \mathcal{D}^{\text{test}}} (y_i - f_{\mathcal{M}_k}(x_i, \boldsymbol{w}))^2.$$

Notice, $f_{\mathcal{M}_k}$ is the model that was fitted to the *training data*. The test error of each of these three models is shown in fig. 9.4. 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

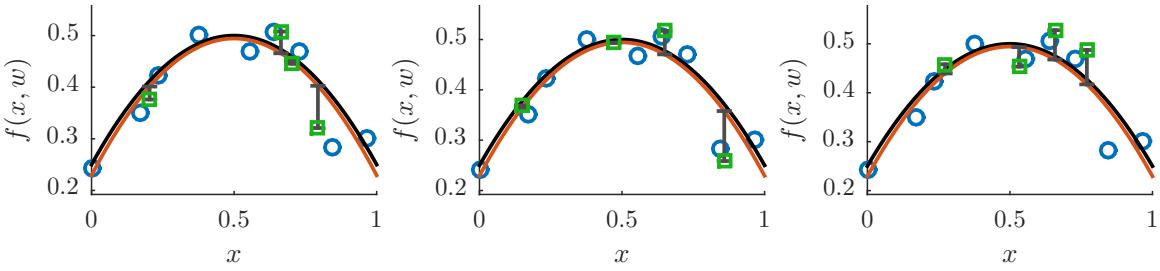


Fig. 9.5: The basic regression problem for model $\mathcal{M}_2$ and three random test sets. Since the test sets are random, the test error too will vary depending on the particulars of the test set. The generalization error overcomes this by averaging over all test sets.

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.

9.1.2 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 $\mathcal{D} = (\boldsymbol{X}, \boldsymbol{y})$. 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, $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_S$ and for each of these models we have access to a *loss function* $L(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i)$ which quantifies the error of predicting $\hat{\boldsymbol{y}}_i$ when the true value is $\boldsymbol{y}_i$. In the regression example the loss function was the least square error $L(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i) = \|\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i\|^2$, but in general the loss function will be defined according to the specific modeling purpose.

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 $\boldsymbol{f}_{\mathcal{M}}$ then we define the training and test errors as:

$$E_{\mathcal{M}}^{\text{train}} = \frac{1}{N^{\text{train}}} \sum_{i \in \mathcal{D}^{\text{train}}} L(\boldsymbol{y}_i, f_{\mathcal{M}}(\boldsymbol{x}_i))^2, \tag{9.1}$$

$$E_{\mathcal{M}}^{\text{test}} = \frac{1}{N^{\text{test}}} \sum_{i \in \mathcal{D}^{\text{test}}} L(\boldsymbol{y}_i, f_{\mathcal{M}}(\boldsymbol{x}_i))^2. \tag{9.2}$$

These definitions are similar except $\boldsymbol{f}_{\mathcal{M}}$ is fitted on the training set in both cases.

Generalization error

A problem with the training error is that it depends on the specific training set. Since we have to construct the training set ourselves, this makes the training error slightly random. This is illustrated in fig. 9.5 for the 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 random assuming we had an infinite amount of

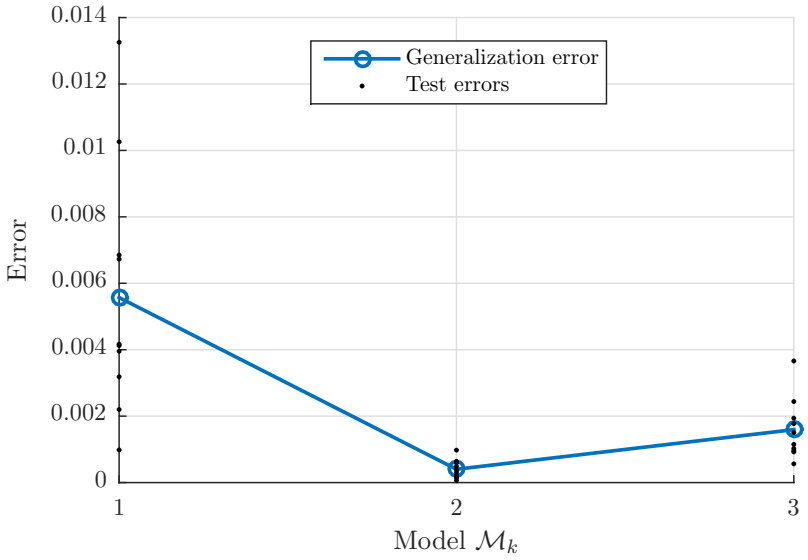


Fig. 9.6: Continuing the example, for the three models different test sets gives different test error as indicated by the black dots. The generalization error is simply the average over all possible test sets according to their probability of occurring.

data to test it on, i.e. the average of the test errors as illustrated in fig. 9.6. *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 $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ come from a distribution $p(\boldsymbol{x}, \boldsymbol{y})$ 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 $\boldsymbol{f}_{\mathcal{M}}$.
- The generalization error of model $\mathcal{M}$ is[1]

$$E^{\text{gen}}_{\mathcal{M}} = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})}\left[L(\boldsymbol{y}, \boldsymbol{f}_{\mathcal{M}}(\boldsymbol{x}))\right] \tag{9.3}$$

$$= \int L(\boldsymbol{y}, \boldsymbol{f}_{\mathcal{M}}(\boldsymbol{x}))p(\boldsymbol{x}, \boldsymbol{y})d\boldsymbol{x}d\boldsymbol{y}. \tag{9.4}$$

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.

9.1.3 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 models generalization error typically based on one of the following three approaches:

[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.

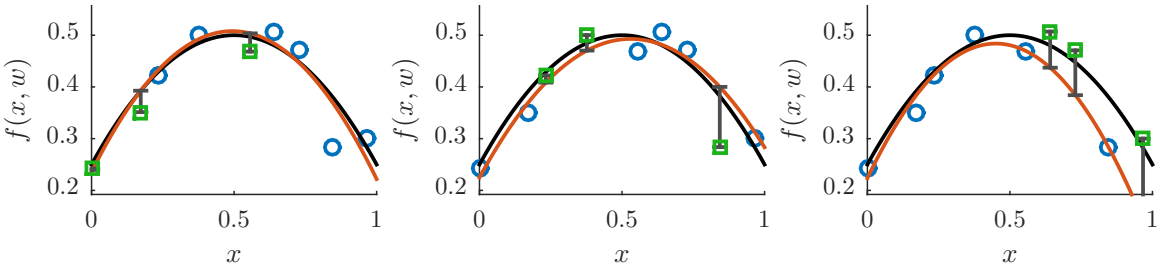


Fig. 9.7: Cross-validation applied to the model $\mathcal{M}_2$ using 3-fold cross-validation. Errors are estimated from the test data points and is indicated by the gray bars. Averaging all errors produce the estimate of the generalization error $\hat{E}^{\text{gen}}_{\mathcal{M}_2}$

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 $E^{\text{test}}_{\mathcal{M}}$ using the test data set $\mathcal{D}^{\text{test}}$ and formula eq. (9.2) and simply use the approximation:

$$E^{\text{gen}}_{\mathcal{M}} \approx E^{\text{test}}_{\mathcal{M}}.$$

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 $p(\boldsymbol{x}, \boldsymbol{y})$, we can expect the empirical average in eq. (9.2) to be quite close to the true average for the generalization error eq. (9.4). By recognizing these limitations we can provide two alternative methods which generally does better but are also computationally more demanding:

K-fold cross-validation

Notice, $E^{\text{test}}_{\mathcal{M}}$ is only an estimate of $E^{\text{test}}_{\mathcal{M}}$ and each data point is either in the test or training set. 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

$$\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \cdots \cup \mathcal{D}_K,$$

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 $E^{\text{test}}_{\mathcal{M},1}, \ldots, E^{\text{test}}_{\mathcal{M},K}$ and we now approximate:

$$E^{\text{gen}}_{\mathcal{M}} \approx \sum_{k=1}^{K} \frac{N^{\text{test}}_k}{N} E^{\text{test}}_{\mathcal{M},k},$$

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 can be considered the method which most faithfully represents the generalization error, the drawback is that it requires N models to be trained which can be very wasteful. For this reason, it is recommended to use K-fold cross-validation with for instance $K = 10$.

In the following we will denote by $\hat{E}^{\text{gen}}_{\mathcal{M}}$ an estimate of the generalization error $E^{\text{gen}}_{\mathcal{M}}$ computed by any of the three above techniques. An illustration where basic 3-fold cross-validation to the linear-regression example is given in fig. 9.7. 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.

9.1.4 Cross-validation for parameter selection

We will accept that the generalization error eq. (9.4) 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 $\mathcal{M}_1, \ldots, \mathcal{M}_S$ 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, estimate the cross-validation error $\hat{E}^{\text{gen}}_{\mathcal{M}_1}, \ldots, \hat{E}^{\text{gen}}_{\mathcal{M}_S}$ using cross-validation.
- Select the optimal model $\mathcal{M}_{s^*}$ as that with the lowest error:

$$s^* = \arg\min_s \hat{E}^{\text{gen}}_{\mathcal{M}_s}$$

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 $\mathcal{D}_1, \ldots, \mathcal{D}_K$) is used for all models. Since the resulting method is so important it is provided as an explicit algorithm in algorithm 4.

An illustration of 3-fold cross-validation for parameter selection in the linear-regression example is given in fig. 9.8. Eac of the columns correspond to a fold and each of the rows to a model. In fig. 9.9 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 $\mathcal{M}_2$.

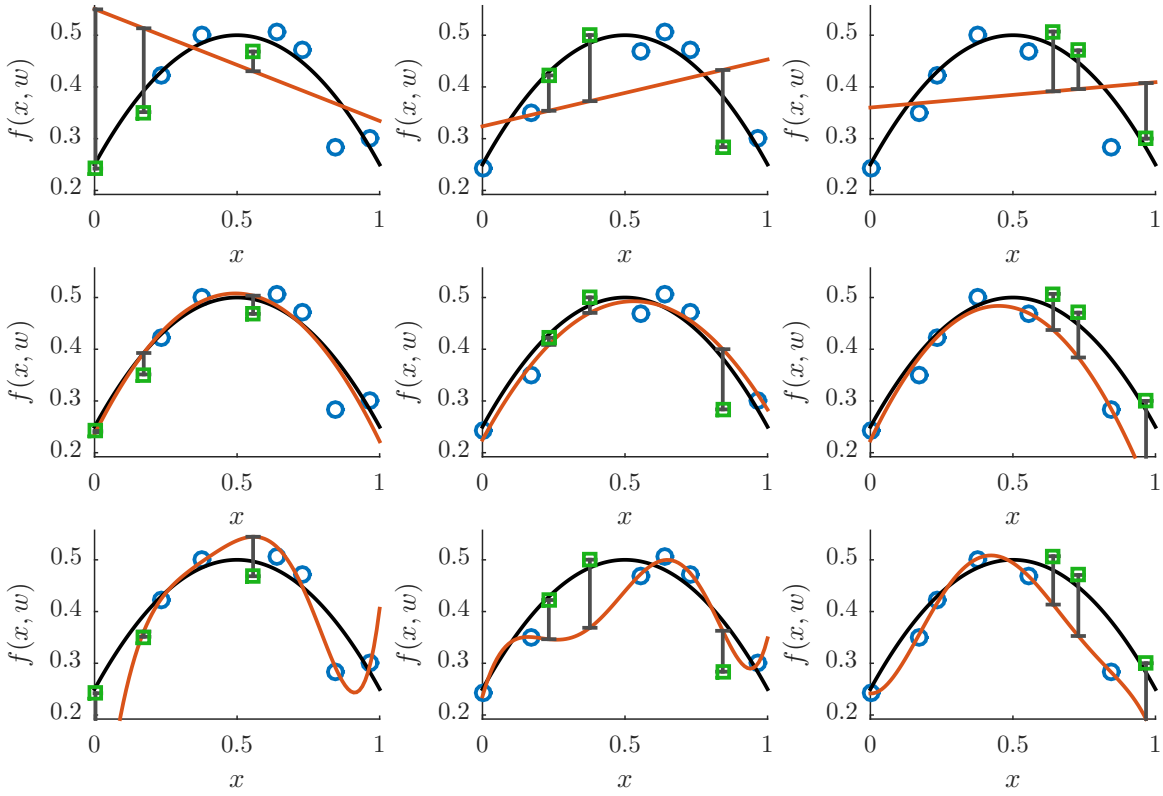


Fig. 9.8: Cross-validation applied to model-selection for the linear regression models. Each row corresponds to one of the three models, and each column to the estimate of the test error on that particular fold.

9.1.5 Two-layer cross-validation

Lets 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 $\mathcal{M}_{k^*}$. A tempting way to accomplish this is to apply K-fold cross-validation to estimate the generalization error for each model $\mathcal{M}_k$ 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 fig. 9.10 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. 9.10). These too are estimates of the true generalization error (black line), but they are independent of the red dots. However, since we always select the red dot with the *lowest* error, and the blue dots

Algorithm 4: K-fold cross-validation for model selection

Require: K, the number of folds in the cross-validation loop
Require: $\mathcal{M}_1, \ldots, \mathcal{M}_S$. The S different models to select between
Ensure: $\mathcal{M}_{s^*}$ the optimal model suggested by cross-validation
  for $k = 1, \ldots, K$ splits **do**
    Let $\mathcal{D}_k^{\text{train}}$, $\mathcal{D}_k^{\text{test}}$ the k'th split of $\mathcal{D}$
    for $s = 1, \ldots, S$ 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 $\mathcal{D}_s^{\text{test}}$
    end for
  end for
  For each s compute: $\hat{E}_{\mathcal{M}_s}^{\text{gen}} = \sum_{k=1}^{K} \frac{N_k^{\text{test}}}{N} E_{\mathcal{M}_s,k}^{\text{test}}$
  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

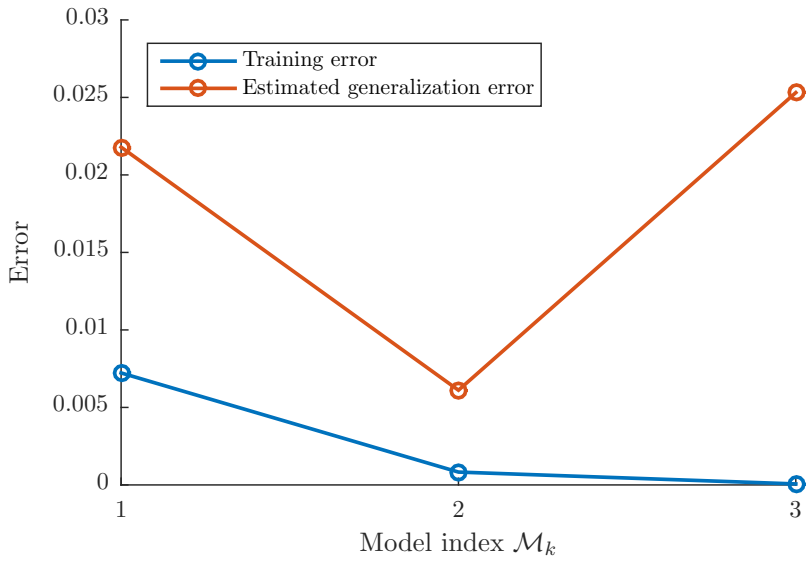


Fig. 9.9: Training error and the cross-validation estimate of the generalization error. The estimate of the generalization error is simply the averages of the errors in fig. 9.8 over all 3 folds as dictated by the cross-validation method for model selection. The model with the lowest estimated generalization error is $\mathcal{M}_2$ even though it does not have the lowest training error.

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 $\mathcal{M}^*$ based on the data we have actually changed the

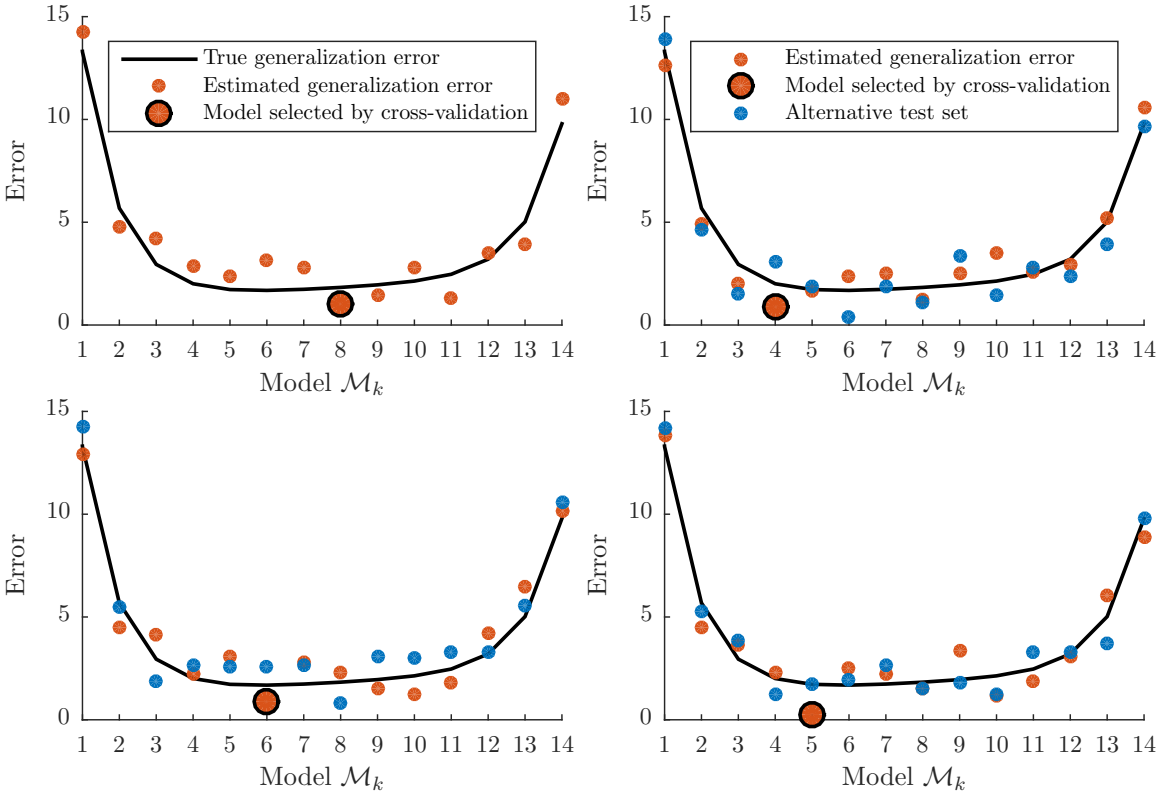


Fig. 9.10: Top right: The true generalization error of 14 models is indicated by the black line and estimates of the generalization error (computed using cross-validation) are indicated by red dots. Standard cross-validation then selects the model with the lowest (estimated) cross-validation error (indicated 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.

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 estimates $\hat{E}_k^{\text{gen}}$
- Select $\mathcal{M}^*$ as the optimal model $\mathcal{M}_{k^*}$ where $k^* = \arg\min_k \hat{E}_k^{\text{gen}}$

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 $\mathcal{M}_{k^*}$ as above and one for estimating performance, the resulting procedure is known as two-layer cross-validation. The method can be sketched as follows:

Algorithm 5: Two-level cross-validation

Require: K_1, K_2, folds in outer, and inner cross-validation loop respectively
Require: $\mathcal{M}_1, \ldots, \mathcal{M}_S$: The S different models to cross-validate
Ensure: $\hat{E}^{\text{gen}}$, the estimate of the generalization error
for $i = 1, \ldots, K_1$ **do**
Outer cross-validation loop. First make the outer split into K_1 folds
Let $\mathcal{D}_i^{\text{par}}$, $\mathcal{D}_i^{\text{val}}$ the i'th split of $\mathcal{D}$
for $j = 1, \ldots, K_2$ **do**
Inner cross-validation loop. Use cross-validation to select optimal model
Let $\mathcal{D}_j^{\text{train}}$, $\mathcal{D}_j^{\text{test}}$ by the j'th split of $\mathcal{D}_i^{\text{par}}$
for $s = 1, \ldots, S$ **do**
Train $\mathcal{M}_s$ on $\mathcal{D}_j^{\text{train}}$
Let $E_{\mathcal{M}_s,j}^{\text{test}}$ be the *test error* of the model $\mathcal{M}_s$ when it is *tested* on $\mathcal{D}_j^{\text{test}}$
end for
end for
For each s compute: $\hat{E}_s^{\text{gen}} = \sum_{j=1}^{K_2} \frac{|\mathcal{D}_j^{\text{test}}|}{|\mathcal{D}_i^{\text{par}}|} E_{\mathcal{M}_s,j}^{\text{test}}$
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^{test} be the *test error* of the model $\mathcal{M}^*$ when it is *tested* on $\mathcal{D}_i^{\text{val}}$
end for
Compute the estimate of the generalization error: $\hat{E}^{\text{gen}} = \frac{1}{K_1} \sum_{i=1}^{K_1} \frac{|\mathcal{D}_i^{\text{val}}|}{N} E_i^{\text{test}}$

- For $i = 1, \ldots, K_1$ cross-validation iterations, split the data $\mathcal{D}$ into a training set $\mathcal{D}_i^{\text{par}}$ and a validation set $\mathcal{D}_i^{\text{val}}$
- For each iteration, find the optimal value k^* using K_2-fold cross-validation on $\mathcal{D}_i^{\text{par}}$.
- Train the model $\mathcal{M}^* = \mathcal{M}_{k^*}$ on on the full parameter-estimation 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 validation set $\mathcal{D}_i^{\text{val}}$
- Estimate the generalization error as $\hat{E}^{\text{gen}} = \sum_{i=1}^{K_1} \frac{|\mathcal{D}_i^{\text{val}}|}{N} E_{\mathcal{M}^*,i}^{\text{test}}$

Again, since this method is so important it is worth providing it in pseudo code as algorithm 5.

9.2 Sequential feature selection

Consider a dataset where observations $\boldsymbol{x}_i$ correspond to patients and we wish to predict a patients 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 the attribute $\boldsymbol{x} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T$ we could just as well consider the smaller dataset: $\boldsymbol{x} = \begin{bmatrix} x_1 & x_3 \end{bmatrix}^T$. 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.

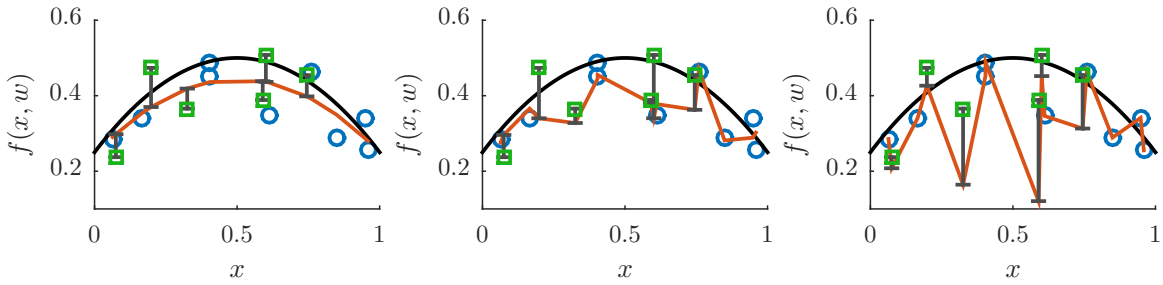


Fig. 9.11: A regularized linear regression model is fitted to the dataset of nine observations and six test observations. The different plots correspond to adding more "junk" attributes, i.e. attributes where the values are just random. As more random attributes are added, the model better fit the training set but does worse on the test set.

- 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.

Lets 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:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \; x_1^2 \end{bmatrix} \tag{9.5}$$

However, we now add "junk" attributes to the dataset and considers

$$\boldsymbol{x} = \begin{bmatrix} x_1 \; x_1^2 \; x_3 \; x_4 \; \ldots \; x_{S+2} \end{bmatrix}$$

where S is the number of junk attributes added to the dataset. Thus $S = 0$ will correspond to eq. (9.5) 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. 9.11. 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. 9.12 for $S = 0, \ldots, 8$ and the three specific values shown in fig. 9.11 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

$$\begin{array}{cccc} \mathcal{M}_{123} = \begin{bmatrix} x_1 \; x_2 \; x_3 \end{bmatrix} & \mathcal{M}_{12} = \begin{bmatrix} x_1 \; x_2 \end{bmatrix} & \mathcal{M}_{13} = \begin{bmatrix} x_1 \; x_3 \end{bmatrix} & \mathcal{M}_{23} = \begin{bmatrix} x_2 \; x_3 \end{bmatrix} \\ \mathcal{M}_1 = \begin{bmatrix} x_1 \end{bmatrix} & \mathcal{M}_2 = \begin{bmatrix} x_2 \end{bmatrix} & \mathcal{M}_3 = \begin{bmatrix} x_3 \end{bmatrix} & \mathcal{M}_\cdot = \begin{bmatrix} \bullet \end{bmatrix}, \end{array}$$

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.

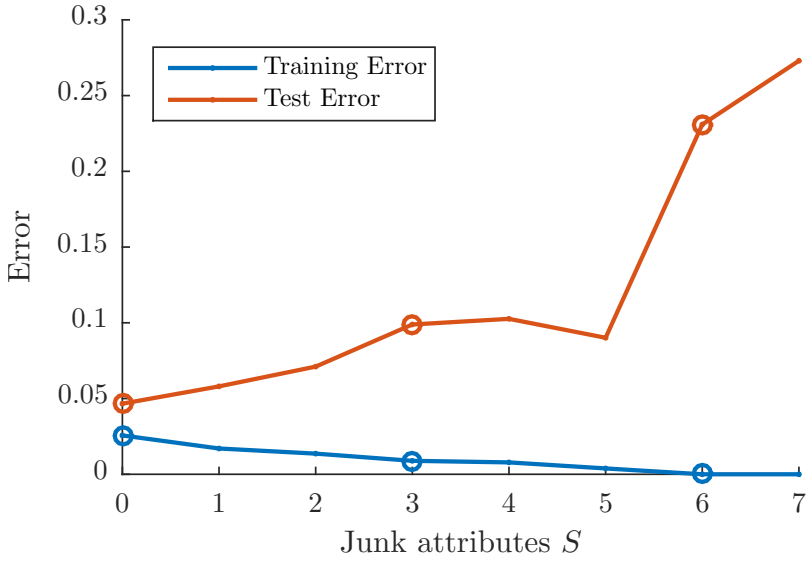


Fig. 9.12: The training and test error for different number of junk attributes in the example of fig. 9.11. The circles indicate the particular values shown in fig. 9.11.

9.2.1 Forward Selection

In *forward selection*, we first consider a model with no features

$$\mathcal{M}_\cdot = [\bullet]$$

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:

$$\mathcal{M}_\bullet = [\bullet], \quad \mathcal{M}_1 = [x_1], \quad \mathcal{M}_2 = [x_2], \ldots, \mathcal{M}_K = [x_M].$$

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 $M_\bullet$ is selected the process terminates. Else, this procedure is now repeated by evaluating the M models corresponding to

$$\mathcal{M}_i = [x_i], \quad \mathcal{M}_{1i} = [x_1\ x_i], \ldots \mathcal{M}_{i-1,i} = [x_{i-1}\ x_i] \ldots \mathcal{M}_{iM} = [x_i\ x_M]$$

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: $\mathcal{M}_{12\ldots M}$.

Example of forward selection

Lets 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 9.1. Forward selection now proceeds as follows

Model	$\hat{E}^{\text{gen}}$
$\mathcal{M}_\bullet$	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 9.1: The *estimated generalization error* $\hat{E}^{\text{gen}}$ as estimated by cross-validation for models trained on different subsets of the features x_1, x_2, x_3 and x_4.

- 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.

9.2.2 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 $\mathcal{M}_{12\ldots M}$ 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

$$(1) + (M) + (M-1) + (M-2) + \cdots + (1) = \frac{M(M+1)}{2} + 1$$

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.

9.3 Quantitative evaluation and comparison of classifiers

Cross-validation provides a way of estimating the generalization error of a model. However, since cross validation only provides an *estimate* it will be influenced by random fluctuations due to the particulars of the training/test sets. Therefore, if we have to make decisions based on the generalization error, this randomness has to be taken into account: we want to know when we can fairly confidently say one model performs better than another and when a difference in performance may just as well be due to random fluctuations. We will consider this problem in two common situations:

First task: performance of a single classifier Suppose a single classifier is evaluated on a test set containing N observations and classify m observations correctly. The *true* accuracy θ can then be *estimated* as

$$\hat{\text{Acc}} = \hat{\theta} = \frac{m}{N}.$$

Especially if N is very small, this estimated accuracy will behave slightly random and not be equal to the *true* accuracy. We essentially want to answer this question: Given N and m, in what interval is it *credible* to think the true accuracy lies?

Second task: comparing two models Suppose we are given two models A and B. We estimate the generalization error of both of these models using K-fold cross-validation and compute the difference:

$$E_A^{\text{gen}} - E_B^{\text{gen}} = \hat{z}$$

If $\hat{z} < 0$ this would indicate model A is better than model B, however once again this number (obtained from cross-validation) will behave slightly random and not be equal to the "true" difference in generalization error. What we want to know is this: Given the behaviour of the two models on the cross-validation splits, what interval can we expect the true difference of the generalization errors to lie within?

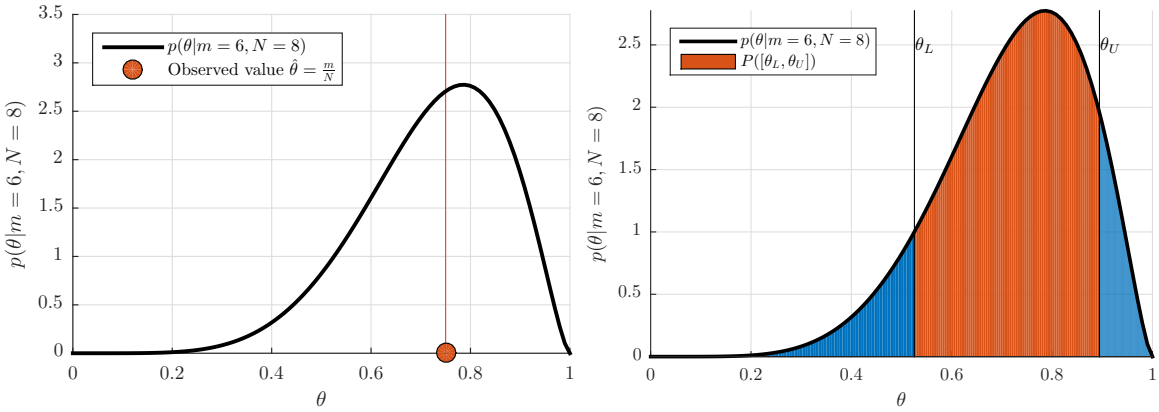


Fig. 9.13: (Left:) Suppose we observe a value $\hat{\theta} = \frac{m}{N} = \frac{3}{4}$ of the accuracy. We don't know what the true accuracy θ is, but we can assume it follows a probability distribution $p(\theta|N,m)$. (Right:) The credibility interval $[\theta_L, \theta_U]$ is formed by selecting θ_L, θ_U such that (i) there is a chance of $P([\theta_L,\theta_U]) = 1-\alpha$ that θ is in the credibility interval and (ii) it is equally likely θ is in the two tail intervals (each has area $\frac{\alpha}{2}$).

