Machine Learning (2023-24)

Dr Peter Huthwaite, Mechanical Engineering

p.huthwaite@imperial.ac.uk

Important note: These notes are still relatively new. Please inform me if you find anything which could be improved for future years.

Second important note: I am happy to make more accessible versions of these notes available electronically on a reasonable case-by-case basis. This may include left justification, different fonts (sizes, sans-serif etc.). Please let me know.

Contents

Chapter 1

Introduction to machine learning

C1A

I heard a story once, which is probably apocryphal[1], about an early attempt to apply Machine Learning (ML) to a classification problem. The challenge was to be able to automatically distinguish between Soviet tanks and those of the USA from photos.

The researchers acquired several images of each type of tank and fed it into the algorithm. Nothing happened; the algorithm was unable to make a decision any better than random guessing. They looked around the literature and realised that these algorithms are often dependent on having large training datasets, so they found more images of the tanks from various sources - books, pieces of government propaganda, newspaper articles. They fed all of these into a machine learning model, and gradually they improved the results; the algorithm started to be able to distinguish between the two types in a statistically significant manner.

However, they then hit a problem. While the model seemed to perform well on the training datasets, when applied to real life images, the results were again little better than guessing. The researchers struggled with this; what was causing the difference?

Eventually they managed to identify that the key distinction in the training data images between the two tanks was that the Soviet tanks were typically pictured against a grey cloudy sky, while the USA tanks typically had a blue sky and sunshine[2]. The researchers had trained an algorithm to distinguish the weather in the image, rather than identifying the tank.

This story is unlikely to have actually happened for a number of reasons, but it serves as a useful introduction. It is one form of overfitting, where the trends in the training data are captured, but the ability to generalise to real data is very limited. It also highlights the need for validation – it is critical to test out any model with real data.

There is also a question about what problem is actually being solved. Some of the best stories about machine learning "going wrong" relate to this; examples of sheep being identified only because they are white blobs on a green background are very common - take the green field away and suddenly the algorithm won't work. However, despite these inherent issues, machine learning does continue its rise, with an increasingly broad applicability of it to a wider range of problems. It is important that we are all aware of the issues and complexity surrounding this when it comes to applying machine learning.

1.1 Why study machine learning?

C1B

I would extend this question slightly, to "Why study machine learning as part of a Mechanical Engineering degree?" – one would not traditionally associate a subject which is essentially data-processing with a degree focused on applying principles such as stress analysis or fluid mechanics to real-world problems. However, ML is a form of engineering; there are underlying principles in the models used which are then applied to the real world. Although these techniques are not

[1] I did a quick google and found very little, but it's still a good story

[2] One imagines that this is more likely to be down to political reasons rather than any inherent national weather discrepancies.

physics-based, they can act to solve exactly the same problems that you have been working on solving throughout your degree, and this makes it quite powerful.

There are many overlaps in the skills and techniques covered on the rest of the degree, including

- Linear algebra
- Matrices
- Eigenvalues
- Statistics
- Optimisation and inversion
- Signal processing
- Computer programming
- Modelling.

The overlap gives additional options for practising and developing knowledge of underlying tools in other subjects further, and there may also be possibilities for closer integration between techniques in one of the more traditional subjects with ML. After all, we could look at lots of subjects and say that they revolve around spotting patterns of behaviour then exploiting them.

Within the Mechanical Engineering department there is research which is linked to ML, including in fluid mechanics to model flow, in non-destructive testing to identify defects from ultrasound data, and in several areas of robotics. This activity is in part a reflection on the power (and to some extent, hype) around ML. This also relates back to the significant student demand to have a course on ML, which should provide greater employability and broaden knowledge for our graduates.

A key outcome of this course will be that our engineering graduates should be "smart users" of machine learning. Consider a scenario where a graduate is working for a large engineering company, and in one of their meetings, their boss suggests using machine learning to solve the problem. Based on this course, it is hoped that the graduate would be able to intelligently explain whether this is a sensible solution or not for this problem, and have sufficient knowledge to develop a solution as appropriate.

1.2 Course approach

C1C

Some courses for ML have heavily focused on writing up algorithms from scratch. This does give very good insight into how an algorithm works, and certainly this would have been a valid approach several years ago, when there were limited alternatives available. However, there are some significant downsides with this; it relies on very strong programming skills from students, it can be very slow to implement more complex algorithms, it is easy to make mistakes and spend a long time debugging with little gain, and the algorithms will typically not be optimised, limiting performance and preventing them from being applied to larger datasets and problems.