These two situations may appear superficially different, but it turns out the solution is nearly identical. We will therefore discuss the general features of the problem using the first task as an example and later return to the second task. The solution we present involves nothing else than two rules of probability we have already seen, however, we have to apply these rules to unfamiliar distributions which involves integration. It is a good idea to treat these integrals as something that happens automatically and focus on the general flow of the solution. Also keep in mind the derivations are starred and only the solutions eq. (9.12) and eq. (9.15) (along with the two examples where confidence intervals are applied) is main material.

9.3.1 The general solution

Let's consider the first problem in more detail. The true accuracy θ (which we don't know) is the chance of classifying a new observation correctly. Since we don't know what θ is, we should assume it follows a probability distribution informed by which value of N and m we happened to observe:

$$p(\theta|N,m) = \{\text{So far unknown distribution of the true accuracy } \theta\}\,. \tag{9.6}$$

As an example, assume we observed $N = 8$ and $m = 6$. Then we might expect the probability distribution $p(\theta|N,m)$ to look similar to fig. 9.13 (left pane), where we have also plotted the observed value of the accuracy $\frac{m}{N} = \frac{3}{4}$. Our first task is to translate this distribution into an interval $[\theta_L, \theta_U]$ where we can expect θ to be with high probability. This is fairly simple: Recall that for any interval $[\theta_L, \theta_U]$ the chance θ is in this interval according to a probability density is (by the definition of probability densities, see eq. (5.8))

$$P(\theta \text{ in the interval } [\theta_L,\theta_U]) = P([\theta_L,\theta_U]) = \int_{\theta_L}^{\theta_U} p(\theta|N,m).$$

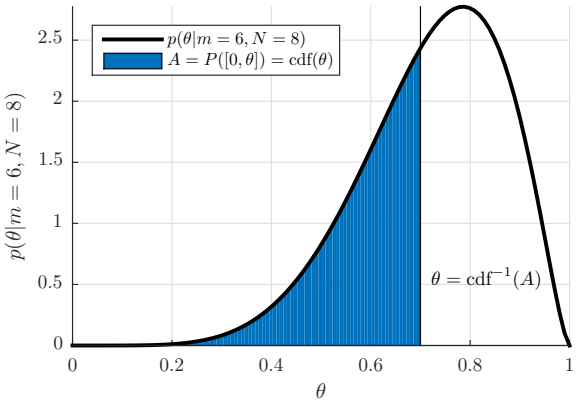


Fig. 9.14: Illustration of the cumulative density function. For a given value of θ, then $\text{cdf}(\theta)$ is the area under the curve up to θ. For a given value of the area A, then the *inverse* of the cumulative density function $\text{cdf}^{-1}(A)$ computes θ such that $\text{cdf}(\theta) = A$.

In fig. 9.13 (right pane) this probability is illustrated as the red area. This immediately gives us an idea: Suppose we select a (high) probability $1-\alpha$ where for instance $\alpha = 0.2$ in our example. We can then choose θ_L, θ_U such that (i) $P([\theta_L, \theta_U]) = 1-\alpha = 80\%$ and (ii) the two remaining blue areas each has an area of $\frac{\alpha}{2} = 10\%$ such that the interval is "symmetric" with respect to the curve. Such a symmetric interval is known as a *credibility interval*. We now only need a slight bit of notation to compute θ_L and θ_U. First we define the *cumulative density function* of $p(\theta|N,m)$ which for a given θ computes the area of the part of the curve which lies left of θ, corresponding to the blue area in fig. 9.14

$$\text{cdf}(\theta) = \int_0^\theta p(\theta'|N,m)d\theta'.$$

We then see we can express the areas of the three regions as (we use that the sum of the three areas must be 1)

$$P([0,\theta_L]) = \text{cdf}(\theta_L), \quad P([\theta_L,\theta_U]) = \text{cdf}(\theta_U) - \text{cdf}(\theta_L), \quad P([\theta_U,1]) = 1 - \text{cdf}(\theta_U).$$

Since we want the area of the blue region to be $1-\alpha$ and the two others to be equal we can therefore conclude that

$$\text{cdf}(\theta_L) = \frac{\alpha}{2}, \quad \text{and} \quad \text{cdf}(\theta_U) = 1 - \frac{\alpha}{2}. \tag{9.7}$$

To proceed, we assume there is a way to compute the *inverse* of the cumulative density function illustrated in fig. 9.14. That is, we assume there is a new function cdf^{-1} which, given an area under the curve A, can find the value of θ such that the integral from $[0,\theta]$ is equal to that area: $A = \text{cdf}(\theta) = \int_0^\theta p(\theta')d\theta'$:

$$\text{cdf}^{-1}(A) = \{\text{the value } \theta \text{ such that } \text{cdf}(\theta) = A\}. \tag{9.8}$$

This solves the problem! Assuming we have somehow found $p(\theta|N,m)$, the credibility interval is simply the interval $[\theta_L, \theta_U]$ such that:

Table 9.2: Intermediate computation and credibility interval for the example in fig. 9.15

	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

$$\theta_L = \text{cdf}^{-1}\left(\frac{\alpha}{2}\right), \quad \text{and} \quad \theta_U = \text{cdf}^{-1}\left(1-\frac{\alpha}{2}\right), \tag{9.9}$$

which guarantees that the chance θ is in the interval $[\theta_L, \theta_U]$ is $P([\theta_L, \theta_U]) = 1-\alpha$. In the previous sections we selected $\alpha = 0.2$, however it is more common to select $\alpha = 0.05$. In the two following sections we will derive credibility intervals for the two above problems which in both cases is simply about learning the distribution p using Bayes' theorem.

9.3.2 First task: Evaluation of a single classifier*

To recap, let N be the number of test observations and m the number of times the model correctly classify a test observation. We assume θ is the (unknown) probability the model correctly classify each observation and we therefore has to learn $p(\theta|N, m)$. This problem is exactly equal to the Bernoulli coin we encountered in section 5.5 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. Using Bayes' theorem and eq. (5.29) we obtain:

$$p(\theta|m, N) = \frac{p(m|\theta, N)p(\theta)}{p(m|N)} = \frac{\theta^m(1-\theta)^{N-m}p(\theta)}{p(m|N)}. \tag{9.10}$$

We need two more ingredients at this point. The first is the *beta distribution* which is simply defined as[2]

$$\text{Beta distribution: } \text{Beta}(\theta|a, b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\theta^{a-1}(1-\theta)^{b-1}.$$

The second is the prior distribution $p(\theta)$. In our original analysis of the biased coin in eq. (5.29) we assumed $p(\theta) = 1$, and can notice that by simply plugging in the values this is equal to $p(\theta) = 1 = \text{Beta}(\theta|1, 1)$. The prior we will consider here is the so-called Jeffrey prior:[3]

$$\text{Jeffrey prior: } p(\theta) = \text{Beta}\left(\theta|\frac{1}{2}, \frac{1}{2}\right) = \frac{1}{\Gamma(\frac{1}{2})^2}\theta^{-\frac{1}{2}}(1-\theta)^{-\frac{1}{2}}.$$

Why is the beta distribution important? If we consider the terms in the beta distribution that involve θ, we see that they are exactly equal to the Bernoulli likelihood in eq. (9.10). It can therefore be shown that:

[2] $\Gamma(x)$ is the Gamma function. If x is an integer then $\Gamma(x) = (x-1)!$ and ! denotes the factorial function, i.e., $4! = 4\cdot3\cdot2\cdot1 = 24$, and $0! = 1$. For further details see `https://en.wikipedia.org/wiki/Gamma_function`

[3] This choice may appear odd (why $\frac{1}{2}$ and not something else?), but it has a strong motivation in the literature as the "true" uninformative prior.

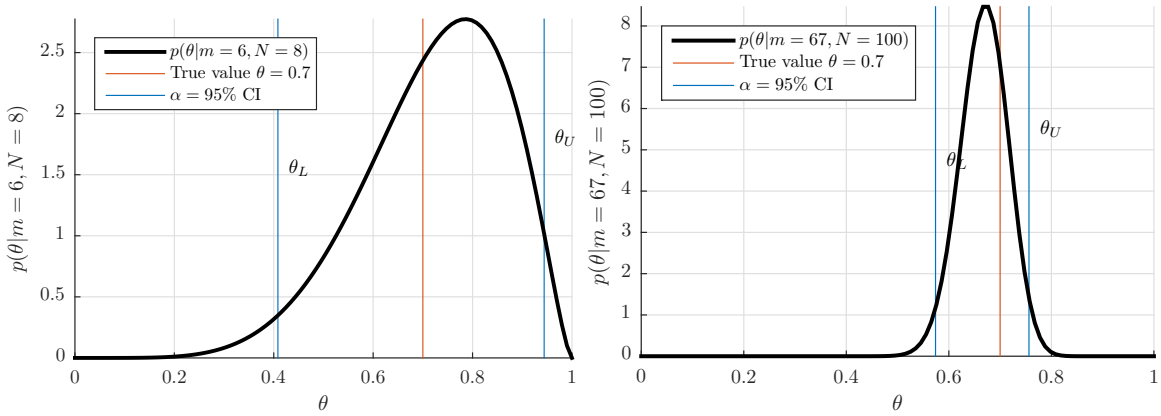


Fig. 9.15: Credibility interval for $\alpha = 0.05$ for a single classifier in two cases, $N = 8, m = 6$ and $N = 100, m = 67$ (left and right pane). The credibility interval indicate where (under the model assumptions) we can expect θ to be with probability $1 - \alpha = 95\%$.

$$p(\theta|m,N) = \frac{\theta^m(1-\theta)^{N-m}p(\theta)}{p(m|N)} = \frac{1}{\Gamma(\frac{1}{2})^2}\frac{\theta^{m+\frac{1}{2}-1}(1-\theta)^{N-m+\frac{1}{2}-1}}{p(m|N)}$$
$$= \text{Beta}(\theta|a,b), \quad a = m + \frac{1}{2}, \text{ and } b = N - m + \frac{1}{2}. \tag{9.11}$$

What we have done so far is simply re-doing the computation in eq. (5.29) with a slightly more general prior. In fact, it is easy to see that the original result for the coin, eq. (5.31), is equal to a $\text{Beta}(\theta|m+1, N-m+1)$ distribution. Since we now have the formula for the posterior distribution eq. (9.6) all subsequent comments apply. The cdf of is simply

$$\text{cdf}(\theta|a,b) = \int_0^\theta \text{Beta}(\theta|a,b)$$

and if we let $\text{cdf}_B^{-1}(\theta|a,b)$ be the inverse cumulative distribution function for the beta distribution[4] the *Jeffrey creditability interval* is simply[5]

$$\theta_L = \text{cdf}_B^{-1}\left(\frac{\alpha}{2}|a,b\right), \ \theta_U = \text{cdf}_B^{-1}\left(1-\frac{\alpha}{2}|a,b\right) \text{ where: } a = m + \frac{1}{2} \text{ and } b = N - m + \frac{1}{2}. \tag{9.12}$$

Example 1:

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 $m = 6$, and in the other $N = 100$ and

[4] Special functions to evaluate cdf_B^{-1} are build into all popular mathematical environments. In matlab: $\text{cdf}_B^{-1}(A|a,b) =$ `betainv`(A,a,b), in R: $\text{cdf}_B^{-1}(A|a,b) = \text{qbeta}(A,a,b)$ and in Python: $\text{cdf}_B^{-1}(A|a,b) =$ `beta.ppf`(A,a,b)

[5] There is one small wrinkle: If $m = 0$ simply let $\theta_L = 0$ and if $m = N$ let $\theta_U = 1$

Table 9.3: $K = 5$ cross-validation example

k	1	2	3	4	5
$E_{A,k}$	1.10	1.64	-0.02	-0.38	2.30
$E_{B,k}$	0.74	0.79	0.77	-1.67	0.34
z_k	0.36	0.85	-0.79	1.29	1.96

$m = 67$. In both cases we compute the 95% creditability interval using eq. (9.12), the intermediate calculations can be found in table 9.2 and the posterior curve of θ is illustrated in fig. 9.15. We see the true value of $\theta = 0.7$ lies within both intervals.

9.3.3 Second task: Comparing two classifiers*

Suppose we have two models $\mathcal{M}_A$ and $\mathcal{M}_B$ and we want to know which is better. An idea is to estimate the generalization error of the two models E_A^{gen} and E_B^{gen} using K-fold cross-validation and then see which generalization error is larger or, equivalently, if the difference $E_A^{\text{gen}} - E_B^{\text{gen}}$ is greater or smaller than zero. Once more, we should worry that since the estimates of the generalization error has a small randomness, then if the estimates are closely together it might not be significant that one happens to be larger than the other.

Some splits into training and test in cross-validation will be harder than others, and we will therefore assume we re-use the K split into training and test data for both models. If we assume the splits are equally large, we obtain:

$$E_A^{\text{gen}} - E_B^{\text{gen}} = \sum_{k=1}^{K} \frac{1}{K} z_k, \quad z_k = E_{A,k}^{\text{test}} - E_{B,k}^{\text{test}}$$

We can therefore formulate the problem as follows: There is a "true" difference of the generalization error $u = E_A^{\text{gen}} - E_B^{\text{gen}}$ and the K folds corresponds to K noisy observations of this difference $z_1, \cdots, z_K$, see table 9.3 for an example. Since we don't know u, it follows it must have a distribution

$$p(u|\boldsymbol{z}), \quad \boldsymbol{z} = z_1, \ldots, z_K \tag{9.13}$$

The plan is exactly as before: First, we find this distribution using Bayes' theorem, and then we simply use to write down the creditability interval. Assume we know u, then each z_k is normal distributed around u with variance σ:

$$p(z_1, \ldots, z_K|u, \sigma^2) = \prod_{k=1}^{K} \mathcal{N}(z_k|u, \sigma^2)$$

We introduce $\tau = \sigma^2$ to make the math easier to write up. If we just apply Bayes' theorem we obtain:

$$p(u, \tau|\boldsymbol{z}) = \frac{p(\boldsymbol{z}|u, \tau)p(u, \tau)}{p(\boldsymbol{z})}.$$

Once again, we can focus on the nominator since the denominator is a normalization constant. The prior distribution has to be specified and a particularly important prior is again the Jeffrey prior $p(u, \tau) = \frac{1}{\tau}$[6]. Plugging this in gives, (we will recover the normalization constant later):

[6] This prior is not normalized because $\int p(u, \tau) du d\tau = \infty$, however it can be understood as a relative statement about how likely particular values of u, τ are compared to each other

Table 9.4: Credibility interval eq. (9.15) computed for example in table 9.3

	K	ν	$\bar{z}$	$\tilde{\sigma}$	θ_L	θ_U
Case 1	5	4	0.7340	0.46	-0.55	2.02
Case 2	10	9	1.4960	0.40	0.60	2.40

$$p(u,\tau|\boldsymbol{z}) \propto p(\boldsymbol{z}|u,\tau)p(u,\tau) = \left[\prod_{k=1}^{K}\mathcal{N}(z_k|u,\tau)\right]\frac{1}{\tau}.$$

We can then obtain $p(u|\boldsymbol{z})$ (up to a constant) by marginalization. The integral is difficult analytical, but notice it is just a 1-dimensional integral which can be found in an integration table:

$$p(u|\boldsymbol{z}) = \int p(u,\tau|\boldsymbol{z})d\tau \propto \int \frac{1}{\tau}\prod_{k=1}^{K}\mathcal{N}(z_k|u,\tau)d\tau \propto \left(1+\frac{1}{\nu}\left[\frac{u-\bar{x}}{\tilde{\sigma}}\right]^2\right)^{-\frac{\nu+1}{2}}, \tag{9.14}$$

where we have defined $\bar{z} = \frac{1}{K}\sum_{k=1}^{K} z_k$, $\nu = K-1$ and $\tilde{\sigma} = \sqrt{\sum_{k=1}^{K}\frac{(z_k-\bar{z})^2}{K(K-1)}}$, see table 9.4 for these quantities computed from the $K=5$ cross-validation example of table 9.3. We still have to find the normalization constant, however the right-hand side of eq. (9.14) is proportional to the so-called *non-standardized Student's t-distribution* defined as:

$$p_{\text{stud}-t}(x|\nu,\mu,\sigma) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)\sqrt{\pi\nu\sigma^2}}\left(1+\frac{1}{\nu}\left[\frac{x-\mu}{\sigma}\right]^2\right)^{-\frac{\nu+1}{2}},$$

and so we can conclude that $p(u|\boldsymbol{z}) = p_{\text{stud}-t}(u|\nu,\bar{z},\tilde{\sigma})$. Since we have found an expression for $p(u|\boldsymbol{z})$ all comments above apply where we simply have to use the cumulative distribution function of the students t-distribution rather than for the Beta distribution. If we denote this by $\text{cdf}_{st}^{-1}(A|\nu,\bar{z},\tilde{\sigma})$ [7] we can write up the credibility interval corresponding to section 9.3.3 as $[z_L, z_U]$ where

$$z_L = \text{cdf}_{st}^{-1}(\frac{\alpha}{2}|\nu,\bar{z},\tilde{\sigma}),\ \ z_U = \text{cdf}_{st}^{-1}(1-\frac{\alpha}{2}|\nu,\bar{z},\tilde{\sigma}). \tag{9.15}$$

Example 2:

As an example, suppose we apply $K=5$ fold cross validation to two classifiers. The error estimates for each of the 5 folds can be found in table 9.3 along with the differences z_k (recall the same splits in test and training data is re-used for both models). The quantities and bounds are computed using eq. (9.15) and can be found in table 9.4 (top row). The resulting credibility interval for $\alpha = 0.05$ is visualized in fig. 9.16 (left pane) along with the z_k's. Notice the creditability interval is for the mean u and therefore it is not unexpected the error for some of the folds fall outside; the red line indicate the "true" difference of the generalization error of the models and is shown to indicate the high degree of variability. The same experiment is repeated for $K=10$ and we see that for

[7] Special functions to evaluate $\text{cdf}_{st}^{-1}(A|\nu,\bar{z},\tilde{\sigma})$ are build into all popular mathematical environments. In matlab: $\text{cdf}_{st}^{-1}(A|\nu,\bar{z},\tilde{\sigma}) = \bar{z}+\tilde{\sigma}\,\texttt{tinv}(A,\nu)$, in R: $\text{cdf}_{st}^{-1}(A|\nu,\bar{z},\tilde{\sigma}) = \bar{z}+\tilde{\sigma}\,\texttt{qt}(A,\nu)$ and in Python: $\text{cdf}_{st}^{-1}(A|\nu,\bar{z},\tilde{\sigma}) = \bar{z}+\tilde{\sigma}\,\texttt{t.ppf}(A,\nu)$

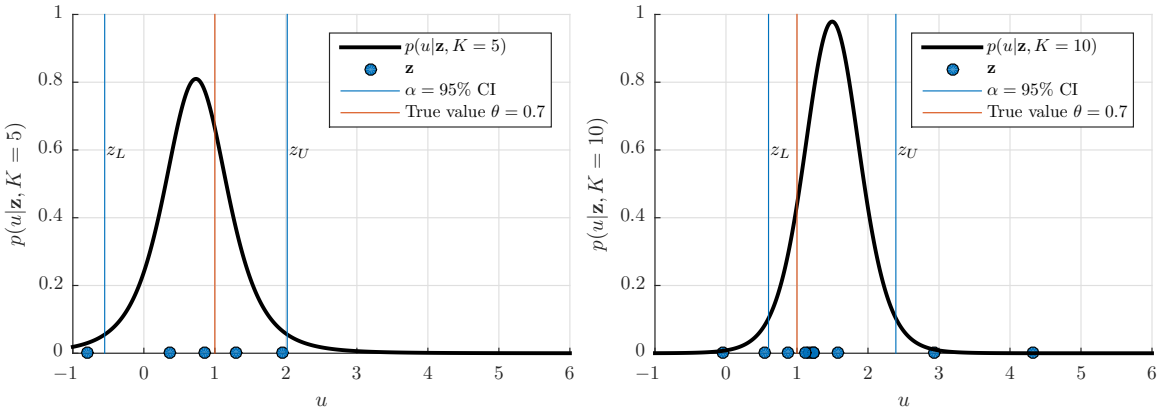


Fig. 9.16: Credibility intervals (CI) for the model comparison example. The circles indicate the observed values of z_k (the numbers are given in table 9.4 and the left pane correspond to the example in table 9.3) and the blue lines the credibility interval. The red line indicate the "true" (unknown) value of $E_A^{\text{gen}} - E_B^{\text{gen}}$. In the left-hand side the credibility interval includes 0 (se we should not believe the models performed differently) whereas in the second pane there is reason to prefer model B over A.

$K = 5$ the credibility interval contains 0 (thus we can't conclude the models might not have the same generalization error at $\alpha = 0.05$) whereas for $K = 10$ there is evidence that model $\mathcal{M}_A$ has a higher generalization error than $\mathcal{M}_B$.

9.3.4 Comments*

It is important to keep in mind the credibility interval, as all other statistics, depend on certain assumptions. For instance in the first example we assumed that each observation was classified correctly with the same probability and in the second that the z_k's were normally distributed. If these assumptions are violated, the conclusions are not trustworthy.

A person familiar with statistics might wonder why we did not use the standard confidence intervals taught in an introductory statistics class. Confidence intervals are perfectly well suited for solving these problems, and a reader who prefers using confidence intervals should by no means feel we are trying to discourage their use or wish to argue they are "bad" or untrustworthy. There are however two reasons we wish to advance in favor of credibility intervals: The first is that it is easy to understand what the credibility interval measure. We derive, based on modelling assumptions, a model of the parameter of interest $p(\theta|N, m)$ or $p(u|\boldsymbol{z})$ which quantifies what value we can expect θ or u to have. The credibility interval is then the interval that contains θ or u with probability $1-\alpha$ (the interval is symmetric in the sense that theta falls either below or above this interval with probability $\alpha/2$). On the other hand, what is the confidence interval?

The second reason is that *it makes very little difference*. The credibility interval in the first case is nearly identical to the corresponding $\alpha = 0.05$ confidence interval known as the Wilson interval, and the Wilson interval is only one of many, different, confidence intervals to choose from. Arguing which interval to use will at the end of the day be about moving the boundaries θ_L and θ_U slightly.

In the second case the credibility interval and the confidence interval is simply the same and we can therefore only argue about the derivation.

Problems

9.1. Fall 2014 question 27: 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 neglible, 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.

9.2. Spring 2013 question 13: We would like to fit an artificial neural network to the PM10 dataset shown in Table 9.5. 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 9.6 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 9.5: 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 rmse	Test 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 9.6: 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.

9.3. Spring 2013 question 14: 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.

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 9.8: 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.

9.4. Fall 2013 question 7: We would like to fit an artificial neural network (ANN) model to the Galagapos dataset shown in Table 9.7. 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 9.8 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 9.7: The seven attributes of the data on a selection of 29 of the Galápagos islands.

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.

9.5. Fall 2013 question 16: A logistic regression model was trained on Haberman's data shown in Table 9.9 using leave-one-out cross-validation and the confusion matrices given in Figure 9.17 were obtained. Which one of the following statements is *correct*?

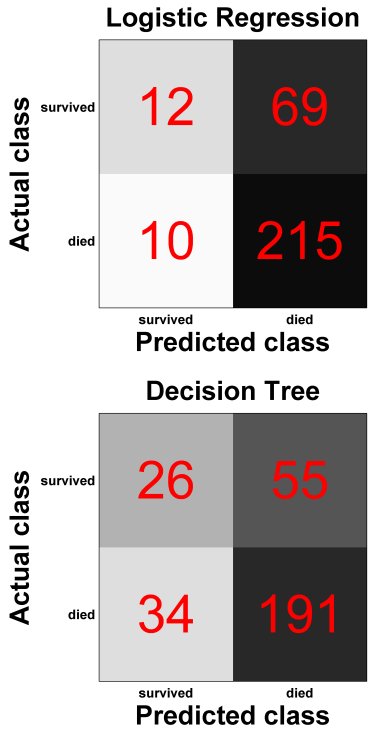


Fig. 9.17: The confusion matrix for the logistic regression and decision tree classifiers used to predict survival based on leave-one-out cross validation.

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 No, $x_3 = 0$ or Yes, $x_3 = 1$	PAN
y	Lived after 5 years No, $y = 0$ or Yes, $y = 1$	Surv

Table 9.9: 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.

9.6. Fall 2014 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 9.10.

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 9.10: 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.

9.7. Fall 2013 question 25: Which one of the following statements pertaining to cross-validation is *correct*?

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.

9.8. Fall 2014 question 18: 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 9.11 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 9.11: 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.

10

Nearest neighbour 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].

10.1 K-nearest neighbour classification

We will introduce the K-nearest neighbour classifier with an example. In fig. 10.1 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 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. 10.1 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. 10.2 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. 10.2 where these correspond to a yellow and two green points. In general, we can define the K-*neighbourhood* of a point $\boldsymbol{x}$ as:

$$N(\boldsymbol{x}, K) = \text{The } K \text{ observations which are nearest to } \boldsymbol{x}. \tag{10.1}$$

The lower-row of fig. 10.2 shows $N(\boldsymbol{x}, K = 5)$ and $N(\boldsymbol{x}, K = 7)$ respectively. A simple classification method is then to fix K (for instance $K = 5$) and just assign $\boldsymbol{x}$ to the class which has the most elements in $N(\boldsymbol{x}, K)$. In the case where two classes has equally many members (we say they are tied), for instance the lower-right pane of fig. 10.2, $\boldsymbol{x}$ is assigned to the class which has a *closest* member to $\boldsymbol{x}$ in $N(\boldsymbol{x}, K)$, in this case the green class. This is simply the KNN classification rule:

- Compute $N(\boldsymbol{x}, K)$.

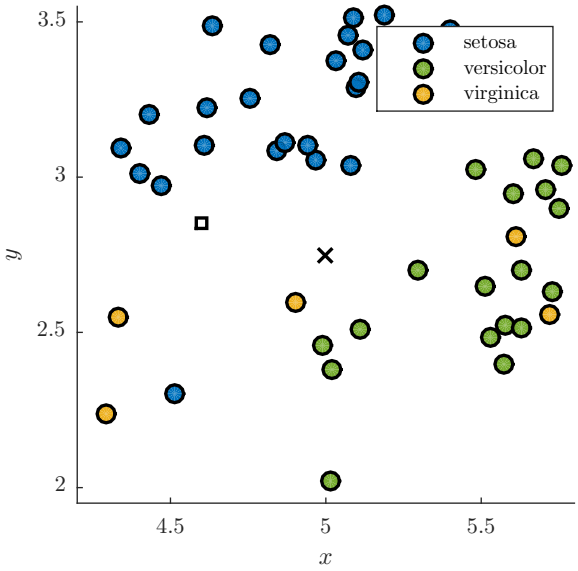


Fig. 10.1: A subset of the Fisher Iris dataset where we only consider two features. Which class and why would you assign the test points at the black square and cross if you were to just make a qualified guess?

- Classify $\boldsymbol{x}$ to the class k which has the most members in $N(\boldsymbol{x}, K)$.
- In the case of ties, simply classify $\boldsymbol{x}$ to the class which has a member nearest to $\boldsymbol{x}$.

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 $\boldsymbol{x}$ to the class of the *nearest* observation in the training set. This is known as the *nearest neighbour* classification rule. In fig. 10.3 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 $K = 1, 3, 5, 7$.

10.1.1 *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 $N(\boldsymbol{x}, K)$ which belong to class c and N_c the number of observations in the *entire* dataset which belongs to class c:

$$K_c = \text{Number of observations } \boldsymbol{x}_i \in N(\boldsymbol{x}, K) \text{ where } y_i = c. \tag{10.2}$$

$$N_c = \text{Number of observations } \boldsymbol{x}_i \text{ where } y_i = c. \tag{10.3}$$

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:

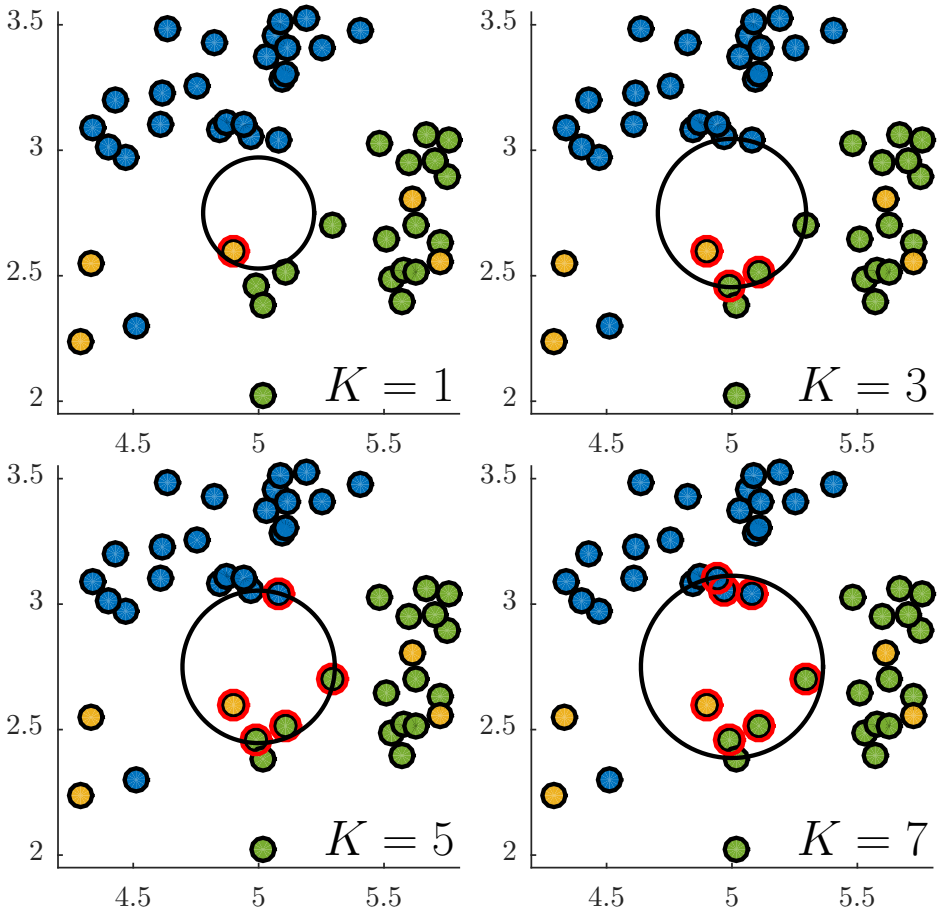


Fig. 10.2: Illustration of the K-nearest neighbourhood $N(\boldsymbol{x}, K)$ of the black cross $\boldsymbol{x}$ for $K = 1, 3, 5, 7$, observations within the neighbourhood is highlighted with the red circles. The KNN classifier simply assigns the observation $\boldsymbol{x}$ to the class with the most observations within the circle.

$$\int_V p(\boldsymbol{x}|y = c)d\boldsymbol{x} = \{\text{Probability an observation of class } c \text{ is in } V\}$$

If we now consider the volume V to be the size of the K-nearest neighbourhood of $\boldsymbol{x}$, i.e. the area of the discs in fig. 10.2) around $\boldsymbol{x}$, the left-hand side and right-hand side of the above becomes:

$$\begin{aligned} \{\text{lhs.}\} &= \int_V p(\boldsymbol{x}|y = c)d\boldsymbol{x} && \approx Vp(\boldsymbol{x}|y = c) \\ \{\text{rhs.}\} &= \frac{\text{Number of observations of class } y = c \text{ in } V_{\boldsymbol{x}}}{\text{Total number of observations of class } c} && \approx \frac{K_c}{N_c} \end{aligned}$$

If we put this together we obtain $p(\boldsymbol{x}|y = c) = \frac{K_c}{N_c V}$. Then simply applying Bayes theorem we obtain:

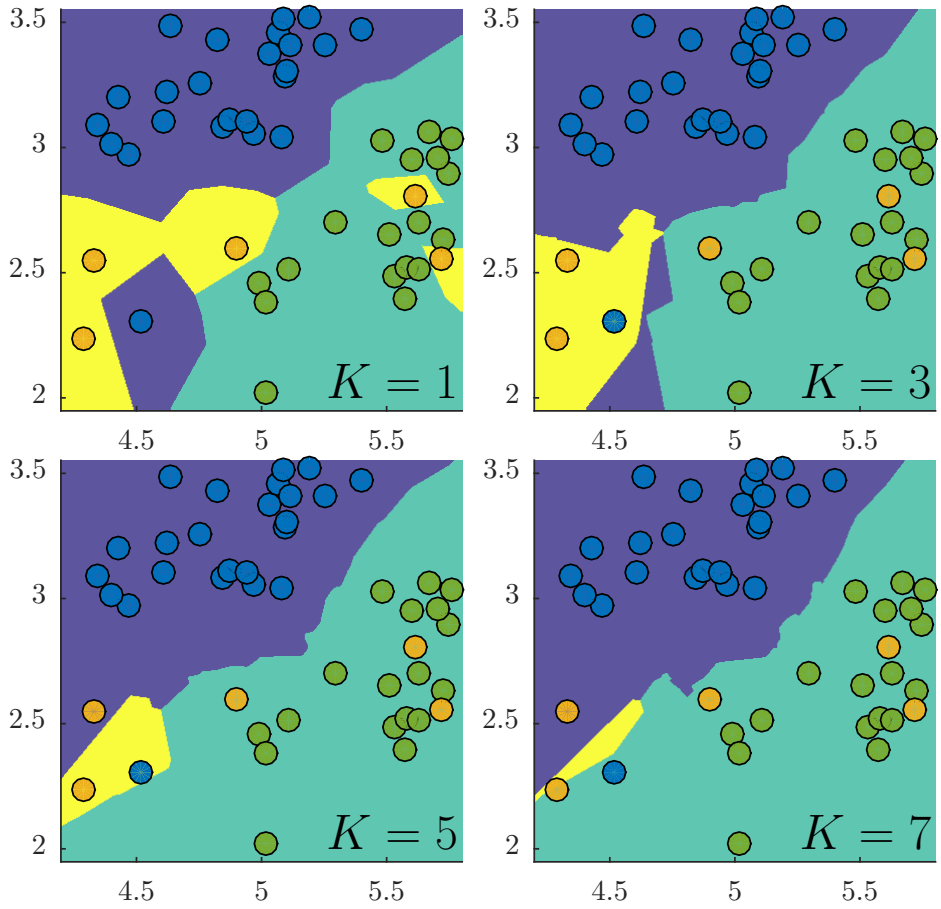


Fig. 10.3: KNN classification boundary for the problem in fig. 10.1 for $K = 1, 3, 5, 7$. Notice as K increases, the boundary becomes more smooth. For $K = 3$ (upper-right corner), the blue point in the lower left corner is able to induce a small blue area due to the tie-breaking rule.

$$p(y = c|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|y = c)p(y = c)}{\sum_{c'=1}^{C} p(\boldsymbol{x}|y = c')p(y = c')} = \frac{K_c}{K} \tag{10.4}$$

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.

10.2 K-nearest neighbour regression

The K-nearest neighbour classification rule is easily modified for regression. Suppose we have the dataset shown in fig. 10.4 which (top left pane) consisting 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, $N(x, K)$, (shown as the circles)

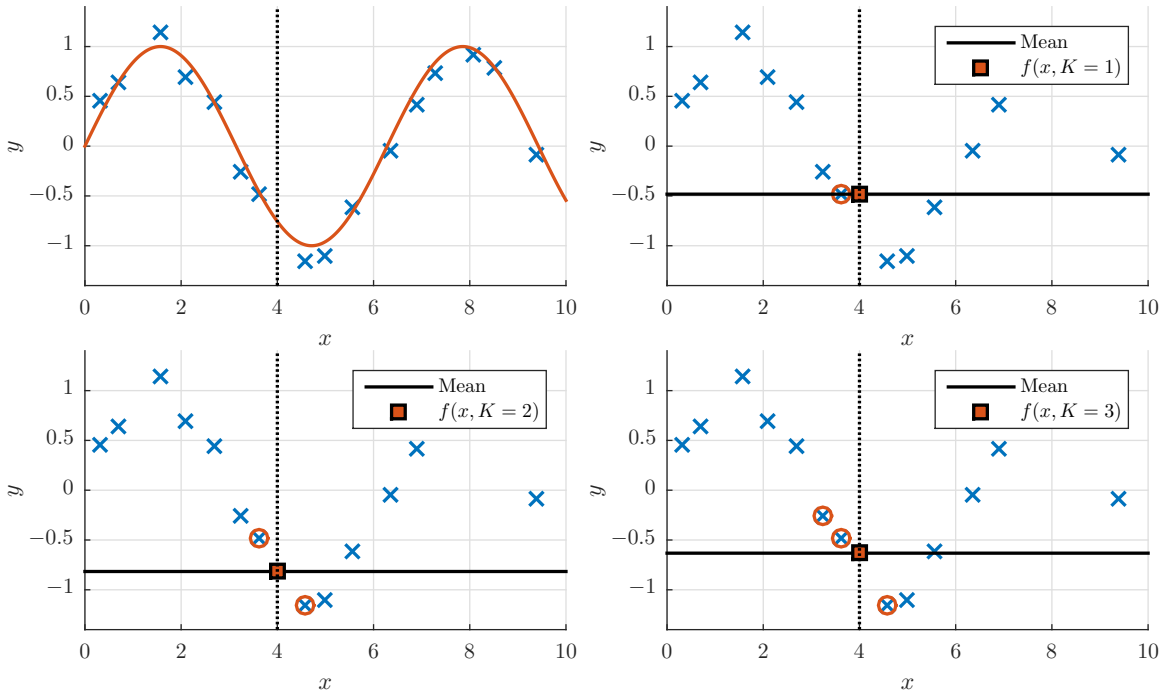


Fig. 10.4: 1D example dataset (upper-right pane) generated as noisy observations of the sinusoidal signal. KNN regression for an observation x indicated by the vertical dotted line first finds the K-nearest neighbourhood of x, $N(x, K)$, and then simply outputs the mean of the observations y_i in $N(x, K)$. The three panes illustrates $K = 1, 2, 3$.

and simply predicting the mean value of the elements in $N(x, K)$ (shown as the horizontal bar). The prediction at $x = 4$ is then just the red square. In general the prediction rule is:

$$f(\boldsymbol{x}, K) = \frac{1}{K} \sum_{i \in N(\boldsymbol{x}, K)} y_i. \tag{10.5}$$

which of course works for arbitrary dimensions. Notice in particular the $K = 1$ prediction rule simply corresponds to finding the observation $\boldsymbol{x}_i$ closest to a test point $\boldsymbol{x}$ and predicting $f(\boldsymbol{x}, K = 1) = y_i$. In fig. 10.5 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.

10.2.1 *Higher-order KNN regression

If we return to the KNN regression dataset in fig. 10.4 and the prediction curves in fig. 10.5, notice 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

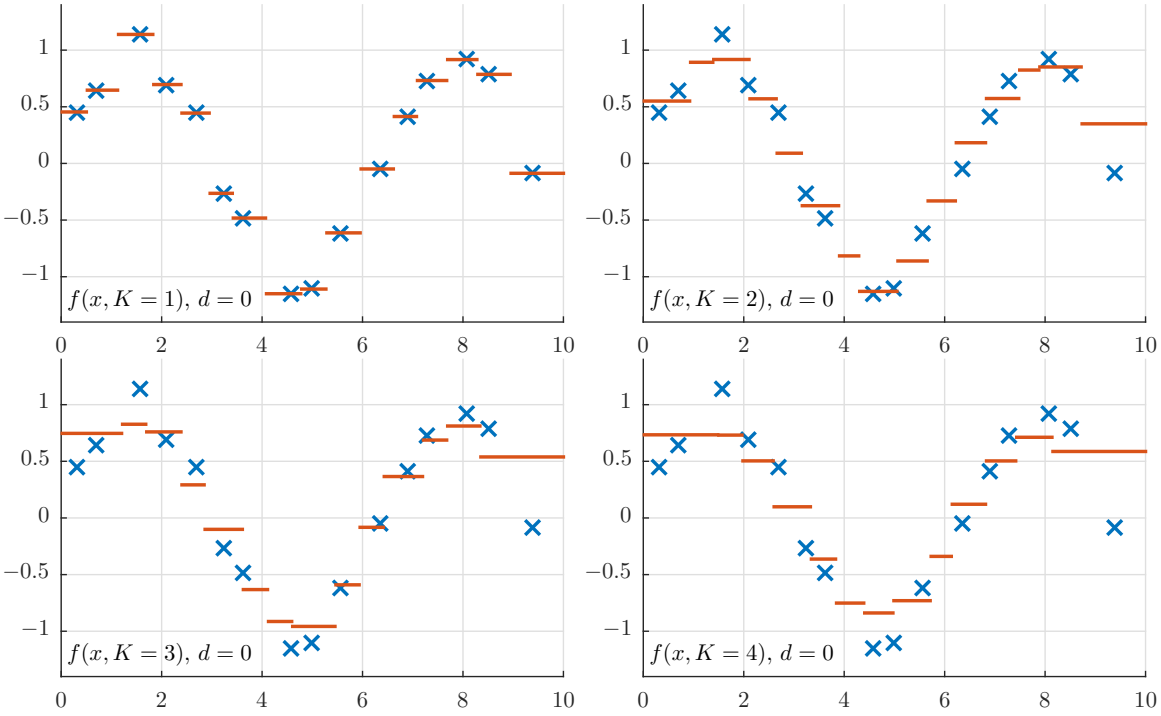


Fig. 10.5: Illustration of the prediction rule for KNN regression from fig. 10.4 (red line) when computed over the entire dataset for $K = 1, 2, 3, 4$. Notice the prediction rule is piece-wise linear corresponding to different neighbourhoods.

model then corresponds to $d = 0$ (the constant polynomial). This is illustrated in fig. 10.6 for $K = 3, 5$ and $d = 1, 2$.

The corresponding prediction curve is shown in fig. 10.7. Compared to the piece-wise linear case in fig. 10.5, 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.

10.3 Cross-validation and nearest-neighbour methods

Selecting K in nearest neighbour methods can easily be accomplished using cross-validation. Simply let $\mathcal{M}_1, \cdots, \mathcal{M}_S$ in algorithm 4 correspond to different values of K (for instance $K = 1, \cdots, S$) and let for instance the error measure be (in case of classification) the classification error, or for regression the sum of square error, then one-layer cross-validation for parameter selection can be used to select K and two-layer cross-validation used for selecting K and estimating the generalization error. A particulary 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 selecting the $K + 1$ neighbourhood $N(\boldsymbol{x}_i, K + 1)$ and leaving out observation $\boldsymbol{x}_i$ and using the remaining K observations in $N(\boldsymbol{x}_i, K + 1)$ to train a classifier (or regression

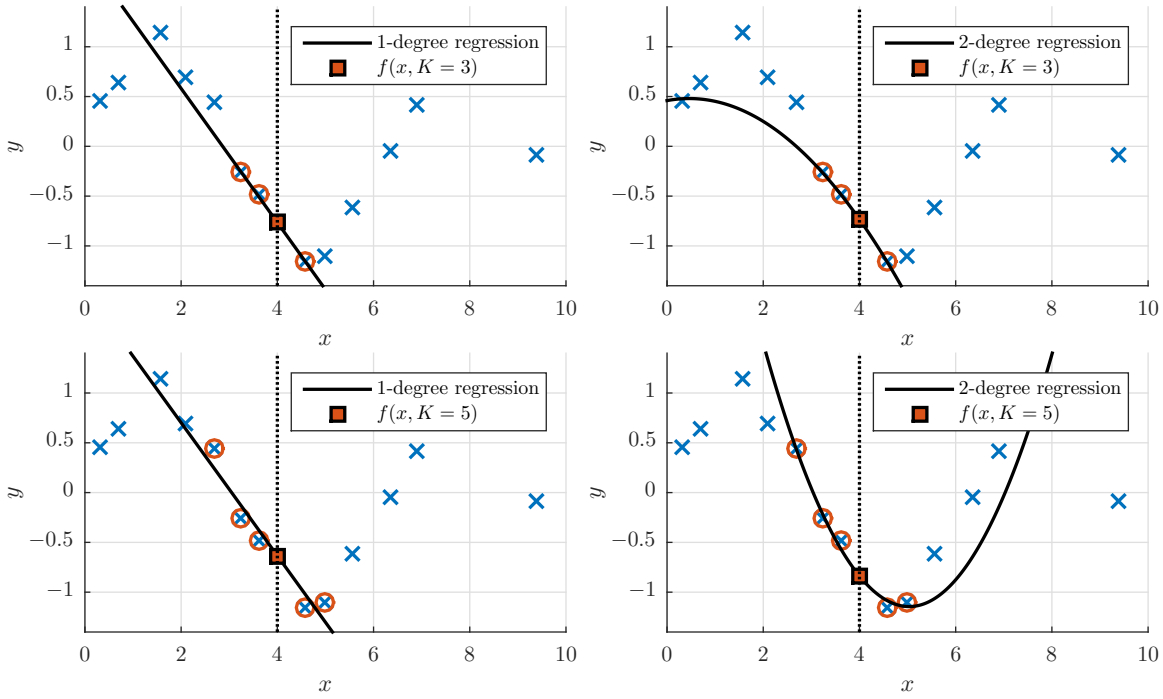


Fig. 10.6: KNN regression can be extended by instead of computing the mean within each region, we carry out a polynomial regression (similar to the example encountered in the previous section on linear regression) for the observations in each neighbourhood $N(\boldsymbol{x}, K)$. For different degree $d = 2, 3$ this allows smoother regression curves, but higher d also requires a larger neighbourhood.

method) and predict $\boldsymbol{x}_i$. In general when using cross-validation to estimate the optimal value of K for nearest-neighbour methods this can be accomplished very efficiently as the neighborhood of each observations only need to be estimated once for the largest value of K in order to have identified the neighborhood also of smaller values of K.

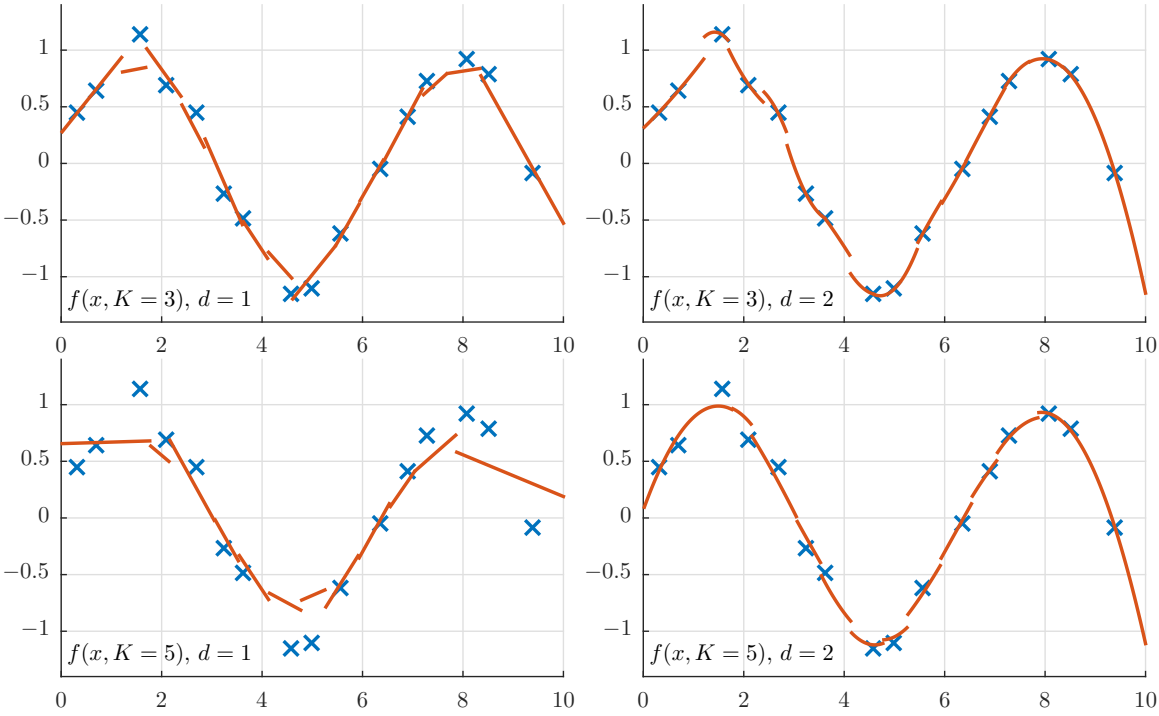


Fig. 10.7: The prediction curves for KNN regression with polynomials introduced in fig. 10.6, here shown for linear polynomials and second-degree polynomials for different neighborhood sizes.

Problems

10.1. Fall 2013 question 9: 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 10.1. 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 10.1. 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 10.1: Pairwise Euclidean distance, i.e $d(Oa, Ob) = ||\boldsymbol{x}_a - \boldsymbol{x}_b||_2 = \sqrt{\sum_m (x_{am} - x_{bm})^2}$, 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.

10.2. Fall 2014 question 23: Consider a two-dimensional data set consisting of $N = 7$ observations as shown in fig. 10.8. 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?

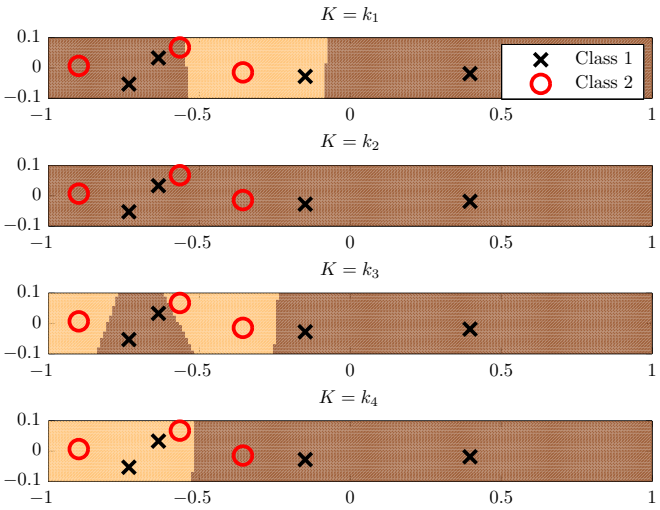


Fig. 10.8: Decision boundaries for four KNN classifiers.

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.

10.3. Fall 2014 question 12: Consider again the distances in table 10.2. We will predict the label indicated by the blue color, i.e. observations $o_1, \ldots, o_4$ belong to class C_1 and observations $o_5, \ldots, o_8$ to class C_2. This will be done using a k-nearest neighbor (KNN) classifier based on the cityblock distance measure indicated in table 10.2. 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 10.2: Pairwise Cityblock distance, i.e $d(o_i, o_i) = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_1 = \sum_{k=1}^{M} |x_{ik} - x_{jk}|$, 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.

11

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 chapter 5 and as an element of the credibility intervals in chapter 9. Hwever, in this chapter we will consider the problem more heads on and discuss additional terminology particular to Bayesian methods, namely Bayesian 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.

11.1 Discriminative and generative modelling

Consider a standard classification problem in which we try to determine what class y_i an observation $\boldsymbol{x}_i$ belongs to. For instance, suppose we try to learn to distinguish between cats ($y_i = 0$) and dogs ($y_i = 1$) based on features $\boldsymbol{x}_i$ of each animal. Logistic regression essentially tries to fit a straight line – a decision boundary– that separate the cats from the dogs. A new instance $\boldsymbol{x}_i$ 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

$$p(y|\boldsymbol{x}, \boldsymbol{w}). \tag{11.1}$$

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 dogs and cats look like is known as *generative* modelling and Bayesian inference naturally corresponds to generative modelling.

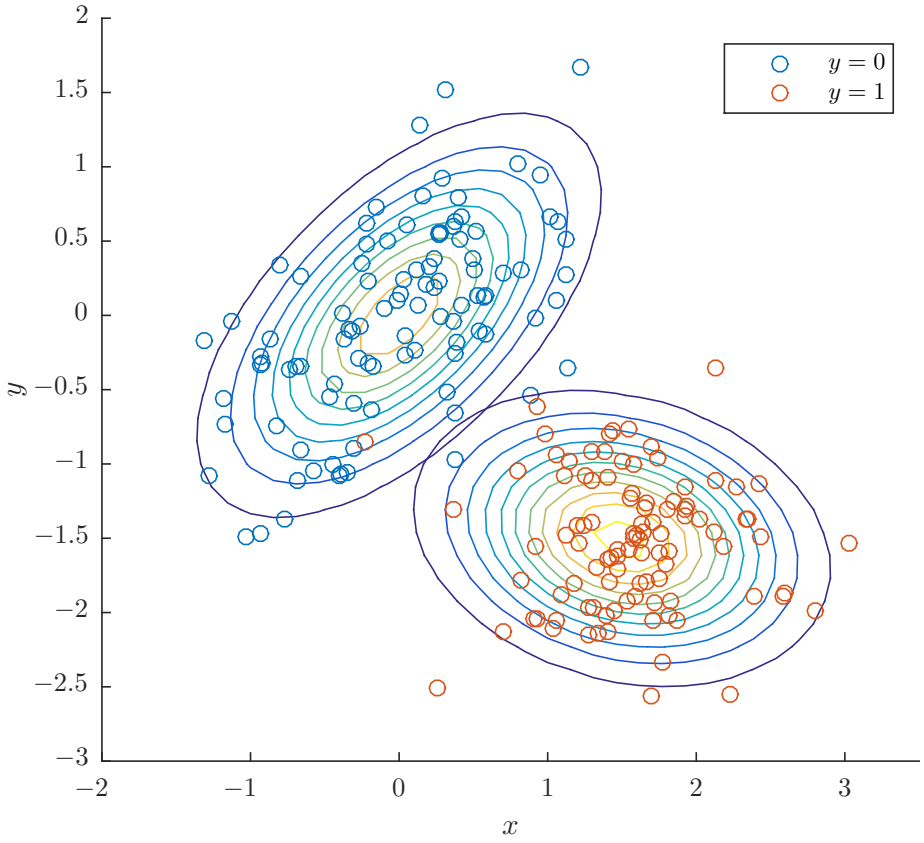


Fig. 11.1: Example of a Bayes classifier fitted to a two-class example dataset. The contours indicate the multivariate Gaussians fitted to each of the two classes.

11.1.1 Bayes classifier

Continuing the above discussion, consider Bayes' theorem in the two-class setting:

$$p(y|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|y)p(y)}{p(\boldsymbol{x}|y=0)p(y=0) + p(\boldsymbol{x}|y=1)p(y=1)} \tag{11.2}$$

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

$$p(\boldsymbol{x}|y=0),$$

multiply by the prior probability the animal is a cat $p(y = 0)$ and divide this by the same quantities but for dogs, $p(\boldsymbol{x}|y = 1)$ and $p(y = 1)$. To make this more concrete, lets suppose we observe n_0 instances of cats, $\boldsymbol{X}^{\text{Cats}}$, and n_1 instances of dogs, $\boldsymbol{X}^{\text{Dogs}}$, each observation consisting of two features. The labelled dataset is plotted in fig. 11.1. Then we can model the observations for instance as two multivariate normal distributions

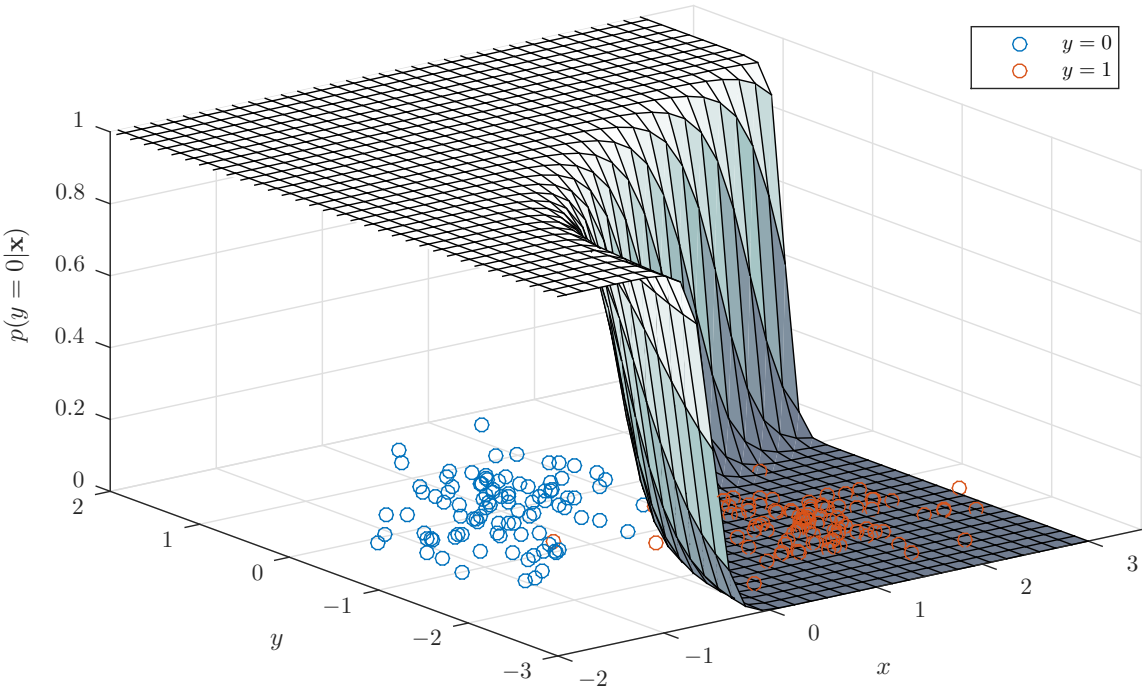


Fig. 11.2: Decision rule, i.e. the probability $p(y = 0|\boldsymbol{x})$, of the Bayes classifier when trained on the two-class dataset from fig. 11.1. Notice, the decision rule is quite steep at the boundary.

$$p(\boldsymbol{x}|y = 0) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0), \tag{11.3}$$
$$p(\boldsymbol{x}|y = 1) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1), \tag{11.4}$$

where the parameters $\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0$ and $\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1$ can be estimated form the data as:

$$\boldsymbol{\mu}_0 = \frac{1}{n_0}\sum_{i=1}^{n_0} \boldsymbol{x}_i^{\text{Cats}}, \quad \text{and} \quad \boldsymbol{\Sigma}_0 = \frac{1}{n_0 - 1}\sum_{i=1}^{n_0} (\boldsymbol{x}_i^{\text{Cats}} - \boldsymbol{\mu}_0)(\boldsymbol{x}_i^{\text{Cats}} - \boldsymbol{\mu}_0)^T, \tag{11.5}$$

corresponding to the two contour plots in fig. 11.1. 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:

$$p(y = c|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|y = c)p(y = c)}{p(\boldsymbol{x}|y = 0)p(y = 0) + p(\boldsymbol{x}|y = 1)p(y = 1)}. \tag{11.6}$$

The decision boundary, i.e. $p(y = 0|\boldsymbol{x})$, is plotted in fig. 11.2 as the gray surface.

Bayes' theorem in general*

The above story can be extended in a number of ways. The basic rule we follow is to write up a posterior distribution of all variables of interest and then use the rules of probability to derive a

probability expression for the quantity of interest, in the above case $p(\boldsymbol{y}|\boldsymbol{X})$. This program is conceptually simply, however, the integrals involved are in many cases quite involved, and it is therefore common to apply approximations techniques such as Variational Bayes or Expectation Propagation or exact methods such as Markov-Chain Monte-Carlo sampling which are topics covered in advanced machine learning courses.