Therefore, on this introductory course, the goal is for students to be able understand the theory to write some of the simpler routines (which will be more of a focus in the exam), but be able to utilise libraries for the more complex scenarios (which will be the focus of the tutorials and corresponding coursework). This should provide a balance between getting some experience coding up algorithms to understand the underlying principles and the ability to apply the routines more broadly[3]. Students completing this course should be able to use their understanding of the material covered here as a basis for researching more advanced techniques in the future, should it be necessary.

1.3 What the course will cover

C1D

It is highlighted that machine learning is a massive subject and continuously increasing in size, so this course does not attempt to cover everything. This is particularly true given that it is an introductory course for engineering students with no experience of the field. Instead it aims to give a cross section of some of the key methods used in practice.

Machine learning is the development of models which identify patterns within data and use these to provide some sort of useful output. The different techniques for ML are quite diverse in nature, and quite disjointed; this contrasts with many of the traditional engineering subjects, for example in stress analysis, where the same principles and techniques are largely utilised through the whole subject.

In this introduction to ML, we will cover methods from areas such as statistics, linear algebra and numerical optimisation. The tools we will study have been selected since they are the most commonly used techniques, and provide a good overview of what is used in practice – these techniques would typically form the first chapters of a general textbook on ML. They also serve to provide a cross-section of the different theories utilised in machine learning, such that, having completed this course, students should have a good basis learn about and utilise new methods they may come across. Each of the tools we study should be viewed independently, not as something which will necessarily flow together, although efforts have been made to make the course link together in a coherent fashion as much possible. The structure of the course is given below.

- Bayesian techniques
  - These are techniques built upon statistics. It makes sense to study these early on in the course because much of the theory depends on statistical understanding. Having said that, since this is for engineers I have tried to keep the statistical content to a minimum throughout.
- Regression
  - Regression is amongst the simplest forms of machine learning; many students will have studied linear regression as part of basic statistics in A-level maths. Here we cover this briefly, then extend it to dealing with higher orders, as well as fitting the multi-parameter problems common in ML.
- Linear discriminant functions
  - Linear discriminant functions are one of the simplest approaches to classification; the example of Fig. 1.4(b) can be expressed as a linear discriminant function. These functions form an important basis for neural networks as well as support vector machines studied next.
- Support vector machines
  - Support vector machines are a powerful tool used for classification problems, and are a specific, efficient, form of linear discriminant functions. They can generalise well to higher orders, allowing greater complexity to be incorporated.
- Neural networks
  - These are hugely important techniques these days and are used extensively for deep learning approaches. They utilise multiple linear discriminant functions, so build on the techniques from there.
- Non-parametric techniques

[3] As an engineering analogy, these days a finite element course would not expect students to code up an FE package themselves; they would use an existing package. But in the past (up until the late-1990s on a course in our department), they would have written their own code.

  - These techniques include estimating probability densities and extend to nearest neighbour techniques. They are amongst the most simple techniques for classification, although do not fit neatly with any other approaches.

- Nonmetric techniques (decision trees and random forests)
  - Decision trees classify input data by taking a series of smaller decisions, with the splits accompanying each of these decisions forming the tree structure. These decisions can be based on linear discriminant functions or in fact any other classification algorithm. They are one of the earliest ML techniques. Random forests combine multiple decision trees together for improved performance. Again, while a key ML tool to understand, they do not directly link to the other approaches.

- Unsupervised learning
  - Everything studied so far has looked at supervised learning. At this point we cover unsupervised learning, specifically clustering and principal component analysis. Clustering closely relates to nearest neighbour techniques, while principal component analysis pulls out key relationships between the different data parameters and utilises similar vector and matrix algebra to that used in linear discriminant functions.

- Machine learning in the real world
  - Having studied a cross-section of methods, this last section looks at the real applications of ML, asking questions such as:
    * What are the consequences to using machine learning in practice?
    * Where is the field going?
    * What are the limitations?

There are a number of general concepts which are covered on the course which do not fall into the main structure. In the taught section of the course these will often be slotted in for practical reasons (such as depending on timing of individual lectures, or to cover things such that necessary tools are in place prior to specific tutorials). The notes here will not necessarily keep this order. These concepts include:

- Validation, including the k-fold technique
- Feature extraction
- Metrics

1.4 Basic machine learning concepts

C1E

The basic concept of machine learning is that we are trying to come up with some model which will learn from some training data, then, based on this, produce some output for a new (unseen) set of data. Often, interesting applications for ML are when this output is some decision, or sort of prediction. As an example, you can heat treat components, but some fail. You want to predict how likely they are to fail, and you have a big database of similar components which have been treated in the past which you want to use for this. ML can be trained on the database and used to make the predictions about failure. To do this, ML will use a MODEL, which is a description of how the data behaves. In most problems a variety of different models can be used. Models will produce specific outputs for particular inputs and have the capability to align with a particular output, i.e. be trained. Some models may be more suitable for different applications than others. On this course we will cover the main models used across machine learning[4], and try to provide insight about how they work and how you can use these. It should be recognised that these methods

[4]This is inevitably subjective, and the course list is based on my experience of the field. It should cover the commonly accepted "key areas".