11.2 Naïve-Bayes classifier

Naïve-Bayes is simply the standard Bayesian approach with a particular simplification. Consider the Bayes classifier for C classes:

$$p(y|x_1, x_2, \ldots, x_M) = \frac{p(x_1, x_2, \ldots, x_M|y)p(y)}{\sum_{c=1}^{C} p(x_1, \ldots, x_M|y=c)p(y=c)} \tag{11.7}$$

A problem with the Bayes classifier is if M is very large, representing the conditional distribution

$$p(x_1, x_2, \ldots, x_M|y),$$

may be very expensive. For instance, for the multivariate normal distribution this requires storing a symmetric covariance matrix $\boldsymbol{\Sigma}$ and the mean vector $\boldsymbol{\mu}$, 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 numbers may not be possible to do reliably. The Naïve-Bayes assumption is simply that we *assume* that the conditional distribution factorizes:

$$p(x_1, x_2, \ldots, x_M|y) = p(x_1|y)p(x_2|y)\ldots p(x_M|y).$$

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 $x_1, x_2, \ldots, x_m$). Plugging this into eq. (11.7) we obtain:

$$p(y|x_1, x_2, \ldots, x_M) = \frac{p(x_1|y) \times \cdots \times p(x_M|y)p(y)}{\sum_{c=1}^{C} p(x_1|y=c)p(y=c) \times \cdots \times p(x_M|y=c)p(y=c)}. \tag{11.8}$$

We will illustrate the procedure with an example. Consider the data shown in table 11.1. 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 $y = 1, 2, 3$ correspond 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 $p(x_i = 0|y = c)$ as:

$$p(x_i = 0|y = c) = \frac{\text{Number of times where } x_i = 0 \text{ and } y = c}{\text{Total number of times where } y = c}. \tag{11.9}$$

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 11.1: 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.

In particular, we obtain:

$$p(x_1 = 0|y = 1) = \frac{1}{3}, \qquad p(x_1 = 0|y = 2) = \frac{1}{3}, \qquad p(x_1 = 0|y = 3) = \frac{1}{2},$$
$$p(x_2 = 0|y = 1) = \frac{1}{3}, \qquad p(x_2 = 0|y = 2) = \frac{2}{3}, \qquad p(x_2 = 0|y = 3) = \frac{0}{2}.$$

Using that $p(x_2 = 1|y = c) = 1 - p(x_2 = 0|y)$ we then compute the probability of the new observation as

$$p(x_1 = 0, x_2 = 1|y = 1) = p(x_1 = 0|y = 1)p(x_2 = 1|y = 1) = \frac{1}{3} \times (1 - \frac{1}{3}) = \frac{2}{9},$$
$$p(x_1 = 0, x_2 = 1|y = 2) = p(x_1 = 0|y = 2)p(x_2 = 1|y = 2) = \frac{1}{3} \times (1 - \frac{2}{3}) = \frac{1}{9},$$
$$p(x_1 = 0, x_2 = 1|y = 3) = p(x_1 = 0|y = 3)p(x_2 = 1|y = 3) = \frac{1}{2} \times (1 - 0) = \frac{1}{2}.$$

In our case $p(y{=}c|x_1{=}0, x_2{=}1)$ can then be computed using eq. (11.8) 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

$$p(y = 1|x_1 = 0, x_2 = 1) = \frac{\frac{2}{9} \times \frac{3}{8}}{\frac{2}{9} \times \frac{3}{8} + \frac{1}{9} \times \frac{3}{8} + \frac{1}{2} \times \frac{1}{4}} = \frac{1}{3},$$
$$p(y = 2|x_1 = 0, x_2 = 1) = \frac{\frac{1}{9} \times \frac{3}{8}}{\frac{2}{9} \times \frac{3}{8} + \frac{1}{9} \times \frac{3}{8} + \frac{1}{2} \times \frac{1}{4}} = \frac{1}{6},$$
$$p(y = 3|x_1 = 0, x_2 = 1) = \frac{\frac{1}{2} \times \frac{1}{4}}{\frac{2}{9} \times \frac{3}{8} + \frac{1}{9} \times \frac{3}{8} + \frac{1}{2} \times \frac{1}{4}} = \frac{1}{2}.$$

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.

11.2.1 Robust estimation

Consider again the previous example where Naïve-Bayes was applied to the market-basket example table 11.1. As part of the computation we made approximations of the form:

$$p(x=1) = \frac{n^+}{n^+ + n^-}$$

where n^+ was the number of times we observed $x=1$ and n^- the number of times we observed $x=0$. A drawback of this way of estimating the probability $p(x=1)$ is that we might get probabilities of 0 which will dominate all other probabilities in the computation when multiplied together. A numerically safer way to get around this problem is to use the M-esimator:

$$p(x=1) = \frac{n^+ + b^+}{n^+ + n^- + b^+ + b^-}$$

where b^+ and b^- are so-called "pseudo-counts" (consider for instance if $b^+ = b^- = 1$ then using the robust estimator is simply equivalent to assuming we had "pseudo-observed" a member of the negative and positive class). It is common to select $b^- = b^+ = \frac{1}{2}$ or $b^- = b^+ = 1$, however in general the value of b^+, b^- could be found by cross-validation.

Where does the robust estimator come from?*

Suppose we define θ as the chance of observing a positive outcome of x. Then the above information is exactly equivalent to the Bernoulli-coin example we encountered previously in eq. (9.10) and in particular eq. (9.11). If we instead of the Beta($\frac{1}{2}, \frac{1}{2}$) prior had chosen a general Beta(b^+, b^-) prior the posterior eq. (9.11) would be:

$$p(\theta|n^+, n^-, b^+, b^-) = \text{Beta}(\theta|n^+ + b^+, n^- + b^-)$$

But we are only interested in the probability of observing a positive outcome $P(x=1)$. This can be obtained as the average of θ:[1]

$$P(x=1) = \int p(x=1|\theta)p(\theta|n^+, n^-, b^+, b^-)d\theta = \int \theta \,\text{Beta}(\theta|n^+ + b^+, n^- + b^-)d\theta = \frac{n^+ + b^+}{n^+ + b^+ + n^- + b^-}.$$

Thus there is nothing "special" about this robust estimator; it is simply about a Bayesian analysis of the chance of observing a positive outcome of x using Beta(b^+, b^-) prior.

11.3 Bayesian networks

A Bayesian 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]

[1] If θ follows a beta distribution then the average of θ is: $\mathbb{E}[\theta] = \int \theta \text{Beta}(\theta|a, b)d\theta = \frac{a}{a+b}$.

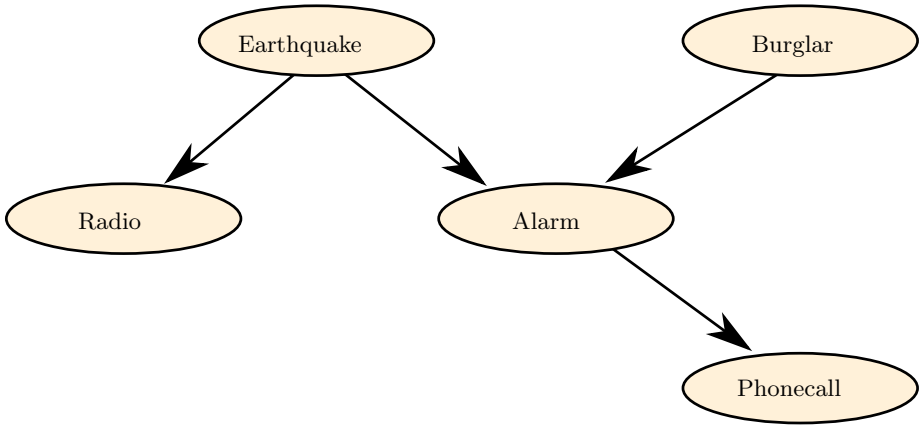


Fig. 11.3: Bayesian network of the burglar example. Each vertex corresponds to a variable, and incident edges corresponds to conditional dependence.

> *Fred lives in Los Angeles and commutes 60 miles to work. Whilst at work, he receives a phone-call from his neighbour saying that Freds 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 presumable unrelated events, so $p(b, e) = p(b)p(e)$. In general, the probability of these variables will factorize as follows:

$$p(a, b, c, e, r) = p(b)p(e)p(a|b, e)p(c|a)p(r|e) \tag{11.10}$$

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. 11.3 for an illustration. So for instance, if there is an edge from A to B, 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:[2]

$$\begin{aligned} p(a = 1|b = 0, e = 0) &= 0.1\% \\ p(a = 1|b = 0, e = 1) &= 1.099\% \\ p(a = 1|b = 1, e = 0) &= 99.001\% \\ p(a = 1|b = 1, e = 1) &= 99.011\% \end{aligned}$$

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:

$$p(a = 1) = \sum_{b\in\{0,1\}} \sum_{c\in\{0,1\}} \sum_{e\in\{0,1\}} \sum_{r\in\{0,1\}} p(a = 1, b, c, e, r), \tag{11.11}$$

However, if we plug in the expression of the likelihood (11.10) we see that variables c and r can trivially be summed out (i.e., marginalized):

$$\begin{aligned} p(a = 1) &= \sum_{b\in\{0,1\}} \sum_{c\in\{0,1\}} \sum_{e\in\{0,1\}} \sum_{r\in\{0,1\}} p(b)p(e)p(a = 1|b, e)p(c|a = 1)p(r|e) \\ &= \sum_{b\in\{0,1\}} \sum_{e\in\{0,1\}} \left[p(b)p(e)p(a = 1|b, e) \left(\sum_{c\in\{0,1\}} p(c|a = 1) \sum_{r\in\{0,1\}} p(r|e) \right) \right] \\ &= \sum_{b\in\{0,1\}} \sum_{e\in\{0,1\}} p(b)p(e)p(a = 1|b, e) \end{aligned} \tag{11.12}$$

[2] 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:

$$\begin{aligned} p(a = 1|b = 0, e = 0) &= f \\ p(a = 1|b = 0, e = 1) &= 1-(1-f)(1-\alpha_e) \\ p(a = 1|b = 1, e = 0) &= 1-(1-f)(1-\alpha_b) \\ p(a = 1|b = 1, e = 1) &= 1-(1-f)(1-\alpha_b)(1-\alpha_e) \end{aligned}$$

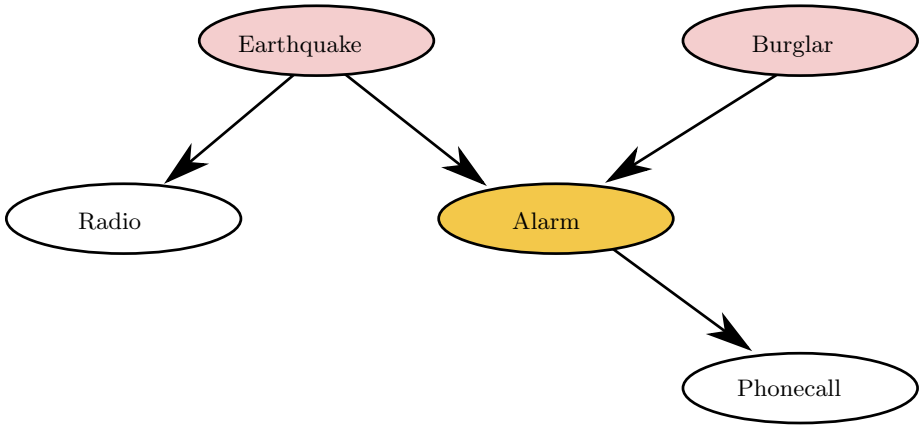


Fig. 11.4: To determine what variables must be summed out when computing the marginal of a variable such as a, we look at all variables such that one can move in the direction of the arrows from those variables to a. This gives $p(a, e, b)$

Comparing to the Bayesian network in fig. 11.3 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. 11.4 where we have illustrated the two nodes that remain, e, b, with red. Using the above numbers we obtain:

$$
\begin{aligned}
&p(a = 1|b = 0, e = 0)p(b = 0)p(e = 0) = 0.00998 \\
&p(a = 1|b = 1, e = 0)p(b = 1)p(e = 0) = 0.00989 \\
&p(a = 1|b = 0, e = 1)p(b = 0)p(e = 1) = 0.000010979 \\
&p(a = 1|b = 1, e = 1)p(b = 1)p(e = 1) = 9.9 \times 10^{-7}
\end{aligned}
$$

By inserting these four numbers into eq. (11.12) and summing we obtain $p(a = 1) = 0.002$ and so from eq. (11.11)

$$
\begin{aligned}
p(b = 0, e = 0|a = 1) &= 0.4993 && (11.13)\\
p(b = 1, e = 0|a = 1) &= 0.4947 && (11.14)\\
p(b = 0, e = 1|a = 1) &= 0.0055 && (11.15)\\
p(b = 1, e = 1|a = 1) &= 0.0005. && (11.16)
\end{aligned}
$$

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:

$$
\begin{aligned}
p(b = 0|a = 1) &= p(b = 0, e = 0|a = 1) + p(b = 0, e = 1|a = 1) && = 0.505\\
p(b = 1|a = 1) &= p(b = 1, e = 0|a = 1) + p(1 = 0, e = 1|a = 1) && = 0.494
\end{aligned}
$$

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

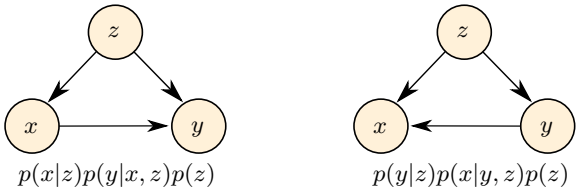


Fig. 11.5: Two bayesian networks which both represent the same distribution $p(x, y, z)$. Since the two networks are not similar this shows a Bayesian network cannot be interpreted as a causal graph.

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 out probability there was a burglar in the house from about 50% to about 8%. This is in according to every day intuition: when we learn about the earthquake, we consider *that* to be the more plausible explanation of the alarm.

11.3.1 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 $p(y|x, \cdots)$ in 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. 11.5, this shows we cannot interpret a Bayesian network as a causal graph.

Problems

11.1. Fall 2015 question 16: Nine of the fifteen observations in Table 11.2 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 11.2 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 11.2: For each observation there are $M = 7$ binary features and $N = 15$ observations $O_1, \ldots, O_{15}$ belonging to two categories (i.e., CKD=1 for $O_1, \ldots, O_9$ and CKD=0 for $O_{10}, \ldots, O_{15}$).

A 2.56 %
B 96.14 %
C 98.03 %
D 100 %
E Don't know.

11.2. Fall 2015 question 17: We will consider a Bayes classifier using the attributes RBC, PC, and DM in Table 11.2 (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.

11.3. Fall 2013 question 19: Five of the ten considered subjects in Table 11.3 survived after five years ($S1-S5$) given in black whereas five subjects died within five years ($NS1 - NS5$) given in red. We would like to predict whether a subject survived using the data in Table 11.3 and the attributes YA_Y, OA_Y, PA_Y. We will apply a Naïve Bayes classifier that assumes independence between the three attributes. Given that a subject had these three attributes (i.e., $YA_Y = 1$, $OA_Y = 1$, $PA_Y = 1$) what is the probability that the subject survived according to the Naïve Bayes classifier. I.e., what is $P(S|YA_Y = 1, OA_Y = 1, PA_Y = 1)$ according to the Naïve Bayes classifier?

	YA_Y	YA_N	OA_Y	OA_N	PA_Y	PA_N
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 11.3: 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 (YA_Y, YA_N), were operated after 1960 or not (OA_Y, OA_N), and had positive axillary nodes or not (PA_Y, PA_N).

A $\frac{16}{125}$
B $\frac{3}{11}$
C $\frac{1}{2}$
D $\frac{8}{11}$
E Don't know.

11.4. Fall 2014 question 15: Consider the observations in table 11.4. 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

Table 11.4: $N = 10$ observations $s_1, \ldots, s_{10}$ belonging to two categories. The black category C_1 (observations $s_1, \ldots, s_5$) and the blue category C_2 (observations $s_6, \ldots, s_{10}$). For each observation there are $M = 6$ binary features $f_1, \ldots, f_6$.

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.

12

Regularization and the bias-variance decomposition

As we already saw in chapter 9, "Overfitting and performance evaluation", a too flexible model can easily overfit the dataset leading to a low 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 particulary 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 tradeof 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].

12.1 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 parameter selection. We illustrate the technique using least-squares regression. Consider the simple linear regression model:

$$\boldsymbol{y}_i = f(\boldsymbol{x}_i, \boldsymbol{w}) = \boldsymbol{x}_i^T \boldsymbol{w},$$

with error term:

$$E(\boldsymbol{w}) = \frac{1}{2} \left\| \boldsymbol{y} - \boldsymbol{X}\boldsymbol{w} \right\|^2 . \tag{12.1}$$

The optimal weights $\boldsymbol{w}^*$ can be found by minimizing the error and are given by:

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} E(\boldsymbol{w}) = (\boldsymbol{X}^T\boldsymbol{X})\backslash(\boldsymbol{X}^T\boldsymbol{y}). \tag{12.2}$$

Suppose we simply change the error term to add a new factor of $\lambda \boldsymbol{w}^T\boldsymbol{w}$ where $\lambda \geq 0$ is a parameter

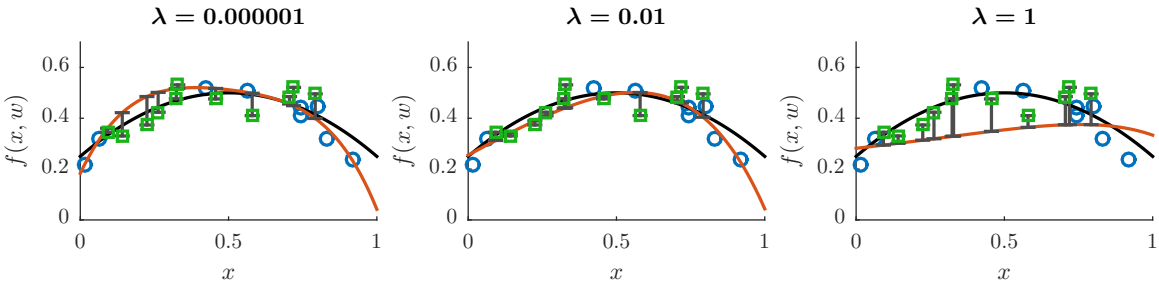


Fig. 12.1: A regularized linear regression model is fitted to the dataset of 9 observations and 10 test observations. The solutions, corresponding to three different values of λ, are shown in the three panes. Notice for larger values of λ, the solution is dragged towards the x-axis because the solution for the weights w^* becomes smaller according to eq. (12.3). The left-most pane has high variance but low bias, the right-most pane has high bias but low variance.

known as the *regularization strength*. Then eq. (12.1) becomes

$$E_\lambda(\boldsymbol{w}) = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|^2 + \lambda \boldsymbol{w}^T\boldsymbol{w}. \tag{12.3}$$

Solving for the optimal weights gives:

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} E(\boldsymbol{w}) = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})\backslash(\boldsymbol{X}^T\boldsymbol{y}). \tag{12.4}$$

Therefore $\boldsymbol{w}^*$ becomes dependent on the regularization strength λ. Notice an advantage to this expression is the matrix $\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}$ is always invertible thus we can also write this as: $\boldsymbol{w}^* = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}$. This means our solutions for w^* becomes unique even when we have fewer training observations N than features M.

Lets make some general comments: If we simply look at the expression for $E_\lambda(\boldsymbol{w})$ in eq. (12.3) then if $\lambda = 0$ we get ordinary linear regression. If on the other hand λ is large, the error term prefers each coordinate of $\boldsymbol{w}$, w_i, to be as small as possible. This is also evident from the expression for the solution eq. (12.4): If we for a moment naïvely consider $\boldsymbol{X}$ and $\boldsymbol{y}$ as being scalars we get:

$$w^* = \frac{Xy}{X^2 + \lambda},$$

so the larger λ is, the smaller $\boldsymbol{w}^*$ becomes and in the limit $\lambda \to \infty$ then $\boldsymbol{w}^* = 0$. In fig. 12.1 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 $\boldsymbol{w}^*$ 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 $\boldsymbol{w}_i^*$ for many values of λ is shown in fig. 12.2.

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. 12.3 the variable λ is tweaked from a very small value

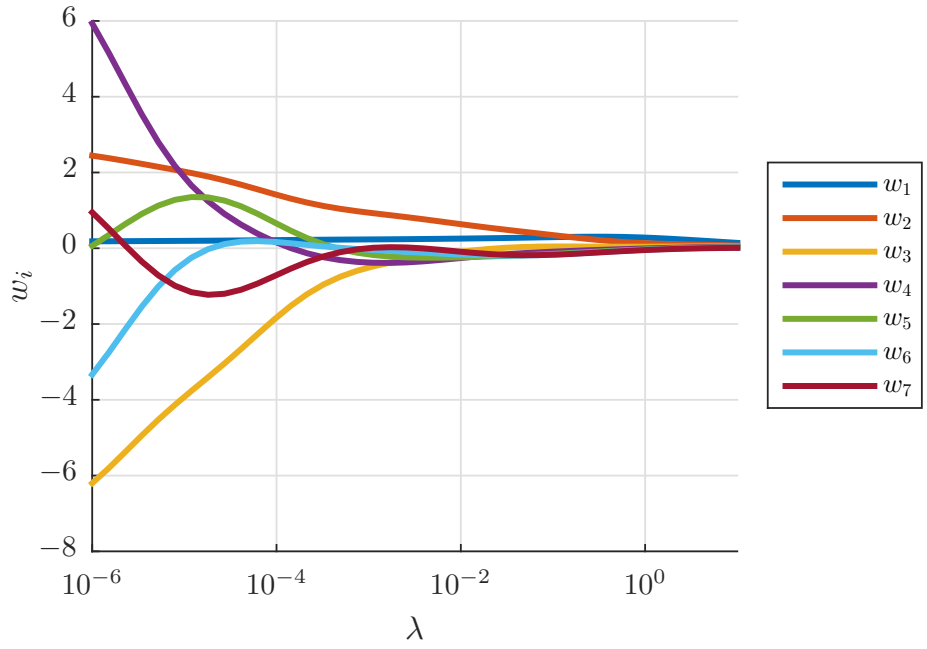


Fig. 12.2: A regularized linear regression model is fitted to the dataset shown in fig. 12.1 and the coordinates of the optimal weights are plotted. When λ is small, the weights are large indicating high variance but low bias. When λ is larger, the weights become smaller indicating lower variance but higher bias in the solutions.

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. 12.1 displayed. The three particular values plotted in fig. 12.1 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 $\lambda \approx 10^{-1}$. In practice when we search for the optimal value of λ, we test S different values of λ, $\lambda_1, \ldots, \lambda_S$ selected beforehand and then compare each of the corresponding linear regression models using cross-validation for parameter selection.

12.1.1 Comments on regularization*

Regularization, as explained here, is simply adding a factor $\lambda \boldsymbol{w}^T \boldsymbol{w}$ to our error which may appear rather arbitrary. It is however possible to give regularization a natural Bayesian interpretation.

Recall that in our original discussion of linear regression in chapter 7 we started by writing up the likelihood of the data given the model parameters:

$$p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w}) = \prod_{i=1}^{N} \mathcal{N}(y_i|f(\boldsymbol{x}_i, \boldsymbol{w}), \sigma^2).$$

We then proceeded by finding $\boldsymbol{w}$ by applying Bayes' theorem giving eq. (12.5)

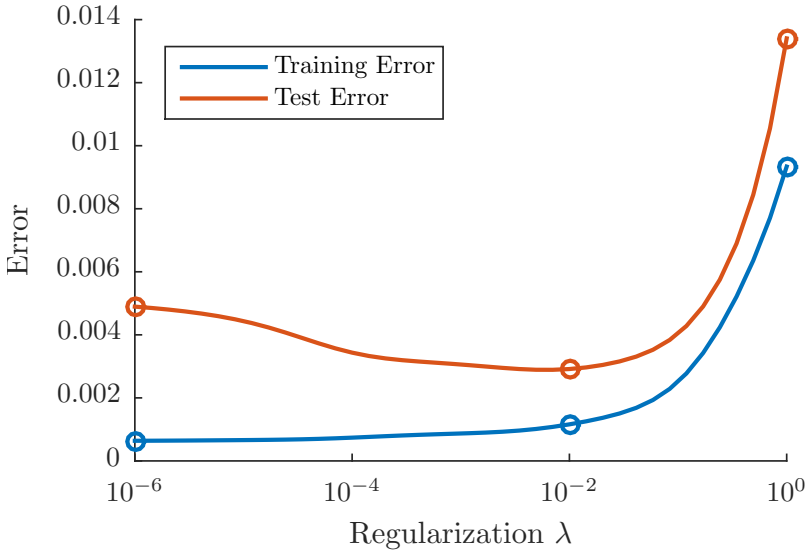


Fig. 12.3: Illustration of our two classifiers with three different threshold values for an inferior classifier.

$$p(\boldsymbol{w}|\boldsymbol{X},\boldsymbol{y}) = \frac{p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{w})p(\boldsymbol{w})}{p(\boldsymbol{y}|\boldsymbol{X})}. \tag{12.5}$$

and then made the assumptions we were only interested in the most probable value of $\boldsymbol{w}$ and importantly assumed the prior could be ignored, i.e. by assuming $p(\boldsymbol{w})$ was an improper uniform distribution. What if we assume the prior can't be ignored? The simplest case is if we assume the prior is normally distributed with diagonal covariance matrix $\delta\boldsymbol{I}$:

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{0}, \delta^2\boldsymbol{I})$$

In this case, the derivation proceed as before:

$$\boldsymbol{w}^* = \arg\max_{\boldsymbol{w}} \left[p(\boldsymbol{w}|\boldsymbol{X},\boldsymbol{y})\right] = \arg\max_{\boldsymbol{w}} \left[\frac{p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{w})p(\boldsymbol{w})}{p(\boldsymbol{y}|\boldsymbol{X})}\right] = \arg\max_{\boldsymbol{w}} \left[\log p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{w})p(\boldsymbol{w})\right]$$

$$\text{where } \log\left(p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{w})p(\boldsymbol{w})\right) = -\frac{1}{2\sigma^2}\sum_{i=1}^{N}\|\boldsymbol{f}(\boldsymbol{x}_i,\boldsymbol{w}) - \boldsymbol{y}_i\|^2 - \frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{2\delta^2}\boldsymbol{w}^T\boldsymbol{w} - \frac{M}{2}\log(2\pi\delta^2).$$

Once again, the constant terms disappear and with re-scaling we recover the least-squares regularized objective with $\lambda = \frac{\sigma^2}{\delta^2}$. In other words, regularization is not an esoteric technique or a trick: It is simply recognizing the prior of $\boldsymbol{w}$ should not be ignored.

If we look a bit beyond this course, the technique of adding a regularization term such as $\lambda\boldsymbol{w}^T\boldsymbol{w}$ to the error term (or alternatively, specifying a different prior $p(\boldsymbol{w})$) is very general, i.e. we also used a prior for the robust estimator in chapter 11. It can be used in conjunction with any weight-based model such as logistic regression or neural networks where it can be used to great effect. The particular choice for the regularization term considered here, $\lambda\boldsymbol{w}^T\boldsymbol{w} = \lambda\|\boldsymbol{w}\|_2^2$, is known

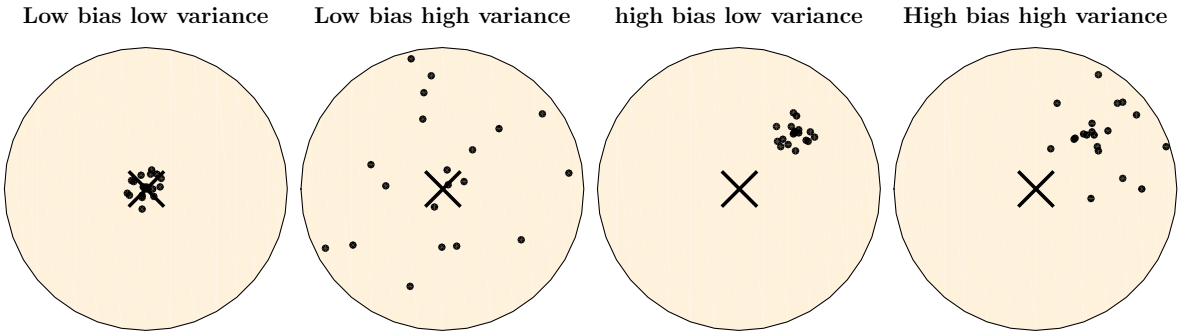


Fig. 12.4: Illustration of bias and variance

as L_2 regularization, however other choices for regularization terms can be considered. A popular alternative choice is the L_1-norm regularization term: $\lambda \left(\sum_i |w_i|\right) = \lambda \|\boldsymbol{w}\|_1$. 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.

12.2 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. 12.4 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 $\boldsymbol{x}$. If $\mathcal{D}$ denotes our training data, a given model learns a function f on the training data to accomplish this task. In fig. 12.5 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 9 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. (9.4))

$$\begin{aligned} E^{\text{gen}}_{\mathcal{M}} &= \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})}\left[L(\boldsymbol{y}, \boldsymbol{f}_{\mathcal{M}}(\boldsymbol{x}))\right] \\ &= \int L(\boldsymbol{y}, \boldsymbol{f}_{\mathcal{M}}(\boldsymbol{x}))p(\boldsymbol{x},\boldsymbol{y})d\boldsymbol{x}d\boldsymbol{y}. \end{aligned} \tag{12.6}$$

Since the generalization error E^{gen} depends on the training data set $\mathcal{D}$, here indicated by the notation $E^{\text{gen}}(\mathcal{D})$, we will in this section consider the expectation of the generalization error over all training data set:

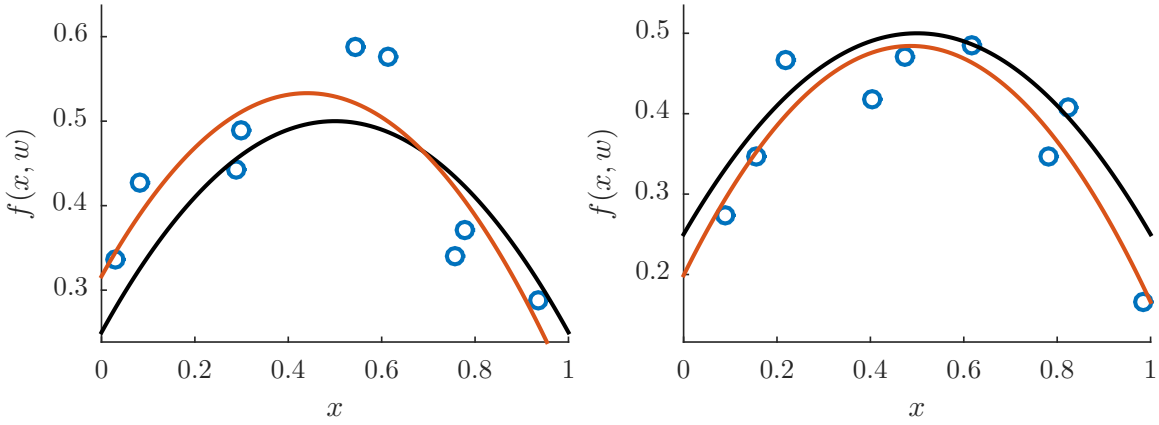


Fig. 12.5: A linear regression model corresponding to a second-order polynomial trained on two different training data sets. The learned model (red line) depends on the training sets. The training set is distributed around the black line.

$$\mathbb{E}_{\mathcal{D}}\left[E^{\text{gen}}\right] = \int E^{\text{gen}}(\mathcal{D})p(\mathcal{D})d\mathcal{D}.$$

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 lets consider the average behavior of the by now well-known linear regression model when trained on different training sets. In fig. 12.6 the three linear regression models are each trained on 10 different training sets and the prediction curves, $f_{\mathcal{D}}$ is 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. 12.6. Formally, this is written as

$$\bar{f}(\boldsymbol{x}) = \mathbb{E}_{\mathcal{D}}\left[f_{\mathcal{D}}(\boldsymbol{x})\right]$$

The black line is the true average of the training sets, i.e. the training points (which are not shown in fig. 12.6) are distributed around this curve. Formally, it is defined as the average value of y given $\boldsymbol{x}$, i.e.:

$$\bar{y}(\boldsymbol{x}) = \mathbb{E}_{y|\boldsymbol{x}}\left[y\right].$$

Considering fig. 12.6 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 has a low bias. Meanwhile, the first model, which is too inflexible, has a high bias because the average of the model $\bar{f}(\boldsymbol{x})$ and the average of the data $\bar{y}(\boldsymbol{x})$ 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.

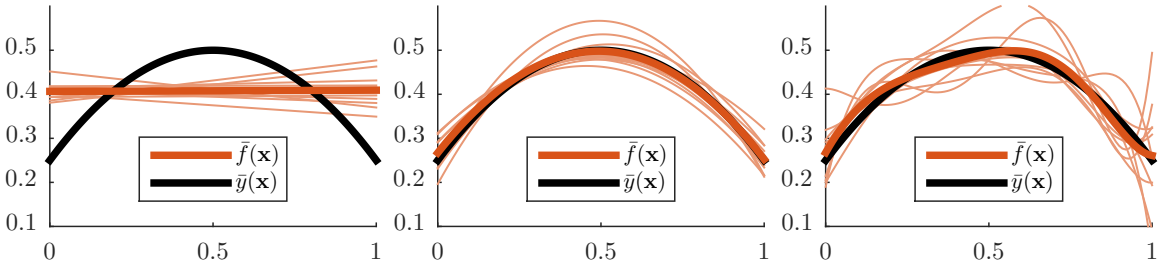


Fig. 12.6: A linear regression model corresponding to a second-order polynomial trained on two different training data sets. The learned models (the thinner red lines) depends on the training sets. The training set is distributed around the black line. The average of all models is shown as the thicker red line.

Derivation of the bias-variance decomposition*

The average generalization error for a training set $\mathcal{D}$ for a square loss is:

$$\mathbb{E}_{\mathcal{D}}\left[E^{\text{gen}}\right] = \mathbb{E}_{\mathcal{D},(\boldsymbol{x},y)}\left[\left(y - f_{\mathcal{D}}(\boldsymbol{x})\right)^2\right],$$

where the expectation can be written out as $\mathbb{E}_{\mathcal{D},(\boldsymbol{x},y)}\left[\,\cdot\,\right] = \int\left[\,\cdot\,\right] p(\boldsymbol{x}, y, \mathcal{D}) d\boldsymbol{x} dy d\mathcal{D}$. We first assume $\boldsymbol{x}$ to be fixed and consider the average:

$$\begin{aligned}
&\mathbb{E}_{\mathcal{D},y|\boldsymbol{x}}\left[\left(y - f_{\mathcal{D}}(\boldsymbol{x})\right)^2\right] \\
&= \mathbb{E}_{\mathcal{D},y|\boldsymbol{x}}\left[\left(y - \bar{y}(\boldsymbol{x}) + \bar{y}(\boldsymbol{x}) - f_{\mathcal{D}}(\boldsymbol{x})\right)^2\right] \\
&= \mathbb{E}_{y|\boldsymbol{x}}\left[\left(y - \bar{y}(\boldsymbol{x})\right)^2\right] + \mathbb{E}_{\mathcal{D}}\left[\left(\bar{y}(\boldsymbol{x}) - f_{\mathcal{D}}(\boldsymbol{x})\right)^2\right] + 2\mathbb{E}_{\mathcal{D},y|\boldsymbol{x}}\left[\left(y - \bar{y}(\boldsymbol{x})\right)\left(\bar{y}(\boldsymbol{x}) - f_{\mathcal{D}}(\boldsymbol{x})\right)\right].
\end{aligned}$$

The last term is zero since

$$\mathbb{E}_{\mathcal{D},y|\boldsymbol{x}}\left[\left(y - \bar{y}(\boldsymbol{x})\right)\left(\bar{y}(\boldsymbol{x}) - f_{\mathcal{D}}(\boldsymbol{x})\right)\right] = \mathbb{E}_{y|\boldsymbol{x}}\left[y - \bar{y}(\boldsymbol{x})\right]\mathbb{E}_{\mathcal{D}}\left[\bar{y}(\boldsymbol{x}) - f_{\mathcal{D}}(\boldsymbol{x})\right],$$

and $\mathbb{E}_{y|\boldsymbol{x}}\left[y - \bar{y}(\boldsymbol{x})\right] = 0$. If we look at the second term we can do the same trick once again:

$$\begin{aligned}
&\mathbb{E}_{\mathcal{D}}\left[\left(\bar{y}(\boldsymbol{x}) - f_{\mathcal{D}}(\boldsymbol{x})\right)^2\right] \\
&= \mathbb{E}_{\mathcal{D}}\left[\left(\bar{y}(\boldsymbol{x}) - \bar{f}(\boldsymbol{x}) + \bar{f}(\boldsymbol{x}) - f_{\mathcal{D}}(\boldsymbol{x})\right)^2\right] \\
&= \mathbb{E}_{\mathcal{D}}\left[\left(\bar{y}(\boldsymbol{x}) - \bar{f}(\boldsymbol{x})\right)^2\right] + \mathbb{E}_{\mathcal{D}}\left[\left(\bar{f}(\boldsymbol{x}) - f_{\mathcal{D}}(\boldsymbol{x})\right)^2\right] + 2\mathbb{E}_{\mathcal{D}}\left[\left(\bar{y}(\boldsymbol{x}) - \bar{f}(\boldsymbol{x})\right)\left(\bar{f}(\boldsymbol{x}) - f_{\mathcal{D}}(\boldsymbol{x})\right)\right].
\end{aligned}$$

Again, since $\mathbb{E}_{\mathcal{D}}\left[\bar{f}(\boldsymbol{x}) - f_{\mathcal{D}}(\boldsymbol{x})\right] = 0$, the third term is zero in the above. Putting all these things together, we obtain:

$$\mathbb{E}_{\mathcal{D},y|\boldsymbol{x}}\left[\left(y - f_{\mathcal{D}}(\boldsymbol{x})\right)^2\right] \tag{12.7}$$

$$= \mathbb{E}_{y|\boldsymbol{x}}\left[\left(y - \bar{y}(\boldsymbol{x})\right)^2\right] + \left(\bar{y}(\boldsymbol{x}) - \bar{f}(\boldsymbol{x})\right)^2 + \mathbb{E}_{\mathcal{D}}\left[\left(\bar{f}(\boldsymbol{x}) - f_{\mathcal{D}}(\boldsymbol{x})\right)^2\right]. \tag{12.8}$$

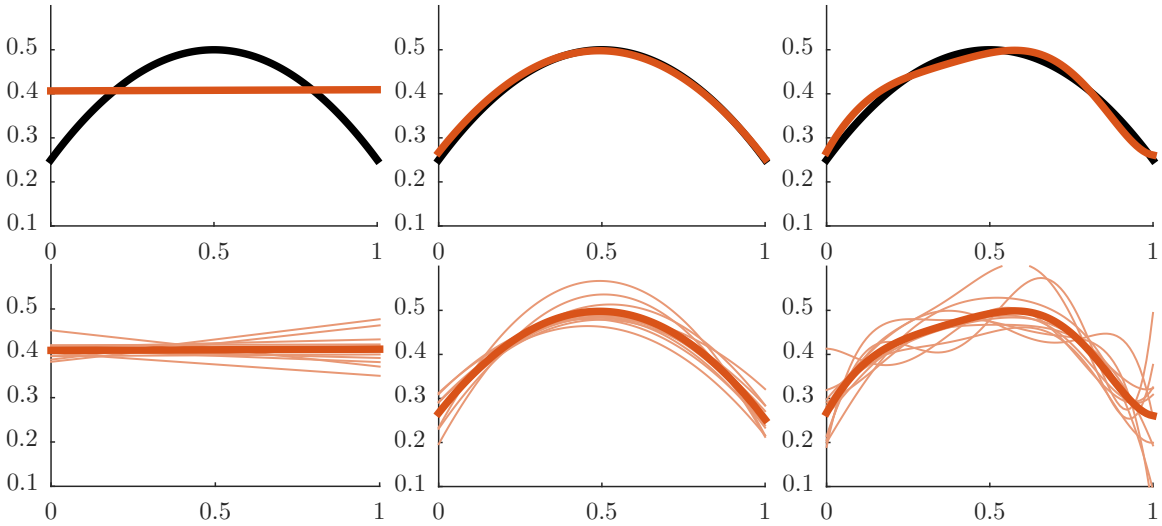


Fig. 12.7: Bias-variance decomposition for the three linear regression models. In the top row is shown the bias term. Namely, how much the average values of the models trained on different random data sets (illustrated with the thick red line) differ from the true mean values of the data illustrated 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.

The last term is simply the variance of $f_{\mathcal{D}}(\boldsymbol{x})$ computed with respect to $\boldsymbol{x}$ and the first term too is just the variance of y conditional on $\boldsymbol{x}$. Using this the above can be written as:

$$\mathbb{E}_{\mathcal{D},y|\boldsymbol{x}}\left[\left(y - f_{\mathcal{D}}(\boldsymbol{x})\right)^2\right] = \mathrm{Var}_{y|\boldsymbol{x}}\left[y\right] + \mathrm{Var}_{\mathcal{D}}\left[f_{\mathcal{D}}(\boldsymbol{x})\right] + \left(\bar{y}(\boldsymbol{x}) - \bar{f}(\boldsymbol{x})\right)^2. \tag{12.9}$$

Taking the expectation with respect to $\boldsymbol{x}$ and rearranging we finally obtain the result:

$$\mathbb{E}_{\mathcal{D}}\left[E^{\mathrm{gen}}\right] = \mathbb{E}_{\boldsymbol{x}}\left[\mathrm{Var}_{y|\boldsymbol{x}}\left[y\right] + \left(\bar{y}(\boldsymbol{x}) - \bar{f}(\boldsymbol{x})\right)^2 + \mathrm{Var}_{\mathcal{D}}\left[f_{\mathcal{D}}(\boldsymbol{x})\right]\right]. \tag{12.10}$$

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 equations tells that us the error of any model is decomposed into the following three parts

- The first term $\mathrm{Var}_{y|\boldsymbol{x}}\left[y\right]$ 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 $\left(\bar{y}(\boldsymbol{x}) - \bar{f}(\boldsymbol{x})\right)^2$ is the *bias* term. It tells us how much each trained model varies compared to the true mean of the data $\bar{y}(\boldsymbol{x})$.
- The third term $\mathrm{Var}_{\mathcal{D}}\left[f_{\mathcal{D}}(\boldsymbol{x})\right]$ 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. 12.7 we have shown what contributes to the two terms. In the top-row is the variance term (the spread of the models) and in the bottom row is the bias terms (the difference between the models mean values and the data mean values). 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

12.1. Spring 2013 question 26: 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.

13

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.

13.1 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.

13.1.1 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 $\boldsymbol{x} = (x_1, \ldots, x_M)$) 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. 13.1[1] is illustrated a simple neural network with one hidden layer. The input layer consists of three neurons, the hidden layer of three 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

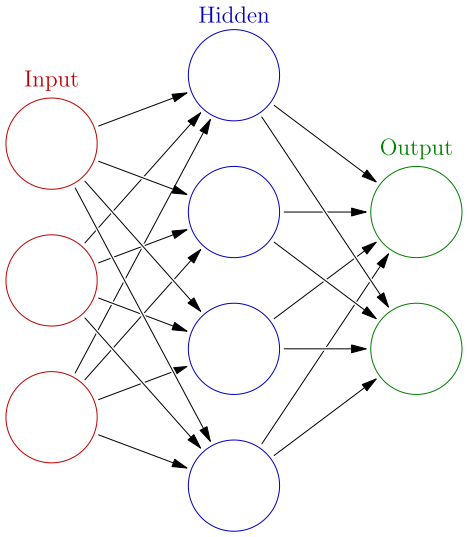


Fig. 13.1: Simple artificial neural network (ANN) consisting of three input units in the input layer, a single hidden layer with four hidden units and two output units in the output layer. This neural network would implement a mapping $\boldsymbol{f} : \mathbb{R}^3 \to \mathbb{R}^2$ and would be suitable for regression or classification in the case of two output variables or classes.

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:

$$\boldsymbol{f} : \mathbb{R}^M \to \mathbb{R}^D$$

which maps from $\boldsymbol{x}$ to $\boldsymbol{y}$: $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$; thus the neural network is useful for solving a (multi-dimensional) regression or classification problem.

13.1.2 The forward pass in details

Recall the basic linear regression model in which the output y is predicted from the rule

$$f(\boldsymbol{x}, \boldsymbol{w}) = \sum_{i=1}^{M} x_i w_i + w_0$$

If we let

$$\tilde{\boldsymbol{x}} = \begin{bmatrix} 1 & x_1 & x_2 & \dots & x_M \end{bmatrix}^T$$

we can write this in a more condensed form

$$f(\boldsymbol{x}, \boldsymbol{w}) = \tilde{\boldsymbol{x}}^T \boldsymbol{w}.$$

The forward pass in a neural network now proceeds as follows for vector $\boldsymbol{x}$:

- Each neuron i in the input layer is initialized to have activity x_i.

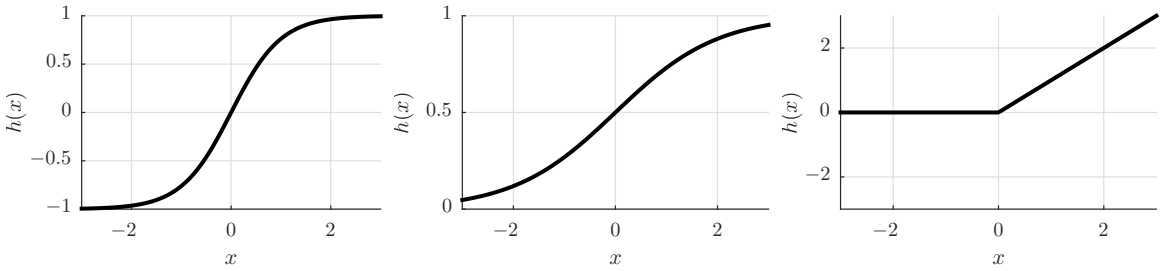


Fig. 13.2: Different choices of activation function. (Left:) hyperbolic tangent: $h(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, (middle:) logistic sigmoid $h(x) = (1 + e^{-x})^{-1}$ and (right:) rectified linear unit: $h(x) = 0$ if $x < 0$ and otherwise $h(x) = x$. The basic information-processing 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 $a_j^{(1)} = \tilde{\boldsymbol{x}}^T \boldsymbol{w}_j^{(1)}$. Notice this is just a real number.
- Each of the H hidden unit is transformed using a nonlinear *activation function* h to give $z_j^{(1)} = h(a_j^{(1)})$. We then define

$$\boldsymbol{z}^{(1)} = \begin{bmatrix} z_1^{(1)} & z_2^{(1)} & \dots & z_H^{(1)} \end{bmatrix}^T$$

- Output neuron k is given an activation of $a_k^{(2)} = \left(\tilde{\boldsymbol{z}}^{(1)}\right)^T w_k^{(2)}$
- 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

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} z_1^{(2)} & z_2^{(2)} & \dots & z_D^{(2)} \end{bmatrix}^T.$$

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

$$W^{(1)} = \begin{bmatrix} \boldsymbol{w}_1^{(1)} & \boldsymbol{w}_2^{(1)} & \dots \boldsymbol{w}_H^{(1)} \end{bmatrix} \quad \text{and} \quad W^{(2)} = \begin{bmatrix} \boldsymbol{w}_1^{(1)} & \boldsymbol{w}_2^{(1)} & \dots \boldsymbol{w}_D^{(1)} \end{bmatrix}.$$

The activation of the kth output neuron is simply:

$$f_k(\boldsymbol{x}, \boldsymbol{w}) = h^{(2)} \left(\sum_{j=1}^{H} W_{kj}^{(2)} z_j^{(1)} \right) \tag{13.1}$$

$$= h^{(2)} \left(\sum_{j=1}^{H} W_{kj}^{(2)} h^{(1)} \left(\tilde{\boldsymbol{x}}^T \boldsymbol{w}_j^{(1)} \right) \right). \tag{13.2}$$

The activation function $h^{(1)}$ of the hidden units could be chosen as the hyperbolic tangent

$$h^{(1)}(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

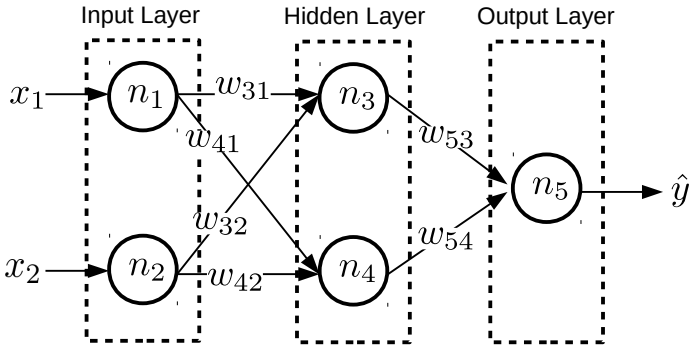


Fig. 13.3: Simple neural network of 6 weights and with one hidden layer with 2 neurons.

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. 13.2.

Example

Consider the feedforward neural network shown in fig. 13.3. The network has no bias weights. Suppose the weights of the neural network after training are

$$\begin{aligned} w_{31} &= 0.05, \quad & w_{41} &= 0, \quad & w_{32} &= 0.1, \\ w_{42} &= -0.05, \quad & w_{53} &= 0.1, \quad & w_{54} &= -10 \end{aligned}$$

and the activation function of all five neurons is the 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 can then be computed as follows: First compute

$$\begin{aligned} x_3 &= h(x_1 0.05 + x_2 0.1) = h(1/8) = 1/8, \\ x_4 &= h(x_1 0 + x_2(-0.05)) = h(-1/20) = -1/200. \end{aligned}$$

Then the output of the neural network is

$$x_5 = h(x_3 0.1 + x_4(-10)) = h(1/16) = 1/16.$$

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 $\boldsymbol{z}^{(0)} = \boldsymbol{x}$ as the input activation
- For each layer $l = 1, \ldots, L$ set $z^{(l)} = h^{(l)}\left((\boldsymbol{W}^{(l)})^T \tilde{\boldsymbol{z}}^{(l-1)}\right)$.
- Return as output $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{z}^{(L)}$.

13.2 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 $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{w})$. For a fixed $\boldsymbol{w}$ this function maps $\boldsymbol{x}$ values to $\boldsymbol{y}$ values such that the vector $\boldsymbol{w}$ contains all the weight-matrices in the network $\boldsymbol{w} = (\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \cdots)$.

We are thus faced with a standard supervised learning problem where we are given instances of observations $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$ and corresponding targets $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_N$, 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 $\boldsymbol{x}_i$ to the corresponding $\boldsymbol{y}_i$, we will assume $\boldsymbol{y}_i$ is normally distributed around the prediction of the neural network. The probability density of observation $\boldsymbol{y}_i$ is then:

$$p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{w}) = \mathcal{N}(\boldsymbol{y}_i|\boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{w}), \sigma^2).$$

Given all observations $\boldsymbol{X}$ and targets $\boldsymbol{y}$ the likelihood of the entire dataset is

$$p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w}) = \prod_{i=1}^{N} p(\boldsymbol{y}_i|\boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{w}), \sigma^2).$$

Similar to eq. (12.5) in chapter 12 we apply Bayes' theorem using a standard normal prior distribution of $\boldsymbol{w}$:

$$p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{y}) = \frac{p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w})p(\boldsymbol{w})}{p(\boldsymbol{y}|\boldsymbol{X})}, \quad p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{0}, \delta^2\boldsymbol{I}). \tag{13.3}$$

We wish to find the value of $\boldsymbol{w}$ which has as high probability density as possible so taking the logarithm of the above expression and maximizing we obtain:

$$\boldsymbol{w}^* = \arg\max_{\boldsymbol{w}} \left[p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{y})\right] = \arg\max_{\boldsymbol{w}} \left[\log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w})p(\boldsymbol{w})\right]$$

$$\log\left(p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w})p(\boldsymbol{w})\right) = -\frac{1}{2\sigma^2}\sum_{i=1}^{N} \|\boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{w}) - \boldsymbol{y}_i\|^2 - \frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{2\delta^2}\boldsymbol{w}^T\boldsymbol{w} - \frac{M}{2}\log(2\pi\delta^2).$$

Ignoring constant terms the above is equivalent to *minimizing* the sum-of-squares error function:

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N} \|\boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{w}) - \boldsymbol{y}_i\|^2 + \frac{1}{2}\lambda\boldsymbol{w}^T\boldsymbol{w}$$

where $\lambda = \frac{\delta}{\sigma}$ is the regularization strength we have to specify. Training a neural network thus consists of finding the *best* value of $\boldsymbol{w}$ which minimizes E:

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} E(\boldsymbol{w}). \tag{13.4}$$

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 $\boldsymbol{w}^*$. Thus, headway on solving the problem eq. (13.4) can be used in a variety of contexts.

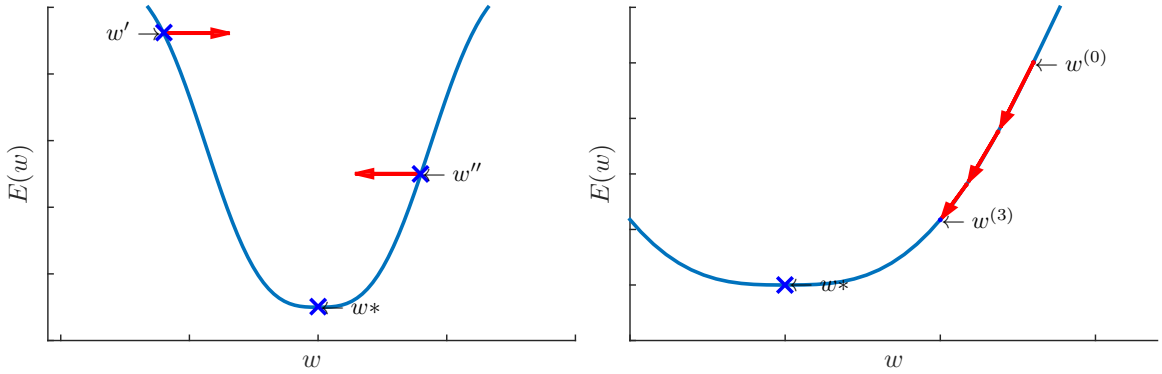


Fig. 13.4: (left:) Value of error function in a one-dimensional example. Weights at w' should move right and weights at w'' should move left in order to approach the minimum point $w*$ of $E(w)$. (right:) Gradient descent algorithm applied for three steps starting at $w^{(0)}$

13.2.1 Gradient Descent*

The problem is that it is impossible to analytically solve for $\boldsymbol{w}^*$. Instead, the following iterative algorithm is proposed:

- Start from an initial guess at $\boldsymbol{w}^*$, $\boldsymbol{w}^{(0)}$.
- At step t, modify $\boldsymbol{w}^{(t-1)}$ by a small amount $d\boldsymbol{w}$ to produce a better guess $\boldsymbol{w}^{(t)}$:

$$\boldsymbol{w}^{(t)} = \boldsymbol{w}^{(t-1)} + d\boldsymbol{w},$$

  where we leave it for later how to compute $d\boldsymbol{w}$.
- Do this for a large number T of iterations to produce (hopefully!) better and better guesses of $\boldsymbol{w}^{(0)}, \boldsymbol{w}^{(1)}, \ldots, \boldsymbol{w}^{(T)}$

After T iterations $\boldsymbol{w}^{(T)}$ is then used as the "best" available guess at $\boldsymbol{w}^*$. 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 $\boldsymbol{w}$ is one dimensional (the neural network only contains a single "weight") and suppose E as a function of w looks like fig. 13.4. 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 $w^{(t-1)}$ we can consider the simple update rule:

$$\theta^{(t)} = \theta^{(t-1)} + dw,$$

where $\epsilon > 0$ is called the *learning rate* of the method, usually set somewhere in the interval $[0, 1]$ for instance $\epsilon = 1/5$. It is easy to check this indeed works – in fig. 13.4 is plotted $w^{(t)}$ and $(E(w^{(t)})$ and as a function of t when this rule is applied for 20 steps. Notice that the method "slows down" when $w^{(t)}$ is closer to w^*; 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 expand [2] E around w' to obtain:

$$E(w' + dw) \approx E(w') + dw \frac{dE}{dw}(w') \tag{13.5}$$

$$\approx E(w') + dwg \tag{13.6}$$

$$\text{where } g = \frac{dE}{dw}(w'). \tag{13.7}$$

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. 13.4). 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 cant 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:

$$E(\boldsymbol{w}' + \boldsymbol{d}w) \approx E(\boldsymbol{w}') + d\boldsymbol{w}^T \boldsymbol{g} \tag{13.8}$$

$$\approx E(\boldsymbol{w}') + d\boldsymbol{w}^T \boldsymbol{g} \tag{13.9}$$

$$\text{where } \boldsymbol{g} = \nabla E(\boldsymbol{w}') \tag{13.10}$$

The multi-dimensional Taylor expansion is briefly reviewed in appendix A. Thus, if we select $d\boldsymbol{w} = -\epsilon \boldsymbol{g}$ we again get

[2] See also `https://en.wikipedia.org/wiki/Taylor_series` and appendix A.

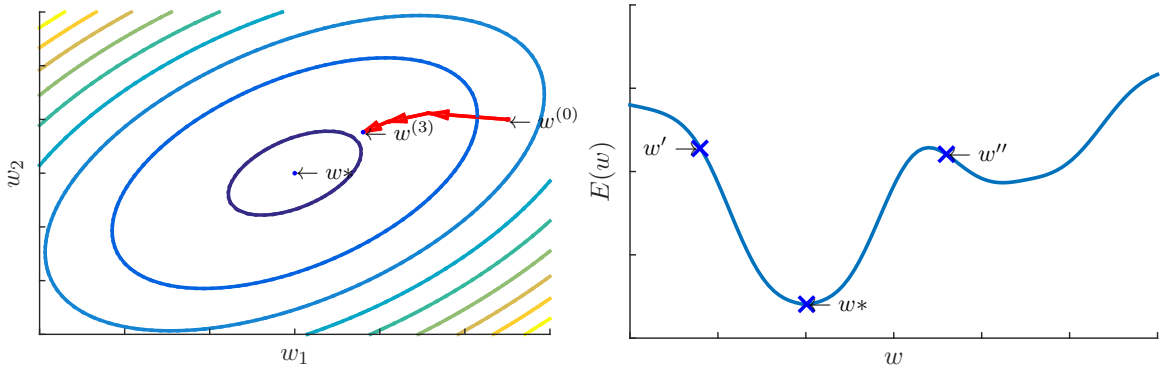


Fig. 13.5: (left:) Value of error function in a two dimensional example as a contour plot along with three steps of the gradient descent algorithm. Notice the step size slows down when moving towards the minimum. (right:) An example with two local minima. If the gradient descent method is initialized 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.

$$E(\boldsymbol{w}^{(t)}) = E(\boldsymbol{w}^{(t-1)} + d\boldsymbol{w}) \approx E(\boldsymbol{w}^{(t-1)}) - \epsilon \|\boldsymbol{g}\|^2 \leq E(\boldsymbol{w}^{(t-1)}),$$

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 $\boldsymbol{w}^*$, $\boldsymbol{w}^{(0)}$
- For each $t = 1, \ldots, T$, compute the divergence $\boldsymbol{g}^{(t-1)} = \nabla E(\boldsymbol{w}^{(t-1)})$
- Compute $\boldsymbol{w}^{(t)} = \boldsymbol{w}^{(t-1)} - \epsilon \boldsymbol{g}$.
- Do this for a large number T of iterations to produce a sequence of (hopefully!) better and better guesses at $\boldsymbol{w}^*$: $\boldsymbol{w}^{(0)}, \boldsymbol{w}^{(1)}, \ldots, \boldsymbol{w}^{(T)}$.

In fig. 13.5 we have illustrated how θ is updated for three iterations in an example where $\boldsymbol{w}$ 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 $\boldsymbol{g}$ efficiently. If we consider the i'th coordinate of $\boldsymbol{g}$, g_i, this can be computed as:

$$g_i = \frac{\partial E(\boldsymbol{w})}{\partial w_i} \tag{13.11}$$

$$= \frac{\partial}{\partial w_i} \left(\sum_{i=1}^{N} \|f(\boldsymbol{x}_i, \boldsymbol{w}) - \boldsymbol{y}_i\|^2 \right) \tag{13.12}$$

$$= 2 \sum_{i=1}^{N} \left[\sum_{k=1}^{D} (f_k(\boldsymbol{x}_i, \boldsymbol{w}) - y_{ik}) \frac{\partial f_k(\boldsymbol{x}_i, \boldsymbol{w})}{\partial w_i} \right]. \tag{13.13}$$

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 tricks 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 mention is when E has different local minima. In fig. 13.5 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 $\boldsymbol{w}^{(T)}$, as well as the training error $E(\boldsymbol{w}^{(T)})$, will depend on how the model is initialized $\boldsymbol{w}^{(0)}$ as well as other parameters of the training.

13.3 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. However, because we wan't the neural network to distinguish between more than two classes, we will use a slightly more general transformation. Suppose therefore $\boldsymbol{y}$ corresponds to a classification problem. To consider the general case lets assume there are C output classes, for instance $C = 3$ corresponding to *"dog"*, *"cat"* and *"cow"*. To solve this problem, assume the classes are in a 1-of-K encoding such that:

$$\boldsymbol{y} = \begin{bmatrix} 0\ 1\ 0 \\ 1\ 0\ 0 \\ 1\ 0\ 0 \\ 0\ 0\ 1 \\ \vdots \end{bmatrix},$$

implying that the first observation is in class 1 (a cat), the second and third observations are in class 1 (a dog) and the fourth observation is in class 3, a cow. In the following we will always assume $\boldsymbol{y}$ is in one-of-K encoding and therefore $y_{ik} = 1$ 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 (this is another way to say that each row in $\boldsymbol{y}$ contains exactly one 1).