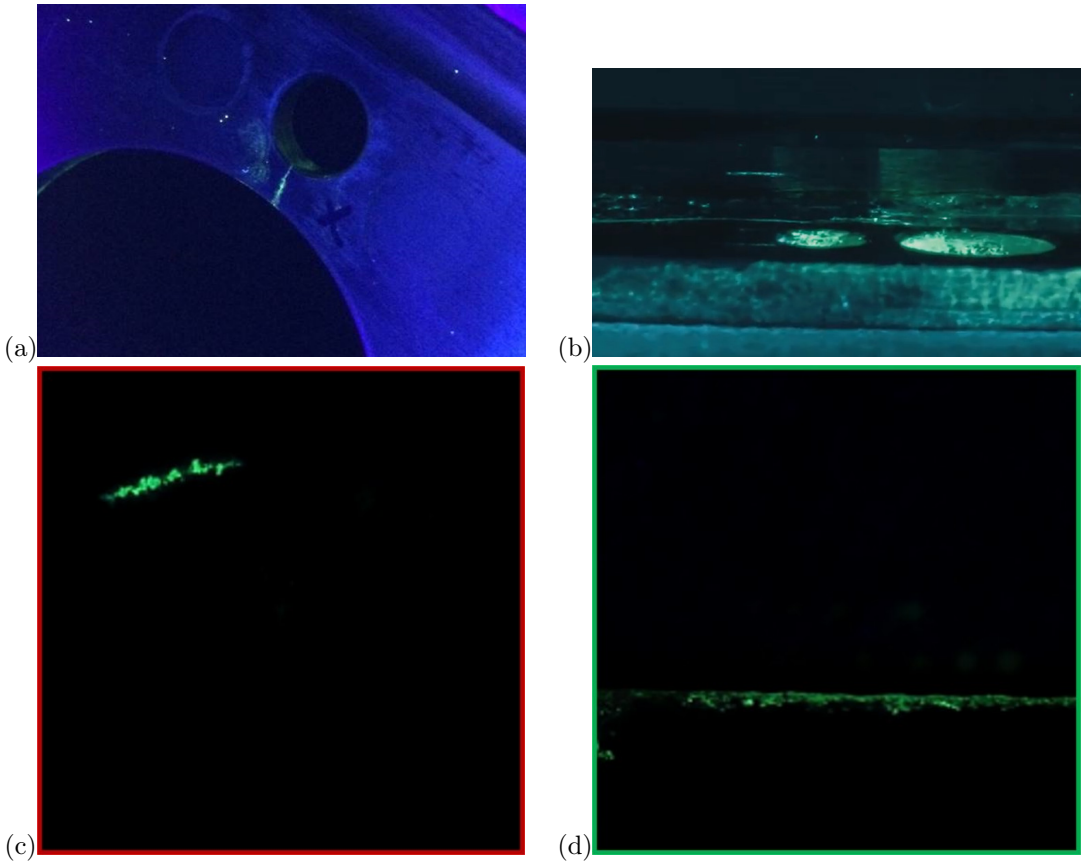


Figure 1.1: Examples of dye penetrant inspection of a component, where fluorescent dye is sprayed onto the surface then cleaned off, to leave indications coming from defects. The images show the dye fluorescing green under the UV light. (a) shows a clear indication, while (b) has numerous additional indications present, making identification a challenge. (c) and (d) show isolated indications, (c) corresponding to a defect and (d) a false indication, likely corresponding to component geometry.

use very different principles and hence can appear very distinct – it is a challenge to find many common threads beyond the principle of fitting inputs to outputs.