What we need in order to apply the machinery from eqs. (13.3) and (13.4) is a way to represent the probability of the data given $\boldsymbol{w}$:

$$p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w}) = \prod_{i=1}^{N} p(\boldsymbol{y}_i = \begin{bmatrix} y_{i1}\ y_{i2} \cdots y_{iC} \end{bmatrix} |\boldsymbol{x}_i, \boldsymbol{w}). \tag{13.14}$$

Regardless of what that probability is, we can denote by $\hat{y}_{ik}$ the *probability* that observation i is in class k:

$$p(y_i = k|\boldsymbol{x}_i, \boldsymbol{w}) = \hat{y}_{ik}$$

(naturally, $\hat{y}_{ik}$ depends on $\boldsymbol{x}_i$ and $\boldsymbol{w}$). Notice that the only restrictions on $\hat{y}_{ik}$ is that (i) $0 \leq \hat{y}_{ik} \leq 1$ and (ii) $\sum_{k=1}^{C} \hat{y}_{ik} = 1$ and that we can write the likelihood of observation i as:

$$p(\boldsymbol{y}_i = \begin{bmatrix} y_{i1} & y_{i2} & \cdots & y_{iC} \end{bmatrix} | \boldsymbol{x}_i, \boldsymbol{w}) = \prod_{k=1}^{C} \hat{y}_{ik}^{y_{ik}}$$

This is easily verified by observing that for instance:

$$p(\boldsymbol{y}_i = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T | \boldsymbol{x}_i, \boldsymbol{w}) = \hat{y}_{i1}^0 \, \hat{y}_{i2}^1 \, \hat{y}_{i3}^0 = \hat{y}_{i2} = p(y_i = 2 | \boldsymbol{x}_w, \boldsymbol{w}).$$

We still need a way to represent $\hat{y}_{ik}$ using a neural network and this is accomplished by the *softmax* function. To solve the problem consider a neural network that maps to $C = 3$ output units:

$$\boldsymbol{a}^{(L)} = \begin{bmatrix} a_1^{(L)} & a_2^{(L)} & a_3^{(L)} \end{bmatrix}$$

We then apply the *softmax function* to $\boldsymbol{a}(L)$ to produce $\hat{\boldsymbol{y}}$. The softmax function applied to a P-dimensional vector $\boldsymbol{a}$ defined as

$$\text{softmax}(\boldsymbol{a}) = \begin{bmatrix} \frac{e^{a_1}}{S} & \frac{e^{a_2}}{S} & \cdots & \frac{e^{a_P}}{S} \end{bmatrix}^T, \quad \text{and} \quad S = \sum_{k=1}^{P} e^{a_k}.$$

Lets consider a concrete example. Suppose in the previous example we have as output from the neural network $a_1 = 2$, $a_2 = 1$ and $a_3 = -1$. We then have that

$$e^{a_1} \approx 7.39, \quad e^{a_2} \approx 2.72, \quad e^{a_3} \approx 0.37 \quad \text{and} \quad S = 10.5$$

and so

$$\hat{y} = \text{softmax}(\boldsymbol{a}) \approx \begin{bmatrix} \frac{7.39}{10.5} & \frac{2.72}{10.5} & \frac{0.37}{10.5} \end{bmatrix} = \begin{bmatrix} 0.7 & 0.26 & 0.04 \end{bmatrix}.$$

Notice, these numbers sum to one and are positive and this must be the case when we consider the definition of the softmax function. Taken together, this defines a new neural-network function

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{w}) = \text{softmax}(\boldsymbol{a}^{(L)}),$$

which returns a 3-dimensional vector where all elements are positive and sum to one. We can therefore interpret the output of $\hat{\boldsymbol{y}}_i = \begin{bmatrix} \hat{y}_{i1} & \hat{y}_{i2} & \hat{y}_{i3} \end{bmatrix} = \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{w})$ as a vector of probabilities, namely, the probability $\boldsymbol{x}$ belongs to class 1 (is a dog) is $\hat{y}_1 = 0.7$, the probability $\hat{y}_2$ belongs to class 2 (is a cat) is $\hat{y}_2 = 0.26$ and finally the probability $\boldsymbol{x}$ belongs to class 3 (a cow) is $\hat{y}_3 = 0.04$. Plugging this into the likelihood eq. (13.14) we obtain:

$$p(\boldsymbol{y} | \boldsymbol{X}, \boldsymbol{w}) = \prod_{i=1}^{N} p(\boldsymbol{y}_i = \begin{bmatrix} y_{i1} & y_{i2} & \cdots & y_{iC} \end{bmatrix} | \boldsymbol{x}_i, \boldsymbol{w}) = \prod_{i=1}^{N} \prod_{k=1}^{C} f_k(\boldsymbol{x}_i, \boldsymbol{w})^{y_{ik}}. \tag{13.15}$$

The error function can then be obtained from eq. (13.15) exactly as in eqs. (13.3) and (13.4) by applying Bayes' theorem, maximizing to find $\boldsymbol{w}^*$ and ignoring constant terms. Again using a prior on $\boldsymbol{w}$ we obtain:

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} E(\boldsymbol{w}), \quad E(\boldsymbol{w}) = -\sum_{i=1}^{N} \sum_{k=1}^{C} y_{ik} \log f_k(\boldsymbol{x}_i, \boldsymbol{w}) + \frac{1}{2} \lambda \boldsymbol{w}^T \boldsymbol{w} \tag{13.16}$$

This error function is known as the *cross-entropy* and can be optimized using gradient descent.

13.3.1 Connection to logistic regression

Logistic regression is a special case of a neural network where there is no hidden layers and two output neurons. For simplicity let $\lambda = 0$ and recall the error function from logistic regression was given as ($\sigma(x) = \frac{1}{1+e^{-x}}$)

$$E^{\text{log.reg}}(\boldsymbol{w}) = \sum_{i=1}^{N} \left[y_i \log \sigma(\tilde{\boldsymbol{x}}_i^T \boldsymbol{w}) + (1 - y_i) \log(1 - \sigma(\tilde{\boldsymbol{x}}_i^T \boldsymbol{w})) \right]. \tag{13.17}$$

Suppose we apply the multi-layer neural network with no hidden layers to the same problem. This neural network will have *two* output neurons (corresponding to the two classes $y_i = 0$ and $y_i = 1$) and y_i will now be in a 1-of-K encoding with $K = 2$. The error function becomes

$$\begin{aligned}
f_k(\boldsymbol{x}, \boldsymbol{\theta}) &= \frac{e^{a_k^{(2)}}}{\sum_{j=1}^{2} e^{a_j^{(2)}}} = \frac{e^{\boldsymbol{w}_k^T \tilde{\boldsymbol{x}}}}{e^{\boldsymbol{w}_1^T \tilde{\boldsymbol{x}}} + e^{\boldsymbol{w}_2^T \tilde{\boldsymbol{x}}}} \\
\boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{\theta}) &= \begin{bmatrix} \frac{1}{1+e^{\boldsymbol{w}_2^T \tilde{\boldsymbol{x}} - \boldsymbol{w}_1^T \tilde{\boldsymbol{x}}}} \\ \frac{e^{\boldsymbol{w}_2^T \tilde{\boldsymbol{x}} - \boldsymbol{w}_1^T \tilde{\boldsymbol{x}}}}{1+e^{\boldsymbol{w}_2^T \tilde{\boldsymbol{x}} - \boldsymbol{w}_1^T \tilde{\boldsymbol{x}}}} \end{bmatrix} = \begin{bmatrix} \frac{1}{1+e^{-\boldsymbol{w}^T \tilde{\boldsymbol{x}}}} \\ 1 - \frac{1}{1+e^{-\boldsymbol{w}^T \tilde{\boldsymbol{x}}}} \end{bmatrix} = \begin{bmatrix} \sigma(\boldsymbol{w}^T \tilde{\boldsymbol{x}}) \\ 1 - \sigma(\boldsymbol{w}^T \tilde{\boldsymbol{x}}) \end{bmatrix} \\
E^{\text{ANN}}(\boldsymbol{\theta}) &= -\sum_{i=1}^{N} \sum_{k=1}^{C} y_{ik} \log f_k(\boldsymbol{x}_i, \boldsymbol{\theta}) \\
&= -\sum_{i=1}^{N} \left[y_{i1} \log \sigma(\boldsymbol{w}^T \tilde{\boldsymbol{x}}_i) + y_{i2} \log(1 - \sigma(\boldsymbol{w}^T \tilde{\boldsymbol{x}}_i)) \right] \\
&= -\sum_{i=1}^{N} \left[y_i \log \sigma(\boldsymbol{w}^T \tilde{\boldsymbol{x}}_i) + (1 - y_i) \log(1 - \sigma(\boldsymbol{w}^T \tilde{\boldsymbol{x}}_i)) \right],
\end{aligned}$$

with $\boldsymbol{w} = \boldsymbol{w}_1 - \boldsymbol{w}_2$. Thus, we see the error function is the same with a re-parameterizations of the weights.

13.3.2 Multinomial regression

Since logistic regression corresponds to a neural network with no hidden layer and two output neurons, logistic regression can easily be extended to multi-class classification using no hidden layer and the more general softmax activation function. Put more simply, this extension is equivalent to simply letting $L = 0$ in which case:[3]

$$\hat{\boldsymbol{y}}_i = \text{softmax}(\boldsymbol{a}^{(0)}) = \text{softmax}(\boldsymbol{W} \boldsymbol{x}_i).$$

Suppose $\boldsymbol{W} = \begin{bmatrix} \tilde{\boldsymbol{w}}_1 & \tilde{\boldsymbol{w}}_2 & \cdots & \tilde{\boldsymbol{w}}_C \end{bmatrix}$, the problem is slightly over-parameterized and it is common to introduce

$$\text{for } k = 1, \cdots, C\text{: } \boldsymbol{w}_k = \tilde{\boldsymbol{w}}_k - \tilde{\boldsymbol{w}}_C$$

[3] Similar to logistic regression, it is assumed $\boldsymbol{X}$ has been extended to include an extra column of ones.

in which case we get:

$$\text{for } k = 1, \cdots, C-1\text{: } \hat{y}_{ik} = p(y_i = k|\boldsymbol{x}_i) = \frac{e^{\tilde{\boldsymbol{w}}_k^T \boldsymbol{x}_i}}{\sum_{k'=1}^{C} e^{\tilde{\boldsymbol{w}}_{k'}^T \boldsymbol{x}_i}} = \frac{e^{\boldsymbol{w}_k^T \boldsymbol{x}_i}}{1 + \sum_{k'=1}^{C-1} e^{\boldsymbol{w}_{k'}^T \boldsymbol{x}_i}}$$
$$\text{for } k = C\text{: } \hat{y}_{ik} = p(y_i = k|\boldsymbol{x}_i) = \frac{1}{1 + \sum_{k'=1}^{C-1} e^{\boldsymbol{w}_{k'}^T \boldsymbol{x}_i}}$$

the error function is then simply found using eq. (13.16):

$$E^{\text{Mul.regression}}(\boldsymbol{w}_1, \cdots, \boldsymbol{w}_{C-1}) = -\sum_{i=1}^{N}\sum_{k=1}^{C} y_{ik} \log \hat{y}_{ik} + \frac{1}{2}\lambda \boldsymbol{w}^T \boldsymbol{w} \tag{13.18}$$

Maximizing this error function to find $\boldsymbol{w}_1, \cdots, \boldsymbol{w}_{C-1}$ is known as *multinomial regression.*

13.3.3 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, 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 units in the hidden layers 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 9.

13.4 Advanced topics*

In this section we will briefly sketch upon some advanced topics of neural network training

13.4.1 Mini-batching

Gradient descent requires computing the divergence of the error $\nabla E(\boldsymbol{w})$ 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 $\mathcal{D}_1, \ldots, \mathcal{D}_m$ each containing B observations. Instead of using the gradient:

$$\boldsymbol{g} = \nabla E(\boldsymbol{w}) = \nabla \left(\sum_{i=1}^{N} \| \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{w}) - \boldsymbol{y}_i \|^2 \right)$$

we use the approximate gradients computed for the observations in each batch k:

$$\boldsymbol{g}_k = \nabla \tilde{E}(\boldsymbol{w}) = \frac{N}{B} \nabla \left(\sum_{i \in \mathcal{D}_k} \| \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{w}) - \boldsymbol{y}_i \|^2 \right).$$

The gradient-descent method is thus simply

- Start at $\boldsymbol{w}^{(0)}$
- For each iteration t:
- For each batch $k = 1, \ldots, m$:
- Update $\boldsymbol{w}^{(t)} = \boldsymbol{w}^{(t-1)} - \epsilon \boldsymbol{g}_k^{(t)}$

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 $\boldsymbol{g}$ at a point $\boldsymbol{w}^{(t)}$ with an approximate gradient $\boldsymbol{g}_k$. 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.

13.4.2 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. 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 $\boldsymbol{A}$ be the matrix representing the image, we first apply the neural network to pixels $\boldsymbol{A}_{[1:11]\times[1:11]}$, then $\boldsymbol{A}_{[5:16]\times[1:11]}$, then $\boldsymbol{A}_{[9:20]\times[1:11]}$ and so on in both the horizontal and vertical direction until we apply the neural network to $\boldsymbol{A}_{[989:999]\times[989:999]}$.

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. 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.

13.4.3 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 $\boldsymbol{x}_i$, we simply predict $\boldsymbol{x}_i$ from $\boldsymbol{x}_i$. That is we model

$$\boldsymbol{x}_i = \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{w}) + \epsilon,$$

where ϵ is noise. Notice, this is entirely trivial when one has a working neural network implementation – simply replace y_i with $\boldsymbol{x}_i$. 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.

13.4.4 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 code 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 $\boldsymbol{x}$ is a sequence of letters in a one-of-K coding i.e. $\boldsymbol{x} = (\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_S)$ for an S-long sequence. We then introduce a new vector $\boldsymbol{h}$ which is initially zero. The idea is to train a neural network which takes $\boldsymbol{h}$ and a letter $\boldsymbol{b}$ as inputs and returns an output $\boldsymbol{y}$ consisting of both the label y and a new state $\boldsymbol{h}'$ concatenated as a vector $\begin{bmatrix} y \\ \boldsymbol{h}' \end{bmatrix}$. I.e.

$$\begin{bmatrix} y \\ \boldsymbol{h}' \end{bmatrix} = \boldsymbol{f}\left(\begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{h} \end{bmatrix}, \boldsymbol{w} \right).$$

This is just a standard feed-forward neural network. We can then apply it to an arbitrary long sequence by first initializing $\boldsymbol{h}^{(1)} = \boldsymbol{0}$ and evaluating

$$\begin{bmatrix} y^{(1)} \\ \boldsymbol{h}^{(1)} \end{bmatrix} = \boldsymbol{f}\left(\begin{bmatrix} \boldsymbol{b}_1 \\ \boldsymbol{h}^{(1)} \end{bmatrix}, \boldsymbol{w} \right)$$

and again for the second digit

$$\begin{bmatrix} y^{(2)} \\ \boldsymbol{h}^{(2)} \end{bmatrix} = \boldsymbol{f}\left(\begin{bmatrix} \boldsymbol{b}_2 \\ \boldsymbol{h}^{(1)} \end{bmatrix}, \boldsymbol{w} \right)$$

and so on until for the nth digit:

$$\begin{bmatrix} y^{(n)} \\ \boldsymbol{h}^{(n)} \end{bmatrix} = \boldsymbol{f}\left(\begin{bmatrix} \boldsymbol{b}_n \\ \boldsymbol{h}^{(n-1)} \end{bmatrix}, \boldsymbol{w} \right).$$

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 $\boldsymbol{w}$ and the input string $\boldsymbol{x}$:

$$y = F(\boldsymbol{x}, \boldsymbol{w}) = f_1\left(\left[\boldsymbol{b}_S \ \boldsymbol{f}\left(\left[\boldsymbol{b}_{S-1} \ \boldsymbol{f}\left(\left[\boldsymbol{b}_{S-2} \cdots \right]^T, \boldsymbol{w} \right) \right]^T, \boldsymbol{w} \right) \right]^T, \boldsymbol{w} \right).$$

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 $\boldsymbol{h}$ 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.

13.4.5 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

13.1. Fall 2013 question 23: 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 $\lambda \boldsymbol{w}^\top \boldsymbol{w}$.
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.

13.2. Fall 2014 question 5: Consider a feedforward neural network shown in fig. 13.6. 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 $n_1, \ldots, n_5$ 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?

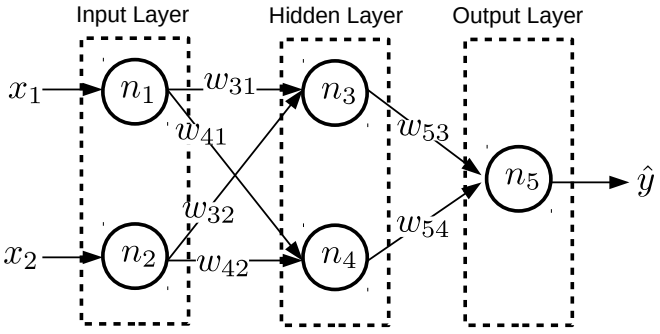


Fig. 13.6: Simple neural network of 6 weights

A $\hat{y} = 0.04$
B $\hat{y} = 0.0$
C $\hat{y} = 1.0$
D $\hat{y} = 0.16$
E Don't know.

13.3. Fall 2013 question 12: Consider the classification problem given in Figure 13.7. 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 13.7 correspond to. Which one of the following statements is *correct*?

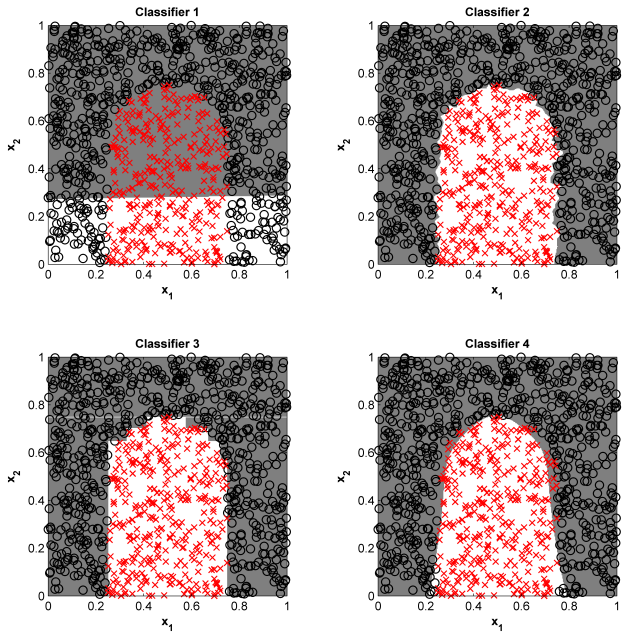


Fig. 13.7: The decision boundaries given in white and gray of four different classifiers used to separate red crosses from black circles.

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.

13.4. Fall 2014 question 19: Consider the classification problem given in fig. 13.9. 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 boundariers of the four plots (P_1, P_2, P_3, P_4) indicated in fig. 13.8?

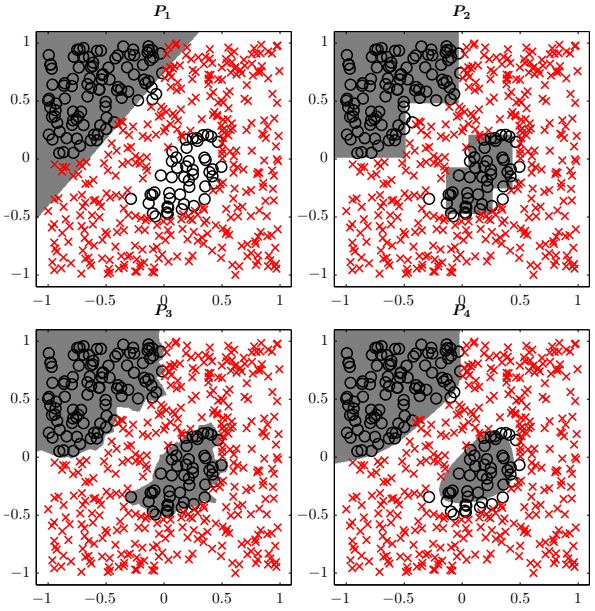


Fig. 13.8: Two-class classification problem

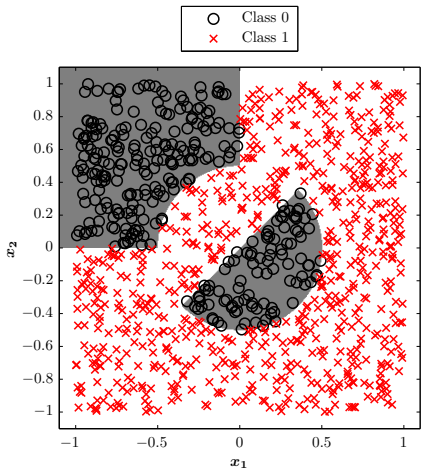


Fig. 13.9: Two-class classification problem

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.

14

Performance evaluation and class imbalance

Class imbalance refers to the situation where the classes in a dataset are not represented equally. The problem 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 Kens test *actually* just consists of a card which says: *"Ebola negative"*, but since there are so few people with Ebola it still obtain an accuracy of roughly:

$$\text{Accuracy of Kens Ebola test} = 1 - \frac{\#\text{Cases of Ebola}}{\#\text{Number of people}} \tag{14.1}$$

$$\approx 1 - \frac{80}{8\,000\,000\,000} = 1 - 10^{-8}. \tag{14.2}$$

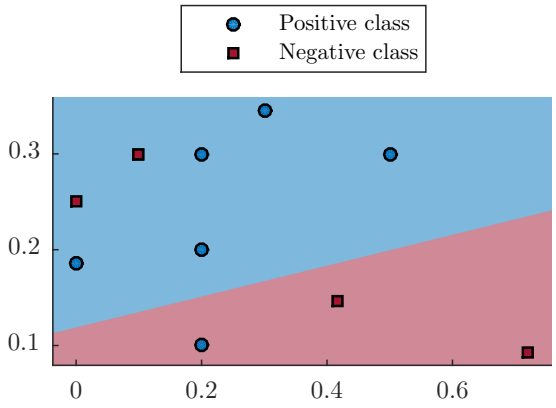
This is probably *the worst* Ebola test imaginable but nobody is ever going to discover it by looking at the accuracy.

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 percent 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].

14.1 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.



$N = 10$	Predicted **Positive**	Predicted **Negative**
Actually **Positive** $N^+ = 6$	TP = 5 (True Positive)	FN = 1 (False Negative)
Actually **Negative** $N^- = 4$	FP = 2 (False Positive)	TN = 2 (True Negative)

Fig. 14.1: (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.
Rethinking the problem: where we try to re-think the objective of the classification task.
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.

14.1.1 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 (randomly) 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 provides 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.

14.1.2 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. 14.1 and which we encountered earlier in section 7.2.1 of chapter 7. 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 $\frac{1000}{2\times1000} + \frac{0}{2\times10} = 50\%$ 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 normal transaction and the negative to a fraudulent transaction. In this case labelling a few good transactions as fraudulent, FN, (transactions that are actually positive labelled negative) is not so bad, but labelling fraudulent transactions as good, FP, correspond to a loss of money. We can therefore consider a general measure of the quality of the classifier of the form

$$w_1\text{TP} + w_2\text{FN} + w_3\text{FP} + w_4\text{TN} \tag{14.3}$$

where $w_1, \cdots, w_4$ are constants. As a crude example, in the credit-card system we could choose $w_1 = 2, w_2 = -1, w_3 = -1000$ 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.

14.1.3 Rethinking the problem

A third type of strategy is to re-think the goal of the classification problem. 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 loose 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 loose by (incorrectly) closing a credit card and annoy a customer and how much do we expect to loose by not closing a credit card in time? This information could in turn be used to select $w_1, \ldots, w_4$ in the penalization scheme eq. (14.3). Keep in mind that especially in medical applications this may lead to fairly uninviting utilitarianism when bad medical decisions are balanced against monetary concerns.

14.2 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 $\hat{y}_i$

$$\hat{y}_i = f(\boldsymbol{x}_i, \boldsymbol{w}).$$

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. 14.2. For instance in logistic

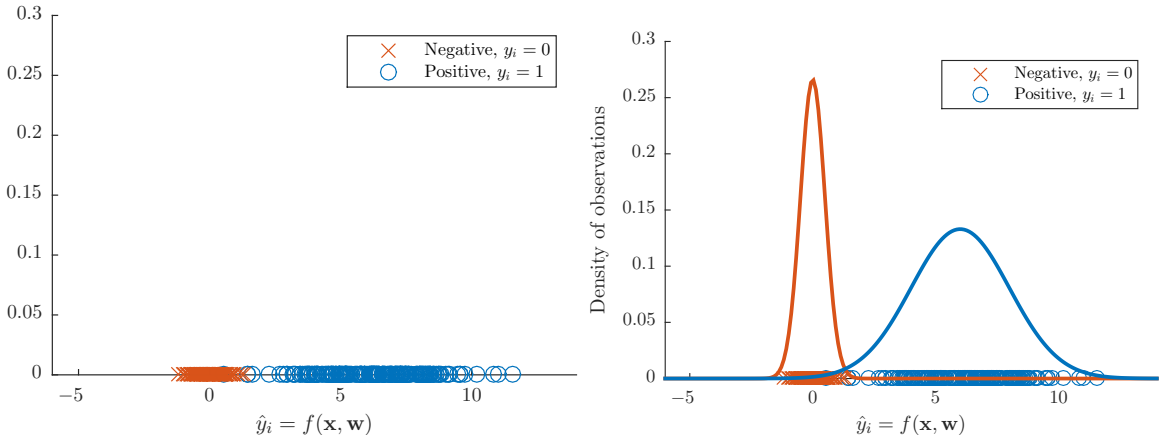


Fig. 14.2: A dataset consisting of a positive $y_i = 1$ class and a negative $y_i = 0$ class. The x-position indicates the predicted y-value by a classification model. In the right plane we have plotted the same dataset but with the density of each class which is easier to visualize and which will be used in the following.

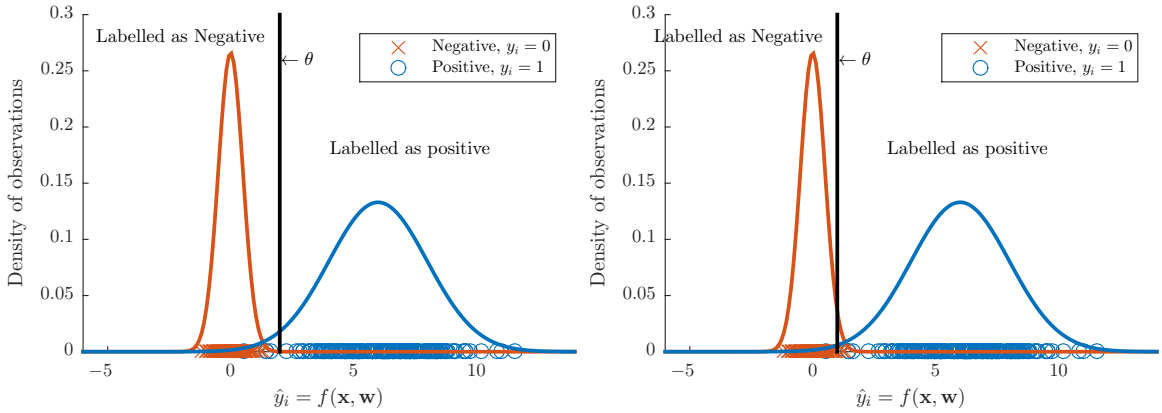


Fig. 14.3: A dataset consisting of a positive $y_i = 1$ class and a negative $y_i = 0$ class. The x-position indicates the predicted y-value by a classification model. In the right plane we have plotted the same dataset but with the density of each class which is easier to visualize and which will be used in the following.

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. 14.3 is shown two different thresholds. Notice, the different thresholds has 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.

14.2.1 The confusion matrix and thresholding

Each thresholding level θ produce a confusion matrix. For convenience this is illustrated in fig. 14.4 and are:

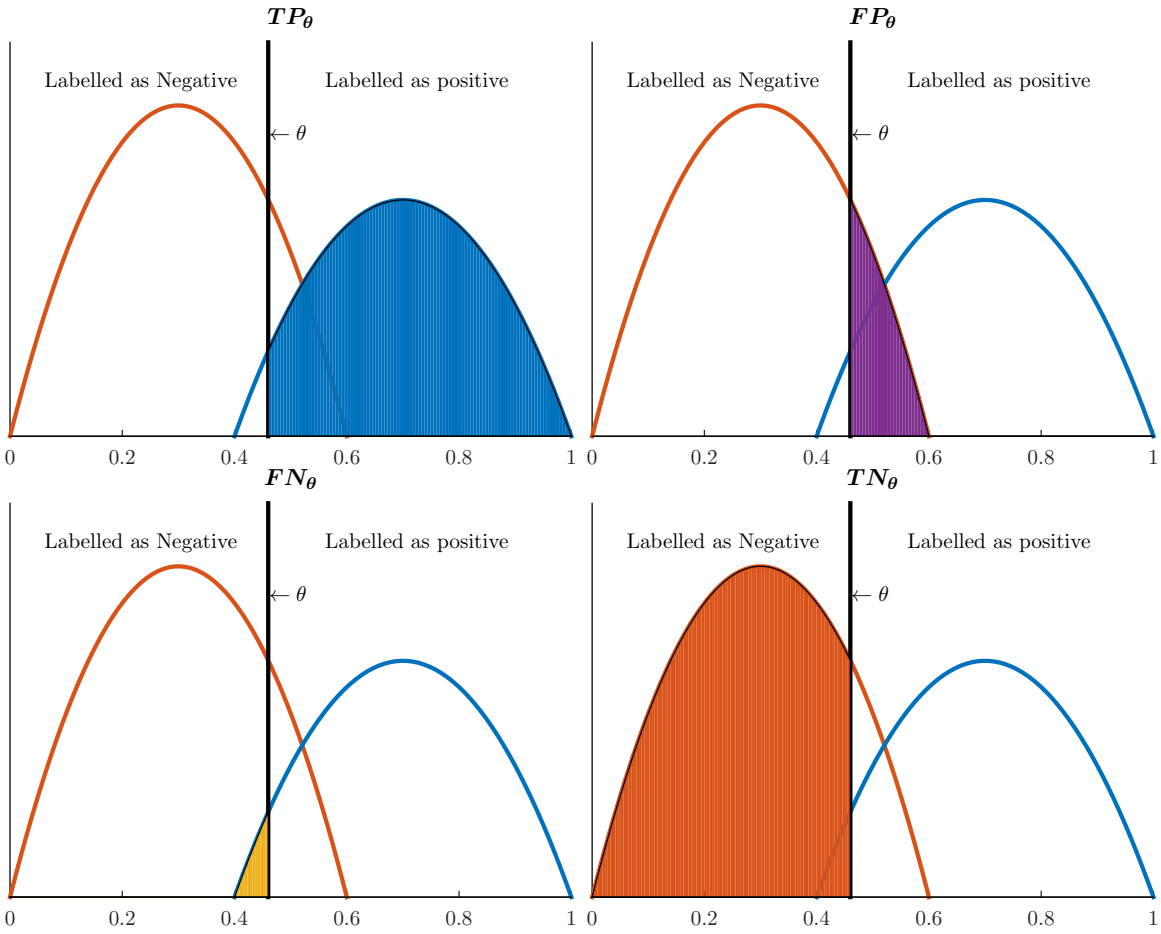


Fig. 14.4: Everything left of the black line θ is labelled to belong to the negative class and everything to the right of the black line is labelled as belonging to the positive class. Clockwise, the areas then indicate the number of true positives, false positives, false negatives and true negatives. See text for details.

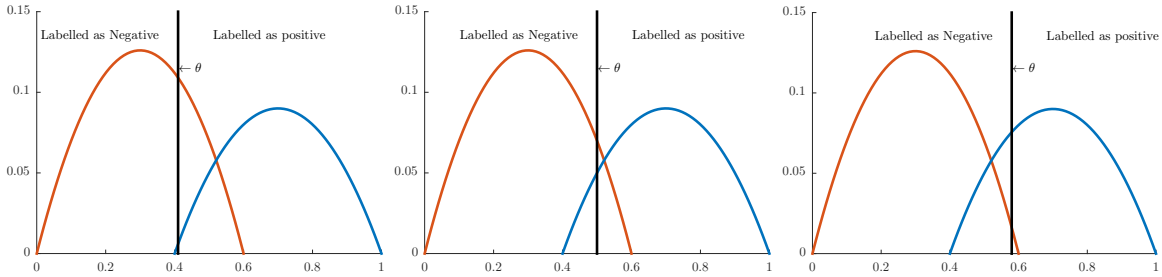


Fig. 14.5: Illustration of our two classifiers with three different threshold values.

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:

$$\mathrm{FPR}_\theta = \frac{\mathrm{FP}_\theta}{\mathrm{FP}_\theta + \mathrm{TN}_\theta}, \tag{14.4}$$

$$\mathrm{TPR}_\theta = \frac{\mathrm{TP}_\theta}{\mathrm{TP}_\theta + \mathrm{FN}_\theta}, \tag{14.5}$$

where by definition $\mathrm{TP}_\theta + \mathrm{FN}_\theta$ is the total number of positive examples and $\mathrm{FP}_\theta + \mathrm{TN}_\theta$ is the total number of negative examples. An illustration of how these numbers depend on θ is properly in order. In fig. 14.5 is illustrated three values of the threshold θ. In fig. 14.6 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. 14.4. Lets 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 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, however, 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 $(\mathrm{FPR}_\theta, \mathrm{TPR}_\theta)$ for each value of these as is done in the right-pane of fig. 14.6. This is our most important diagram. Notice that 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* as simply the area under the curve

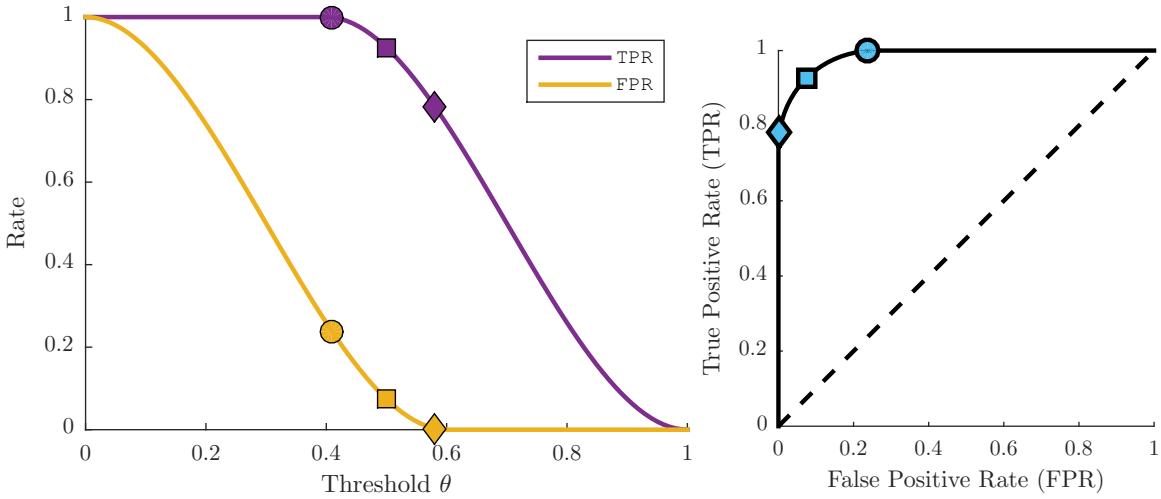


Fig. 14.6: (left:) Illustration of the TPR and FPR curves for different thresholds for the classifier indicated in fig. 14.5. The solid points corresponds to the three specific threshold values indicated. (right:) Illustration of the AUC; the AUC is simply the area under the curve obtained by plotting TPR_θ against FPR_θ for different values of θ. High AUC is good.

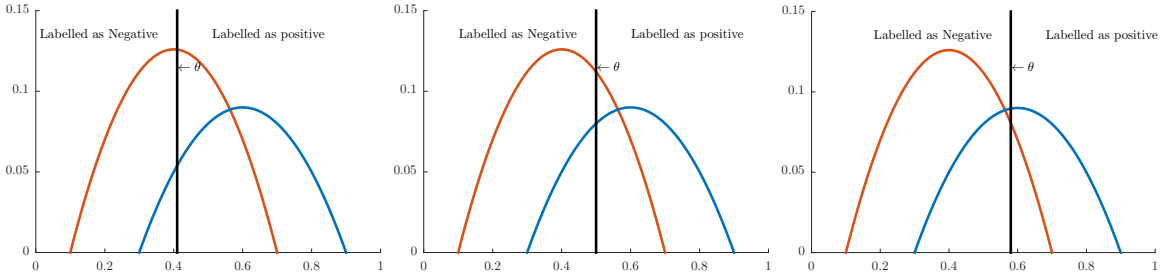


Fig. 14.7: Illustration of three different threshold values for an inferior classifier. The classifier is inferior since the two predicted classes overlap such that no single threshold can tell them apart.

shown in the right-hand side of fig. 14.6, for instance obtained by numerical integration. Since the curves are generally above the dotted line, this value will be between 0.5 and 1.

Lets 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, lets suppose we have an inferior classifier as indicated in fig. 14.7, again with three values of θ selected. The corresponding plot of the TPR and FPR and AUC is given in fig. 14.8. 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. Secondly,

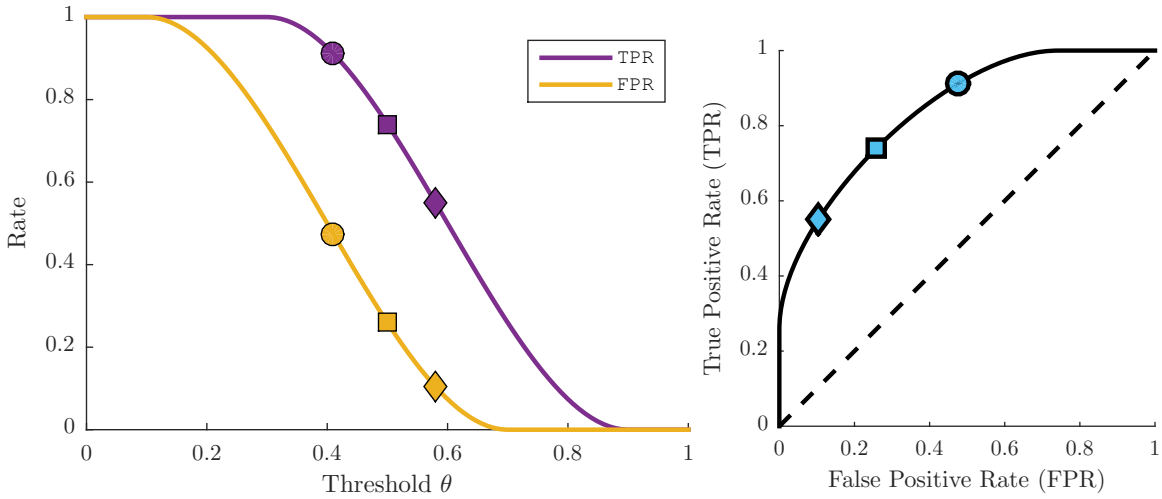


Fig. 14.8: (left:) Illustration of the TPR and FPR curves for different thresholds for the inferior classifier indicated in fig. 14.7. The solid points corresponds to the three specific threshold values indicated. (right:) Illustration of the AUC; the AUC is simply the area under the curve obtained by 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. 14.5

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

14.1. Fall 2014 question 24: 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. 14.10. Which of the true positive rate (TPR), and false positive rate (FPR) curves A,B,C or D shown in fig. 14.9 corresponds to the ROC curve in fig. 14.10?

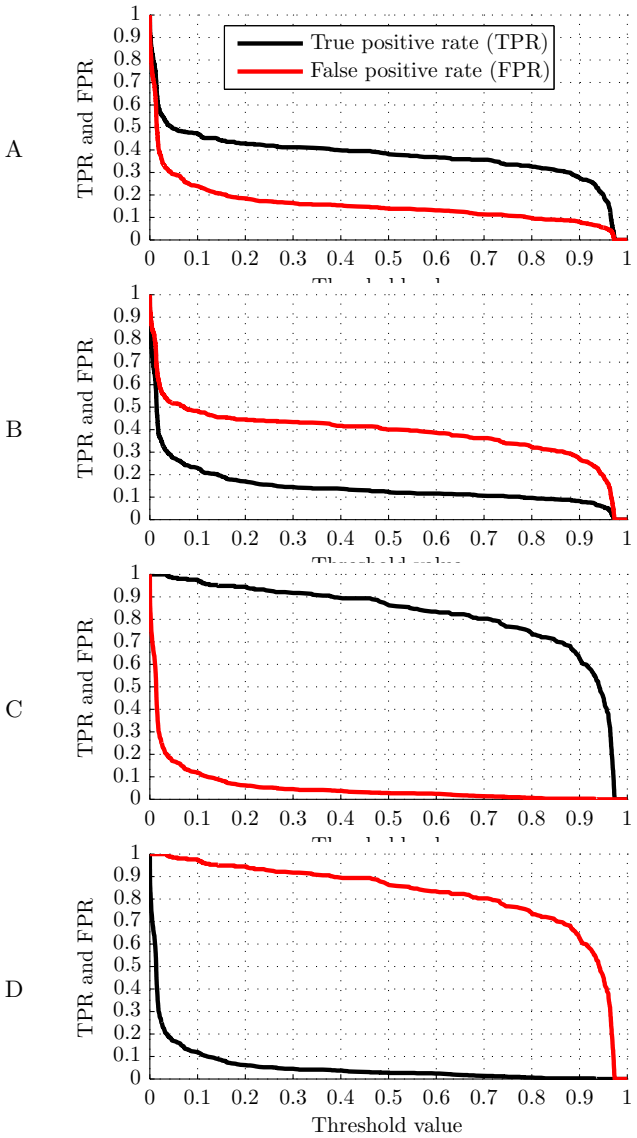


Fig. 14.9: Proposed values of the TPR and FPR, as calculated for $N = 1000$ predicted values, for different values of the threshold θ.

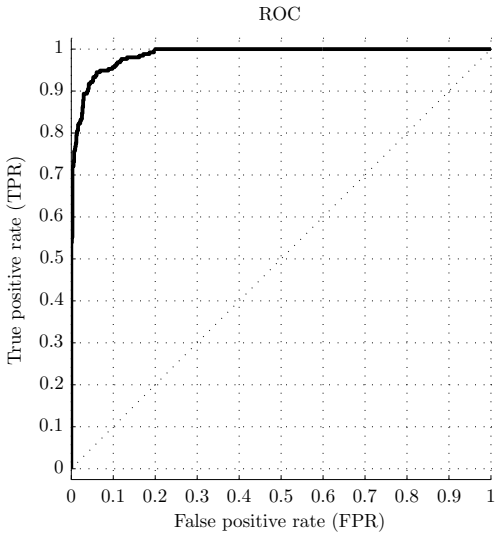


Fig. 14.10: ROC curve of a neural network.

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

14.2. Fall 2014 question 25: 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. 14.9(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.

15

Ensemble methods

Ensemble methods are methods where multiple models are trained and their output is 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 opinion 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].

15.1 Introduction to ensemble methods

The basic idea of ensemble methods is very simple: train multiple models and combine their output into a single model as illustrated in fig. 15.1. Let's consider how this combination takes place before we discuss how the ensemble of models is produced: Suppose $\mathcal{M}_1, \ldots, \mathcal{M}_T$ are T regression models and model $\mathcal{M}_t$ is trained on some data $\mathcal{D}$ to learn a regression function $y = f_t(\boldsymbol{x})$, we can then define a new model $\mathcal{M}^*$ as simply the average:

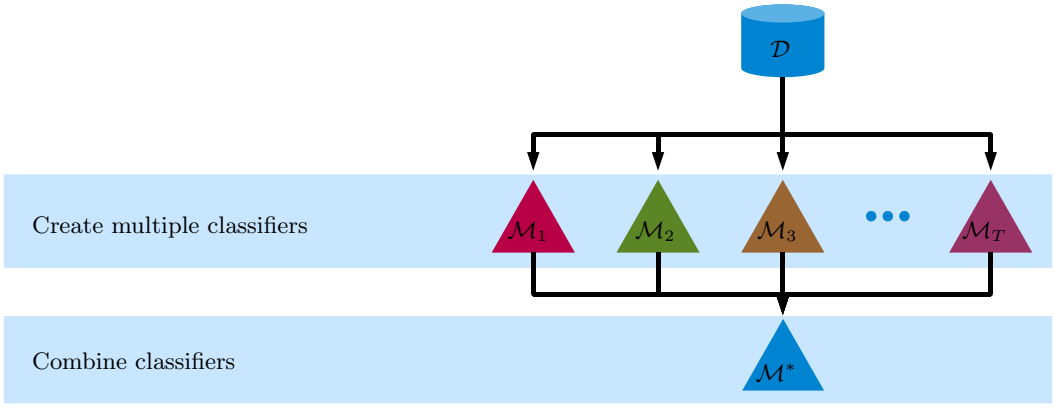


Fig. 15.1: Combining T models $\mathcal{M}_1, \ldots, \mathcal{M}_T$ to a single classifier $\mathcal{M}^*$ using majority voting (classification) or averaging (regression) is often a useful strategy to come up with a classifier which outperforms each individual model.

$$\textit{(Regression:)} \quad y = \frac{1}{T}\sum_{t=1}^{T} f_t(\boldsymbol{x}). \tag{15.1}$$

fig. 15.1 Alternatively, suppose $\mathcal{M}_1, \ldots, \mathcal{M}_T$ are classifiers, i.e. $f(\boldsymbol{x}) = y = 1, \ldots, C$, and TODO-ONE-OF-K We can then combine their output by letting each classifier "vote" for an output and then select the class which most classifiers agree is the correct one.

$$\textit{(Classification:)} \quad f(\boldsymbol{x}) = \underset{c=1,\ldots,C}{\arg\max}\{\text{Number of classifiers which output } f_t(\boldsymbol{x}) = c\}, \tag{15.2}$$

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, supposing T is uneven,

$$\begin{aligned} P(\text{Majority voting is correct}) &= \sum_{t=(T+1)/2}^{T} \{t \text{ of the classifiers are correct}\} \\ &= \sum_{t=(T+1)/2}^{T} \binom{T}{t} p^t (1-p)^{T-t}, \end{aligned}$$

the graph is plotted in fig. 15.2. 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. 15.3. There are essentially two strategies:

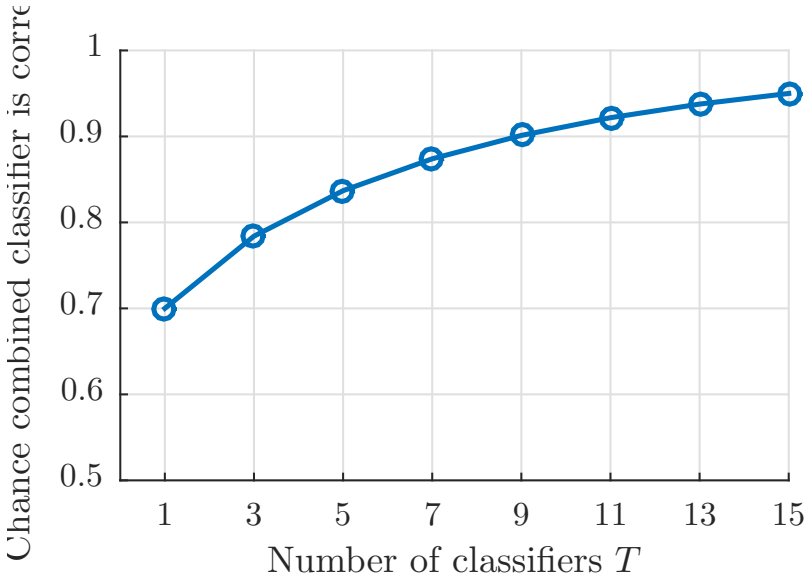


Fig. 15.2: If T independent classifiers is combined using majority voting, each with an accuracy of only $p = 0.7$, the accuracy of the resulting classifier quickly approaches 1. In practice, it is difficult to find independent classifiers, however, the picture still holds approximately for dependent 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 there known as *random forests* which we will consider in section 15.3. 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.

15.2 Bagging

Bagging begins with a dataset $\mathcal{D}$ of size N, and then randomly selects T new datasets $\mathcal{D}_1, \ldots, \mathcal{D}_T$ 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 the T datasets T times to produce T different classifiers which are then combined into a single classifier using eq. (15.1) or eq. (15.2). This procedure is illustrated in fig. 15.3 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.

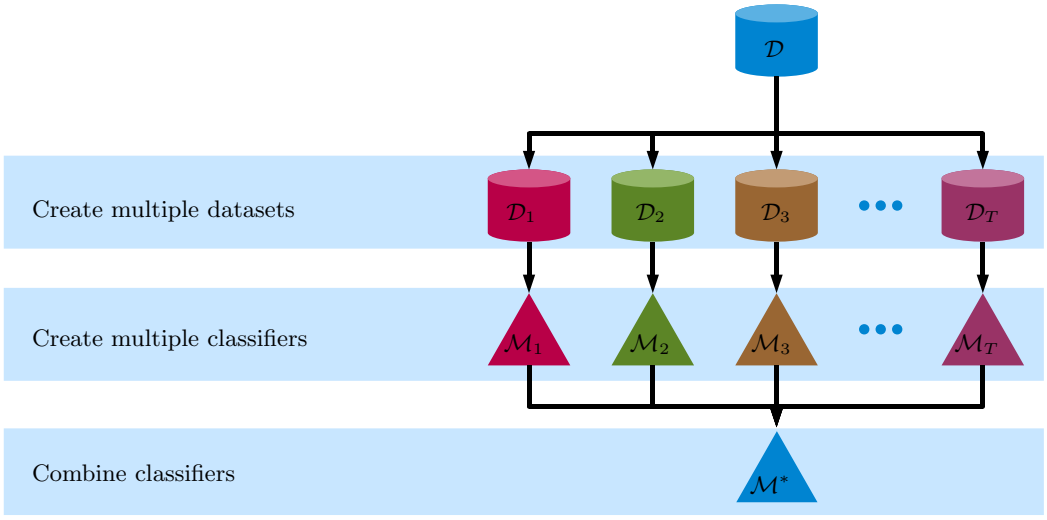


Fig. 15.3: A different strategy for obtaining multiple classifiers is to create T new datasets $\mathcal{D}_1, \ldots, \mathcal{D}_T$ from the training dataset $\mathcal{D}$ and train classifiers to each of the dataset. The T obtained classifiers can then be combined as in fig. 15.1.

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. 15.4 (left). The dataset is fitted with a standard logistic regression model giving the linear decision boundary $p(y|\boldsymbol{x}, \mathcal{D})$ shown in the middle pane. Since we are only interested in the class labels for the majority-voting scheme eq. (15.2) we will assume the predictions of the logistic regression model is threshold at 0.5 to produce the decision boundary shown in the right pane.

In fig. 15.5 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 consists of few observations. In fig. 15.6 the bagging classifiers are combined, i.e. for each point $\boldsymbol{x}$ we plot the bagged classifiers predictions:

$$y = \frac{1}{T} \sum_{t=1}^{T} f_t(\boldsymbol{x}), \tag{15.3}$$

and the black line corresponds to $y > \frac{1}{2}$ corresponding to majority voting eq. (15.2). As seen from the figure, each single classifier is worse than the classifier which used all data shown in fig. 15.4 (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. 15.6 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

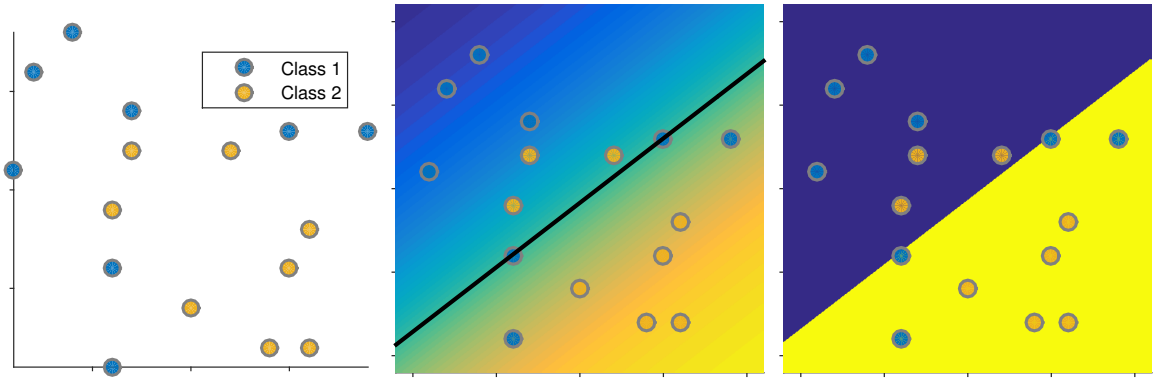


Fig. 15.4: (left:) A simple 2D classification problem with two classes and two features. (middle:) A logistic regression model is fitted to the data to give the class-probability indicated with the colors. (right:) thresholding the logistic regression output at 0.5 gives the classification boundary indicated by the colors. This is the decision rule of the classifier.

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 15.3 we will look at a technique for creating a diverse class of classifiers by manipulating the tree-learning method.

15.3 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 produce T datasets (by sampling with replacement) from the original dataset $(\boldsymbol{X}, \boldsymbol{y})$, then train the standard decision tree algorithm on each sampled dataset to produce a predictor $f_t(\boldsymbol{x})$ for $t = 1, \ldots, T$ and finally combine the predictors using either eq. (15.1) (regression) or eq. (15.1) (regression). 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].

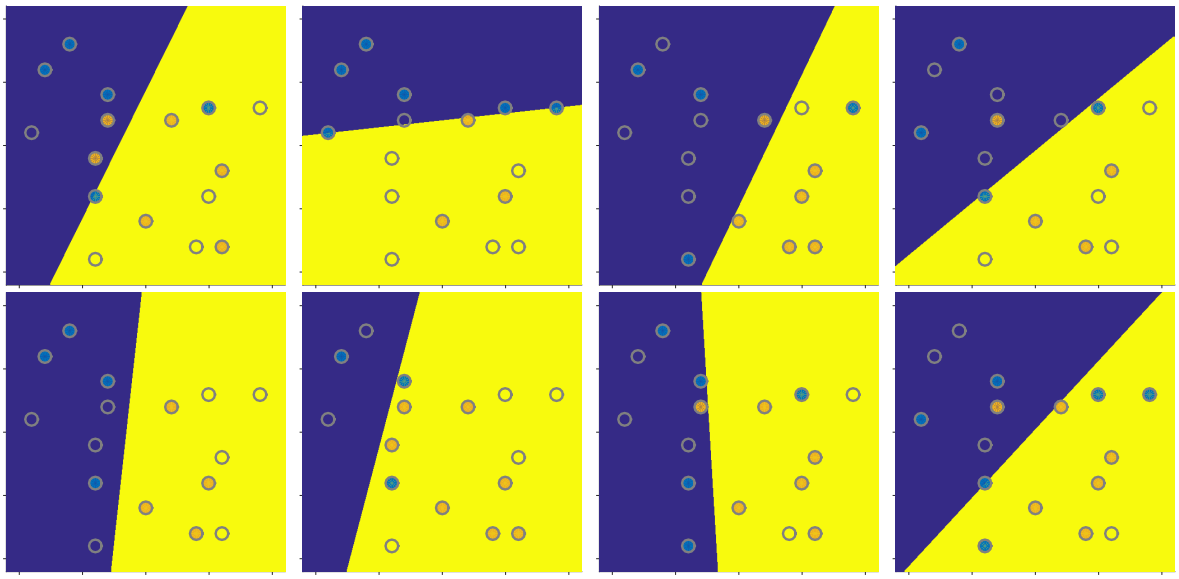


Fig. 15.5: Example of bagging for the classification problem in fig. 15.4. In each pane, a new training set $\mathcal{D}_t$ is obtained by sampling N points with replacement from $\mathcal{D}$ and a logistic regression model is fitted. Notice not all observations are selected and some points may be selected multiple times. Observations not selected are indicated by hollow circles.

15.4 Boosting

As we saw bagging produced some non-linearity in the decision surface, 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 $\boldsymbol{x}_i$ 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. 15.7 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 is increased and the weights of the correctly classified

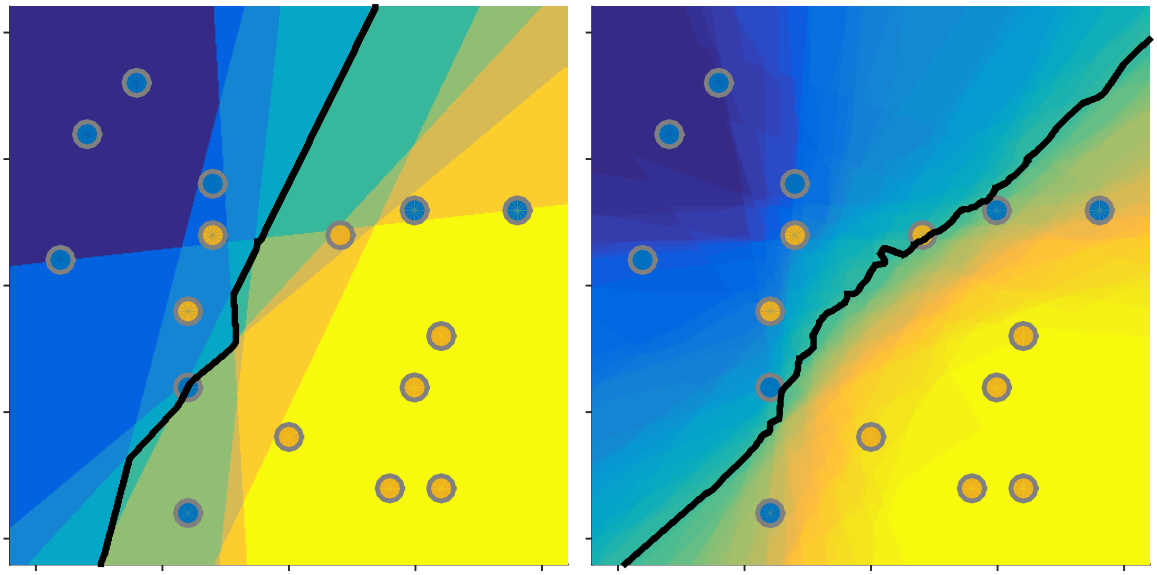


Fig. 15.6: (left:) By averaging the individual prediction boundaries from fig. 15.5 using eq. (15.3) we can define the majority voting rule. The resulting classification boundary (i.e. thresholding at 0.5) is indicated by the black line. In the right pane the same construction is shown but for $T = 100$ datasets. Notice, the resulting rule is smoother and still slightly non-linear.

points is decreased. The weights still sum to 1; this is indicated by the size of the points in the third pane of fig. 15.7.

- Finally, a new dataset $\mathcal{D}_2$ is selected by randomly sampling according to the new weights and the procedure is repeated for dataset $\mathcal{D}_3$ 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 ones intuition, however the weight-updating problem can be analyzed from decision theory which has led to the AdaBoost algorithm [Freund and Schapire, 1997].