To run through some more of the concepts, we will focus on a specific example to discuss the basic concepts of machine learning which comes from non-destructive testing. Dye penetrant inspection is a method to detect and size defects (typically cracks) appearing at the surface of a component. The surface is sprayed with dye, which is allowed to rest there before being cleaned off. The small amount which has soaked into any cracks will remain there and will be visible once the residual has been removed. This dye can be brightly coloured, but often for maximum sensitivity (such as for applications in the aerospace industry) fluorescent penetrant is used, which glows green under UV light. Figure 1.1 shows two examples of dye penetration images. While Fig. 1.1(a) shows a very clear indication, Fig. 1.1(b) is less clear and several of the bright indications are present, which do not all correspond to defects.

Traditional approaches to identifying the defects rely on human interpretation of the component, typically undertaken in shifts of many hours. This is done by highly trained and certified operatives. Compounding this, defects are very rare, ultimately increasing the risk that something important may be missed. While much of the dye penetrant process can be undertaken automatically, the final interpretation stage is the only manual part. Being able to automatically identify defects would greatly improve the technique. Even being able to automatically eliminate a large fraction (e.g. 90%) of indications which we can be certain are not defects, leaving the remainder to the operator, would be very valuable in improving the efficiency and reliability of the process.

1.4.1 Supervised learning

C1F

Supervised learning involves training a model using data where there is a specific output (or set of outputs) identified for a given set of input parameters. We may have photos taken of dye penetrant inspections of a component, and we ultimately want to predict from this whether there is a defect present. Our training dataset will have a baseline truth of whether or not there is actually a defect present – the desired output, along with parameters extracted from the image (such as length or brightness, or possibly pixel values for all points in the image), which are the inputs. This is a supervised learning problem, because the outcome is known in the training data. There are two main types of supervised learning that we will consider: classification and regression.

1.4.1.1 Classification

The example given above is a classic classification scenario: the output (whether a defect is present or not) can be one of two scenarios and hence is discrete - there is no continuous measure. In a classification problem you often have two classes, but there can be any number of different classes as appropriate.

1.4.1.2 Regression

In regression, rather than our output being in discrete classes, we have a continuous output. To adjust the above example, we could have our output as the estimated time that the component can be used for before failure. Time is a continuous quantity, so this is a regression problem.

1.4.2 Unsupervised learning

In unsupervised learning, we wish to identify patterns within the data available, without any prior knowledge about what is important. As an example, we may know that we are getting failures at particular times at particular locations around the country. Unsupervised learning can be used to process the data about all these failures and identify if there are any trends which could help pinpoint why this is happening. On this course we will look at clustering and Principal Component Analysis (PCA) techniques.

1.4.3 Rules-based techniques

It is possible to manually develop rules-based techniques to solve some of these problems. For example, we can measure the length of an indication in a dye penetrant image. If we know that this being above 5mm indicates that it is a defect, we could just write that as a rule into our model. The challenge here is (a) how to generate these rules and (b) how to program these in a general way, given that some behaviour can be highly complex. Machine learning provides a toolbox of different models which can describe the rules, and a method to generate the rules in the first place through training. This automated training process may require significant data, but can capture complexity that a human would find very difficult to achieve. The rise of ML is partly due to the availability of huge amounts of data.

1.4.4 Basic machine learning

C1G

Let us consider a basic machine learning approach applied to the dye penetrant problem. We will assume that we have a training dataset of indications containing both defects and otherwise, and that this classification is known. This dataset also contains information about the different indications, such as length, colour and brightness. For illustrative purposes I have generated some example data for these two classes. Figure 1.2 shows a histogram which compares distributions for this two-class problem as the length of the indication varies.

We notice there is a difference in the distributions; the indications in the defect class tend to be longer than the non-defect class. This provides an opportunity, based on the training dataset, to come up with a rule which can be applied to the problem to classify the points. We can place a decision boundary in length, as a threshold, and state that everything above that is a defect and

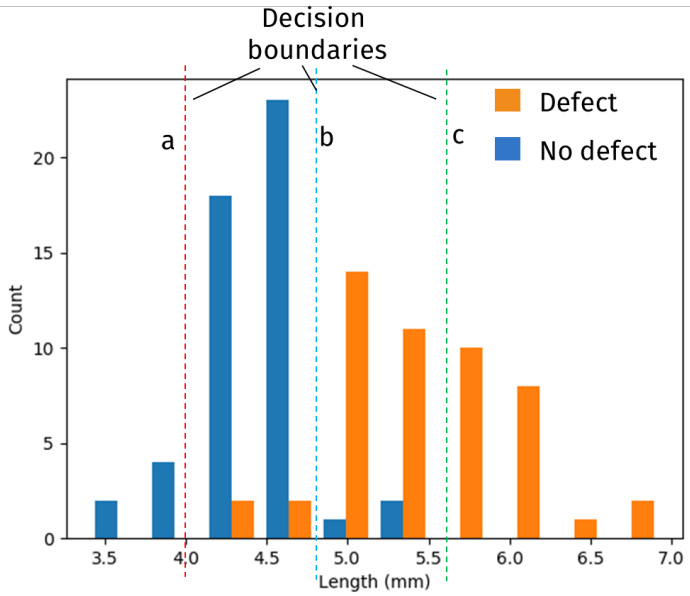


Figure 1.2: Histogram for two-class problem looking at whether a particular length of indication corresponds to a defect or not.

everything below is not. Three potential positions of this, a, b, and c, are marked in the figure. Of these three, have a think about which you think is best and why.