15.4.1 AdaBoost

Suppose we denote by $\boldsymbol{w}(t)$ the weight of the observations at step t which determines how likely that observation is to be included in the training set. The AdaBoost algoithm then produces T classifiers $f_1(\boldsymbol{x}), \cdots, f_t(\boldsymbol{x})$ and *importance weights* $\alpha_1, \cdots, \alpha_T$ (which determines how important each classifier is) which are combined to produce the output of the method:[1]

[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: $\sum_{i:y_t(\boldsymbol{x}_i)=1} \alpha_i$ (and similarly for the negative class, $a^- = \sum_{i:y_t(\boldsymbol{x}_i)=0} \alpha_i$) and then output 1 if $\alpha^+ > \alpha^-$ and otherwise 0

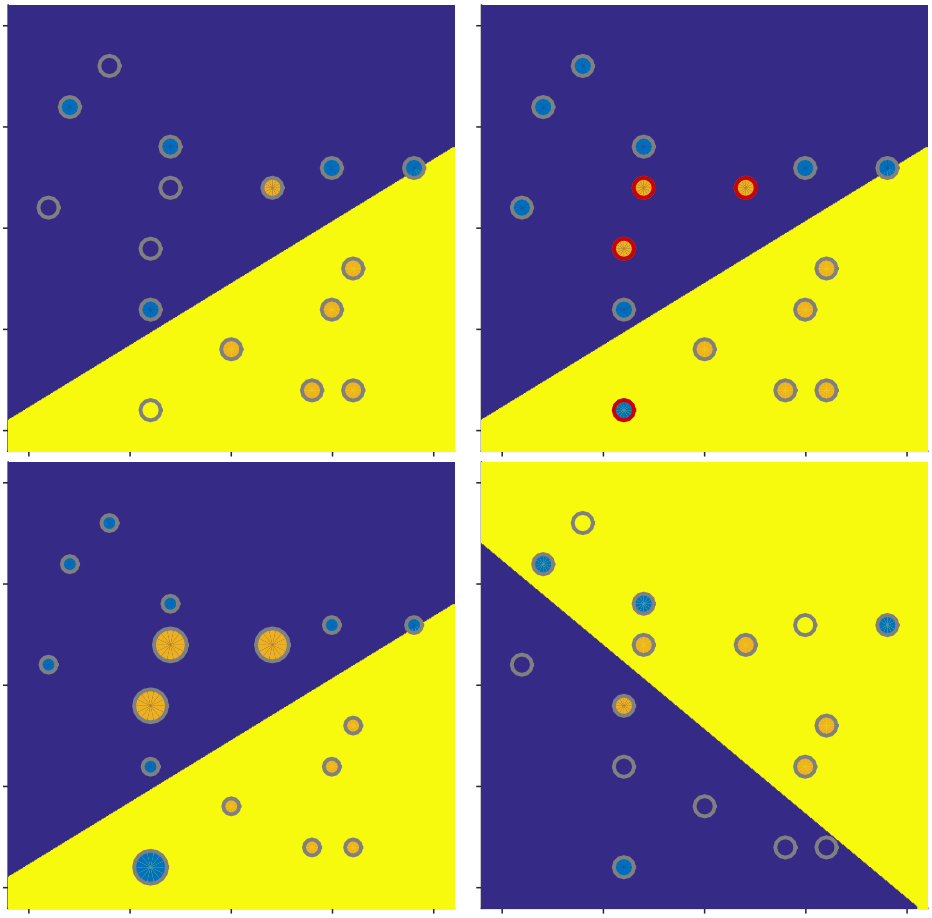


Fig. 15.7: Illustration of a boosting sweep. (Pane 1:) a dataset $\mathcal{D}_t$ is selected by random sampling from $\mathcal{D}$ with replacement but with probability w_i of selecting observation i and a logistic regression model is fitted to the dataset. Points not selected are hollow. (Pane 2:) *all* points are classified using the trained classifier and the misclassified observations are shown in red. (Pane 3:) the weights w_i corresponding to the red misclassified points are increases and the rest are decreased (indicated by the size). (Pane 4:) The next dataset is selected by random sampling using the *new* weights and the procedure is repeated.

$$f^*(\boldsymbol{x}) = \arg\max_{y=1,2} \sum_{t=1}^{T} \alpha_t \delta_{f_t(\boldsymbol{x}),y}. \tag{15.4}$$

The AdaBoost algorithm updates $\boldsymbol{w}(t)$ and α_t by first computing the weighted error:

$$\epsilon_t = \sum_{i=1}^{N} w_i(t) \left(1 - \delta_{f_t(\boldsymbol{x}_i),y_i}\right) \tag{15.5}$$

The importance of the classifier at step t is then computed as:

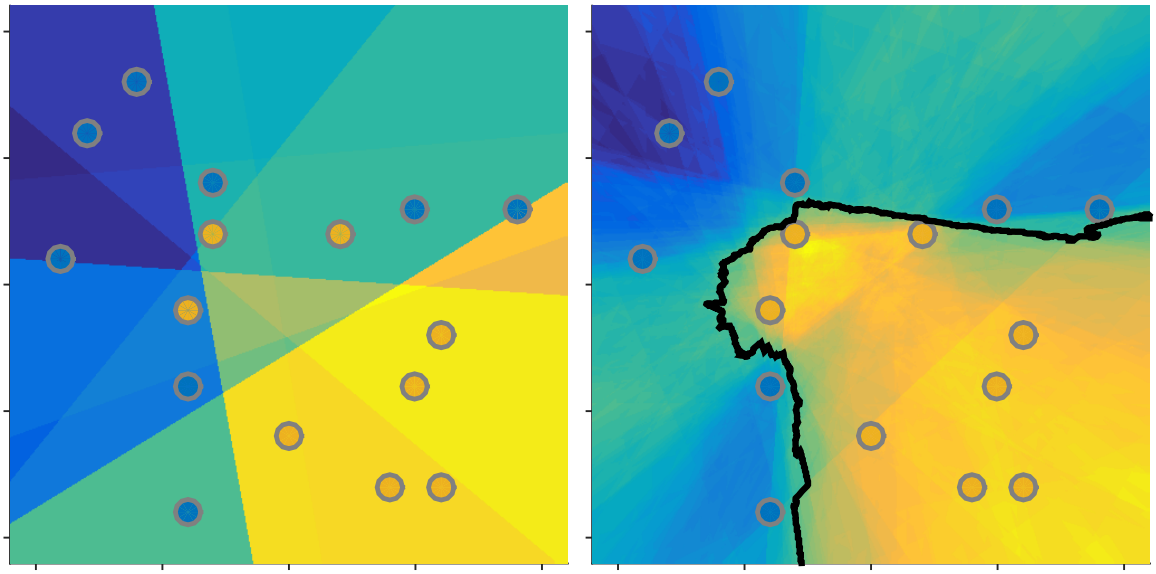


Fig. 15.8: The (importance-weighted) decision function eq. (15.4) when Boosting is applied for $T = 10$ (left) and $T = 500$ (right) rounds, The decision boundary is indicated by the black line. Notice, the decision boundary is highly non-linear and perfectly fit the training data even though each classifier is linear.

$$\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t} \tag{15.6}$$

and finally the new weights $\boldsymbol{w}(t+1)$ are updated by computing:

$$w_i(t+1) = \frac{\tilde{w}_i(t+1)}{\sum_{j=1}^{N} \tilde{w}_j(t+1)} \tag{15.7}$$

$$\text{where } \tilde{w}_j(t+1) = \begin{cases} w_j(t) e^{-\alpha_t} & \text{if } f_t(\boldsymbol{x}_i) = y_i \\ w_j(t) e^{\alpha_t} & \text{if } f_t(\boldsymbol{x}_i) \neq y_i. \end{cases}$$

Thus, this mechanism either up or downscale 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:

$$f^*(\boldsymbol{x}) = \arg\max_{c=1,2} \sum_{t=1}^{T} \alpha_t \delta_{f_t(\boldsymbol{x}),y}.$$

The full AdaBoost procedure can be seen in algorithm 6 and in fig. 15.8 we have plotted importance-weighted decision function.

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 6: AdaBoost algorithm

1: Initialize $w_i(1) = \frac{1}{N}$ for $i = 1, \ldots, N$
2: **for** $t = 1, \ldots, T$ **do**
3: Create $\mathcal{D}_t$ by sampling (with replacement) from $\mathcal{D}$ according to $\boldsymbol{w}(t)$
4: Let f_t be the classifier *trained* on $\mathcal{D}_t$
5: $\epsilon_t = \sum_{i=1}^{N} w_i \left(1 - \delta_{f_t(\boldsymbol{x}_i), y_i}\right)$ *(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. (15.7):

$$w_i(t+1) = \frac{\tilde{w}_i(t+1)}{\sum_{j=1}^{N} \tilde{w}_j(t+1)}, \quad \tilde{w}_j(t+1) = \begin{cases} w_j(t)e^{-\alpha_t} & \text{if } f_t(\boldsymbol{x}_i) = y_i \\ w_j(t)e^{\alpha_t} & \text{if } f_t(\boldsymbol{x}_i) \neq y_i. \end{cases}$$

8: **end for**
9: $f^*(\boldsymbol{x}) = \arg\max_{y=1,2} \sum_{t=1}^{T} \alpha_t \delta_{f_t(\boldsymbol{x}), y}$ *(Majority voting classifier)*

15.4.2 *Properties of the AdaBoost algorithm

As mentioned, the peculiar form of the update rules for α_t and $\boldsymbol{w}(t)$ 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

$$\epsilon^* \leq \prod_{t=1}^{T} \sqrt{\epsilon_t(1-\epsilon_t)},$$

where ϵ_t are the error rates of each of the classifiers as described in algorithm 6. 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:

$$\epsilon^* \leq \prod_{t=1}^{T} \sqrt{\epsilon_t(1-\epsilon_t)} = \prod_{t=1}^{T} \sqrt{1-4\gamma_t^2} \leq e^{-2\sum_{t=1}^{T} \gamma_t^2}.$$

Consequently, if all $\gamma_t \leq \gamma_0$ 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 9 a low *training* error is not in itself a good sign. Theoretical analysis of AdaBoost reveals that with high probability: [Freund and Schapire, 1997]

$$\text{Text error} \leq \text{Text error} + \mathcal{O}\left(\sqrt{\frac{dT}{N}}\right),$$

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. 15.8. 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

15.1. Fall 2014 question 26: Suppose Jane wishes to apply a decision tree classifier to a binary classification problem of only $N = 4$ observations. Training and applying the decicion tree to the full dataset $\boldsymbol{X}$ and $y_1, \ldots, y_4$ gives predictions $\hat{y}_1, \ldots, \hat{y}_4$ shown in table 15.1.

y	$\hat{y}$
1	1
1	0
0	0
0	0

Table 15.1: True values y_j and predictions $\hat{y}_j$ for a decision tree classifier trained on the full data set with observed values $y_1, \ldots, y_4$.

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 *without* 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 $w = \begin{bmatrix}0.123 & 0.630 & 0.123 & 0.123\end{bmatrix}$
B $w = \begin{bmatrix}0.167 & 0.5 & 0.167 & 0.167\end{bmatrix}$
C $w = \begin{bmatrix}0.081 & 0.756 & 0.081 & 0.081\end{bmatrix}$
D $w = \begin{bmatrix}0.077 & 0.769 & 0.077 & 0.077\end{bmatrix}$
E Don't know.

15.2. Fall 2013 question 24: 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.

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 3.14 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 substracting the mean from $\boldsymbol{X}$ and projecting onto the first two columns of $\boldsymbol{V}$ (see appendix B of Tan et al). The first point with the mean subtracted has coordinates

$$\begin{bmatrix} 3-7/3 & 2-4/3 & 1-5/3 \end{bmatrix}$$

This should be (left) multiplied with the first two columns of $\boldsymbol{V}$:

$$\begin{bmatrix} 3-7/3 \\ 2-4/3 \\ 1-5/3 \end{bmatrix}^\top \begin{bmatrix} -0.99 & -0.13 \\ -0.09 & 0.7 \\ 0.09 & -0.7 \end{bmatrix} = \begin{bmatrix} -0.78 & 0.85 \end{bmatrix}$$

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

$$\text{SMC}(o_1, o_3) = \frac{f_{11} + f_{00}}{M} = \frac{M - f_{01} - f_{10}}{M} = \frac{8}{15} = 0.53$$

The other options are not true, on the simulated dataset the correct options are: $\text{COS}(o_1, o_3) = 0.565685424949$, $\text{J}(o_1, o_3) = 0.363636363636$

4.2 The correct answer is D: The three measures of similarity is given by

$$\begin{aligned} J(\boldsymbol{x}, \boldsymbol{y}) &= \frac{f_{11}}{f_{01} + f_{10} + f_{11}} \\ SMC(\boldsymbol{x}, \boldsymbol{y}) &= \frac{f_{00} + f_{11}}{f_{01} + f_{10} + f_{11} + f_{00}} \\ cos(\boldsymbol{x}, \boldsymbol{y}) &= \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|} \end{aligned}$$

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

$$\begin{aligned} J(\boldsymbol{x}, \boldsymbol{y}) &= \frac{f_{11}}{f_{01} + f_{10} + f_{11}} \\ SMC(\boldsymbol{x}, \boldsymbol{y}) &= \frac{f + f_{11}}{f_{01} + f_{10} + f_{11} + f} \\ cos(\boldsymbol{x}, \boldsymbol{y}) &= \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|} \end{aligned}$$

We now have

$$\begin{aligned} J(\mathrm{S1},\mathrm{S2}) &= \frac{2}{4} = \frac{1}{2}, \\ J(\mathrm{S1},\mathrm{NS1}) &= \frac{2}{4} = \frac{1}{2}, \\ SMC(\mathrm{S1},\mathrm{S2}) &= \frac{4}{6} = \frac{2}{3}, \\ SMC(\mathrm{S1},\mathrm{NS1}) &= \frac{4}{6} = \frac{2}{3}, \\ cos(\mathrm{S1},\mathrm{S2}) &= \frac{2}{\sqrt{3}\sqrt{3}} = \frac{2}{3}, \\ cos(\mathrm{S1},\mathrm{NS1}) &= \frac{2}{\sqrt{3}\sqrt{3}} = \frac{2}{3}. \end{aligned}$$

Hence, the correct answer is $SMC(S1, S2) = cos(S1, S2)$.

Problems of Chapter 5

5.1 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 $p(B|H) = 1 - 0.2 = 0.8$

5.2 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:

$$\begin{aligned} P(NS|CD_Y) &= \tfrac{P(CD_Y|NS)P(NS)}{P(CD_Y)} \\ &= \tfrac{P(CD_Y|NS)P(NS)}{P(CD_Y|NS)P(NS)+P(CD_Y|AS)P(AS)} \\ &= \tfrac{0.125\cdot 0.88}{0.125\cdot 0.88+0.167\cdot 0.12} \\ &= 0.8459 \end{aligned}$$

5.3 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 6

6.1 The correct answer is B: The 25th and 50th percentile but not the 50th and 75th percentiles of the attribute DB coincides. AlA 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.

6.2 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.

6.3 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 7

7.1 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.

7.2 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.

7.3 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 7.10 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 7.10 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.

7.4 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:

$$p = \frac{12}{12+10} = \frac{6}{11}, \tag{15.8}$$

$$r = \frac{12}{12+69} = \frac{12}{81}, \tag{15.9}$$

$$TNR = \frac{215}{215+10} = \frac{43}{45}, \tag{15.10}$$

$$FPR = \frac{10}{215+10} = \frac{2}{45}. \tag{15.11}$$

Decision tree:

$$p = \frac{26}{26+34} = \frac{13}{30}, \tag{15.12}$$

$$r = \frac{26}{26+55} = \frac{26}{81}, \tag{15.13}$$

$$TNR = \frac{191}{191+34} = \frac{191}{225}, \tag{15.14}$$

$$FPR = \frac{34}{191+34} = \frac{34}{225}. \tag{15.15}$$

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.

7.5 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:

$$A, D : \frac{1}{1+e^{-w_0}} = (1+e^{-2})^{-1} = 0.88$$
$$B, C : \frac{1}{1+e^{-w_0}} = (1+e^{-(-2)})^{-1} = 0.12$$

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 8

8.1 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:

$$\begin{aligned}\Delta =& (1 - ((\tfrac{81}{306})^2 + (\tfrac{225}{306})^2)) \\ & -[\tfrac{170}{306}(1 - ((\tfrac{62}{170})^2 + (\tfrac{108}{170})^2)) \\ & +\tfrac{136}{306}(1 - ((\tfrac{19}{136})^2 + (\tfrac{117}{136})^2))] = 0.025.\end{aligned}$$

8.2 The correct answer is A: The two classes would be well separagted by the decisions
A=$\| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.25 \end{bmatrix} \|_\infty < 0.25$
B=$\| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \|_2 < 0.25.$
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.

8.3 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 $\Delta \approx 0.143$.

8.4 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 of" 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 9

9.1 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

$$T_{cv} = LK(D^2_{cv-train} + 1/2D^2_{cv-test})$$

This gives us the optimal value of λ. To estimate the performance we need to test and train one final model. This takes:

$$T_{val} = (D^2_{cv} + 1/2D^2_{val})$$

The total time elapsed is then

$$T = T_{cv} + T_{val} = 31.956 \cdot 10^6.$$

9.2 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.

9.3 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.

9.4 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.

9.5 The correct answer is B: The error rate and accuracy for each of the classifiers is given by:
Logistic regression:
error rate=$\frac{10+69}{306} = \frac{79}{306}$,
accuracy=$\frac{215+12}{306} = \frac{227}{306}$
Decision tree:
error rate=$\frac{34+55}{306} = \frac{89}{306}$,
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.

9.6 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 $\theta = 0.45$

9.7 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.

9.8 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 10

10.1 The correct answer is B: O1-04 are all closet 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.

10.2 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.

10.3 The correct answer is A: The true accuracy is 0.125 or 1/8. This is easy to see by going through table 10.2 and observing all observations except o_1 is misclassified

Problems of Chapter 11

11.1 The correct answer is B: According to the Naïve Bayes classifier we have

$$P(CKD = 1|RBC = 1, PC = 1, DM = 1, CAD = 1) =$$
$$\frac{\begin{pmatrix} P(RBC = 1|CKD = 1)\times \\ P(PC = 1|CKD = 1)\times \\ P(DM = 1|CKD = 1)\times \\ P(CAD = 1|CKD = 1)\times \\ P(CKD = 1) \end{pmatrix}}{\begin{pmatrix} P(RBC = 1|CKD = 1)\times \\ P(PC = 1|CKD = 1)\times \\ P(DM = 1|CKD = 1)\times \\ P(CAD = 1|CKD = 1)\times \\ P(CKD = 1) \end{pmatrix} + \begin{pmatrix} P(RBC = 1|CKD = 0)\times \\ P(PC = 1|CKD = 0)\times \\ P(DM = 1|CKD = 0)\times \\ P(CAD = 1|CKD = 0)\times \\ P(CKD = 0) \end{pmatrix}}$$
$$= \frac{2/9 \cdot 7/9 \cdot 6/9 \cdot 1/9 \cdot 9/15}{2/9 \cdot 7/9 \cdot 6/9 \cdot 1/9 \cdot 9/15 + 1/6 \cdot 1/6 \cdot 1/6 \cdot 1/6 \cdot 6/15} = 0.9614.$$

11.2 The correct answer is D: According to the Bayes classifier we have

$$
\begin{aligned}
&P(CKD=1|RBC=1,PC=1,DM=1) = \\
&\frac{\begin{pmatrix} P(RBC=1,PC=1,DM=1|CKD=1)\times \\ P(CKD=1) \end{pmatrix}}{\begin{pmatrix} P(PRBC=1,PC=1,DM=1|CKD=1)\times \\ P(CKD=1) \end{pmatrix} + \begin{pmatrix} P(RBC=1,PC=1,DM=1|CKD=0)\times \\ P(CKD=0) \end{pmatrix}} \\
&= \frac{2/9 \cdot 9/15}{2/9 \cdot 9/15 + 0/6 \cdot 6/15} = 1
\end{aligned}
$$

11.3 The correct answer is D: According to the Naïve Bayes classifier we have

$$
\begin{aligned}
&P(S|YA_Y=1,OA_Y=1,PA_Y=1) = \\
&\frac{\begin{pmatrix} P(YA_Y=1=1|S)\times \\ P(OA_Y=1|S)\times \\ P(PA_Y=1|S)\times \\ P(S) \end{pmatrix}}{\begin{pmatrix} P(YA_Y=1=1|S)\times \\ P(OA_Y=1|S)\times \\ P(PA_Y=1|S)\times \\ P(S) \end{pmatrix} + \begin{pmatrix} P(YA_Y=1=1|NS)\times \\ P(OA_Y=1|NS)\times \\ P(PA_Y=1|NS)\times \\ P(NS) \end{pmatrix}} \\
&= \frac{2/5 \cdot 4/5 \cdot 4/5 \cdot 5/10}{2/5 \cdot 4/5 \cdot 4/5 \cdot 5/10 + 1/5 \cdot 3/5 \cdot 4/5 \cdot 5/10} = \frac{8}{11}.
\end{aligned}
$$

11.4 The correct answer is A: True answer is: 0.83. This can be found by computing the per-class probabilities

$$
\begin{aligned}
&p(f_1=0|C_1) = 3/5,\ p(f_2=1|C_1) = 1 \\
&p(f_1=0|C_2) = 1/5,\ p(f_2=1|C_2) = 3/5
\end{aligned}
$$

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 12

12.1 The correct answer is B: The aim of regularized leats squares regression is to reduce the model's bias without introducing too much variance 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 13

13.1 The correct answer is C: The aim of regularized leats 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 minimize 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.

13.2 The correct answer is A: To compute the output, initialize $n_1 = f(1) = 1, n_2 = f(2) = 2$. Then we can compute:

$$n_3 = f(1 * 0.5 + 2 * (-0.4)) = f(-3/10) = 0$$
$$n_4 = f(1 * 0.4 + 2 * 0) = f(0.4) = 0.4$$

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 $\hat{y} = 0.04$.

13.3 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.

13.4 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 14

14.1 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).

14.2 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

$$TPR = \frac{TP}{TP + FN}$$
$$N = (TP + FN) + (TN + FP)$$

Then

$$TPR = \frac{TP}{N - (TN + FP)}$$

which implies $TP = TPR \times (N - TN - FP) \approx 171.0$

Problems of Chapter 15

15.1 The correct answer is B: There is one misclassified observation and so $\varepsilon_1 = 1/4$ and $\alpha_1 = \frac{1}{2} \log \frac{1-\varepsilon_1}{\varepsilon_1} = \frac{1}{2}\sqrt{3}$. Then the un-normalized weights become:

$$w = \left[e^{-\alpha_1}\ e^{\alpha_1}\ e^{-\alpha_1}\ e^{-\alpha_1}\right] = \left[\tfrac{1}{\sqrt{3}}\ \sqrt{3}\ \tfrac{1}{\sqrt{3}}\ \tfrac{1}{\sqrt{3}}\right]$$

normalizing gives:

$$w = \frac{1}{6}\left[1\ 3\ 1\ 1\right]$$

and so option B is the correct.

15.2 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.

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 $(0, 10, \frac{1}{3}, -4, \sqrt{2}, \pi$, 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 $\boldsymbol{x}, \boldsymbol{y}, \cdots$. 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

$$\boldsymbol{x}^T = \begin{bmatrix} -1 & 4 & 1 \end{bmatrix}.$$

Uppercase bold roman letters $\boldsymbol{W}, \boldsymbol{A}, \boldsymbol{B}, \cdots$ 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 ith 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 $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N$ of a M-dimensional vector

$$\boldsymbol{x} = \begin{bmatrix} x_1 & \ldots, x_M \end{bmatrix}^T$$

we can combine the observations into a $N \times M$ data matrix $\boldsymbol{X}$

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^T \\ \vdots \\ \boldsymbol{x}_N^T \end{bmatrix} \tag{A.1}$$

in which the ith 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

$$\mathbb{E}_x[f] = \int_{-\infty}^{\infty} dx\ p(x)f(x, y).$$

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 $\boldsymbol{A}\boldsymbol{x}$. Recall for two matrices $\boldsymbol{A}, \boldsymbol{B}$: $(\boldsymbol{A}\boldsymbol{B})^T = \boldsymbol{B}^T\boldsymbol{A}^T$ and that the identity matrix I_M is the $M \times M$ matrix such that for instance

$$\boldsymbol{I}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We may suppress M and simply write $\boldsymbol{I}$. Remember $\boldsymbol{A}\boldsymbol{I} = \boldsymbol{I}\boldsymbol{A} = \boldsymbol{A}$ assuming the dimensions match.

A $N \times M$ matrix $\boldsymbol{A}$ is said to be symmetric if $\boldsymbol{A}^T = \boldsymbol{A}$ and quadratic if $N = M$. For a quadratic matrix $\boldsymbol{A}$, if there exists a matrix $\boldsymbol{A}^{-1}$ such that

$$\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I}$$

then $\boldsymbol{A}$ is said to be invertible and $\boldsymbol{A}^{-1}$ is the inverse. A necessary and sufficient requirement is that the determinant of $\boldsymbol{A}$, $\det(\boldsymbol{A})$, is non-zero.

Subspaces and Eigenvalues

If we are given vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$ and numbers $a_1, \ldots, a_n \in \mathbb{R}$ then the vector

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

is said to be a *linear combination* of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$. 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 $\boldsymbol{x}_1, \cdots, x_n \in V$ then

$$a_1\boldsymbol{x}_1 + a_2\boldsymbol{x}_2 + \cdots + a_n\boldsymbol{x}_n \in V.$$

All vectors that can be written as a linear combination of a set of vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ is called the *span* of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$. Notice the span is a subspace of $\mathbb{R}^d$.

A set of vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ is said to be linearly independent if

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

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 $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, $n \leq d$ such that $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d$ are linearly independent and such that the span of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ is V. Such as 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

$$\boldsymbol{x}_i^T\boldsymbol{x}_j = \delta_{ij},$$

where $\delta_{ij} = 1$ if $i = j$ and 0 otherwise. Finally, recall that if $\boldsymbol{A}$ is quadratic and there exists a vector $\boldsymbol{x} \neq \boldsymbol{0}$ and a number λ such that

$$\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$$

then $\boldsymbol{x}$ is said to be an *eigenvector* of $\boldsymbol{A}$ and λ the corresponding *eigenvalue*. Suppose $\boldsymbol{x}_1, \boldsymbol{x}_2$ are eigenvectors of $\boldsymbol{A}$ with eigenvalues λ_1, λ_2 and suppose $\lambda_1 \neq \lambda_2$ then $\boldsymbol{x}_1, \boldsymbol{x}_2$ are orthogonal:

$$\boldsymbol{x}_1^T\boldsymbol{x}_2 = 0$$

In particular, if $\boldsymbol{A}$ is a $d \times d$ symmetric matrix, $\boldsymbol{A}^T = \boldsymbol{A}$, 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

$$f : \mathbb{R}^D \to \mathbb{R}$$

to explicitly specify the dimensions of the spaces f map between. This notation, as well as the notation $\boldsymbol{x} \in \mathbb{R}^d$, 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 $\boldsymbol{f}$ with a boldface:

$$\boldsymbol{f} : \mathbb{R}^D \to \mathbb{R}^M$$

To give two quick examples, if

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

then we can define $f : \mathbb{R}^3 \to \mathbb{R}$ where:

$$f(\boldsymbol{x}) = x_1 + x_2 x_3$$

or another example $\boldsymbol{g} : \mathbb{R}^3 \to \mathbb{R}^2$

$$\boldsymbol{g}(\boldsymbol{x}) = \begin{bmatrix} x_1 + \sin(x_2) \\ x_3 + 2e^{-x_1} \end{bmatrix}.$$

Notice we may also sometimes write $f(\boldsymbol{x})$ 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 $\boldsymbol{x}$

$$\frac{df}{dx}(x) \quad \text{and} \quad \frac{\partial f}{\partial x_2}(\boldsymbol{x}) \tag{A.2}$$

as well as integrals over some or all variables of a function:

$$I = \int_{\mathbb{R}^D} d\boldsymbol{x} f(\boldsymbol{x}), \quad I(x_1) = \int_{\mathbb{R}^2} dx_2 dx_3 f(x_1, x_2, x_3) \tag{A.3}$$

However, knowledge of integrating actual function will not be used and integrals are mostly used to state theoretical results. Finally we will use $\nabla f(\boldsymbol{x})$ as the divergence of a function $f : \mathbb{R}^D \to \mathbb{R}$ evaluated at $\boldsymbol{x}$:

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(\boldsymbol{x}) \\ \vdots \\ \frac{\partial f}{\partial x_D}(\boldsymbol{x}) \end{bmatrix}$$

Slightly more advanced concepts

Recall the multivariate Taylor expansion of a function $f : \mathbb{R}^D \to \mathbb{R}$ around a point $\boldsymbol{x}$ can be written as

$$f(\boldsymbol{x} + \boldsymbol{\delta}) = f(\boldsymbol{x}) + \boldsymbol{\delta}^T \nabla f(\boldsymbol{x}) + \text{ higher order terms}$$

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

$$\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}} f(\boldsymbol{x})$$

as the value $\boldsymbol{x}^*$ 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 $h : \mathbb{R}^D \to \mathbb{R}$ is zero, written as:

$$\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}: h(\boldsymbol{x})=0} f(\boldsymbol{x})$$

this can be done by introducing a Lagrange multiplier $\lambda \in \mathbb{R}$ and solve the problem

$$f(\boldsymbol{x}) + \lambda h(\boldsymbol{x}) = 0$$

by taking derivatives with respect to $\boldsymbol{x}$ and λ and set these equal to zero. That is, by simultaneous solving the $D + 1$ equations:

$$\nabla f(\boldsymbol{x}) + \lambda \nabla h(\boldsymbol{x}) = 0 \text{ and } h(\boldsymbol{x}) = 0$$

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 $(x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$ is a non-negative function

$$p : \mathbb{R}^d \to \mathbb{R}_+$$

which integrates to one:

$$\int_{\mathbb{R}^d} dx_1, \ldots, dx_d \; p(x_1, \ldots, x_d) = 1$$

where $\mathbb{R}_+$ is the interval $[0, \infty[$. Since the integration limits are often redundant we may also write the above as simply

$$\int_{\mathbb{R}^d} d\boldsymbol{x} \; p(\boldsymbol{x}) = 1$$

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

$$p(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

notice that $\int dx \; p(x) = 1$. In this case the probability that the random variable x falls within the interval $[2, 3]$ is simply:

$$P(x \in [2, 3]) = \int_2^3 dx \; p(x)$$

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

$$\mathbb{E}_x[f] = \int dx \; p(x) f(x, y).$$

References

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`.

C.M. Bishop. *Pattern Recognition and Machine Learning*. Information science and statistics. Springer, 2013. ISBN 9788132209065. URL `https://books.google.dk/books?id=HL4HrgEACAAJ`.

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-WOmSyQC`.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

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.

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.

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. Quételet. *Correspondance mathématique et physique*. Number v. 10. Impr. d'H. Vandekerckhove, 1838. URL `https://books.google.dk/books?id=8GsEAAAAYAAJ`.

C.F. Gauß. *Theoria Motvs Corporvm Coelestivm In Sectionibvs Conicis Solem Ambientivm*. Frid. Perthes et J. H. Besser, 1809. URL `https://books.google.dk/books?id=7jJbAAAAcAAJ`.

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=tVIjmNS3Ob8C.

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.

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.

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.

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=FRcOAAAAQAAJ.

David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

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.

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. 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.

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.

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.

Andrey Nikolayevich Tikhonov. On the stability of inverse problems. In *Dokl. Akad. Nauk SSSR*, volume 39, pages 195–198, 1943.

Alan M Turing. Computing machinery and intelligence. *Mind*, pages 433–460, 1950.