This introduces a question of context, which is particularly important with engineering problems. We are looking at a safety-critical problem; there is a significant difference in cost between incorrectly labelling a non-defect as a defect vs labelling a defect as a non-defect. If we get things wrong, in one case we may waste a small amount of money repairing or replacing a component unnecessarily (false positive), and in the other we could cause huge loss of life and financial expense (false negative). We will come to study this cost principle later in the course. However, based on this principle, you may have picked a, which will incorrectly label many non-defects as defects, but should capture almost all defects, which could be important in safety critical situations. Alternatively, you may have picked threshold b, which will generally result in the lowest chance of error, but this does have a risk of missing some defects. Picking c will identify scenarios which are definitely defects, but miss quite a few; the value of doing this in practice may be limited in many cases[5]. There is no real "right answer" in this case because I have not given you enough information; I have not said to you what is the most important thing to achieve. It should also be noted that picking a or c changes the classification problem slightly; rather than classifying between defect/no defect we are classifying into no defect/unknown and defect/unknown respectively.

From the example above, we have significant uncertainty in our classification. There may be a different feature which we could study which may work better. Figure 1.3 shows the histograms for both classes as brightness varies. There is a slight difference between the means of the two distributions, but significant overlap overall, and this parameter is a significantly worse indicator than the length value from above, so this doesn't help much. This parameter doesn't seem to help much, so it is tempting to discard it. However, it may be possible to use these together somehow. C1H

Rather than considering the parameters in isolation, it may be possible to combine them together to improve the classification performance further. Figure 1.4(a) shows a scatter plot of the two classes together. Plotting in this way gives an indication about how we can add a new decision boundary between the two classes, one which is a function of both length and brightness; Fig. 1.4(b) shows an example of this. This boundary is closer to optimal, enabling the two classes to be separated better and exploiting the multi-parameter nature of the data available. For refer-

[5]Some people have made suggestions in the past for this. Thanks to A MacLaren for: "Could boundary c be useful in non-safety-critical manufacturing applications to identify "factory seconds", which are to be sold more cheaply or repurposed? This would require a high purity of defects, e.g. misshapen chocolate bars, crockery with misaligned patterns etc."

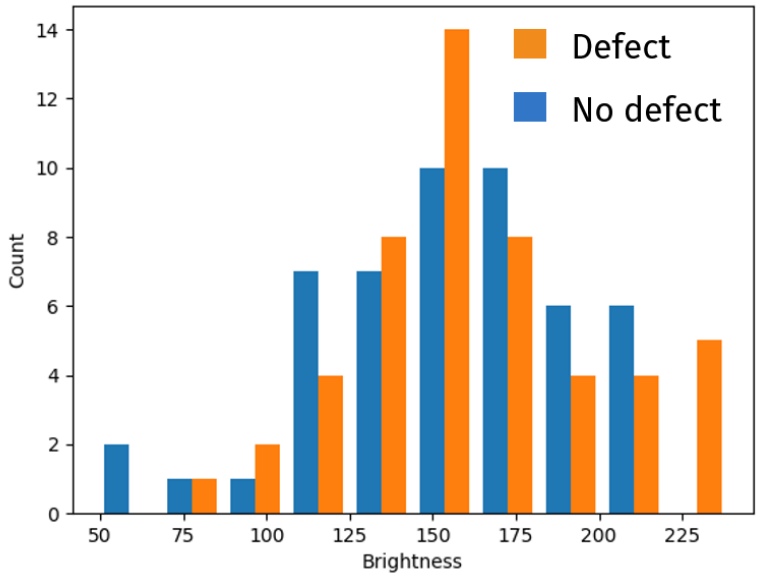


Figure 1.3: Histogram showing distributions of indications which are defects or otherwise as the brightness of the indication varies.

ence, the previous 1D approaches would have limited the decision boundary here to be horizontal or vertical. It is possible to make the decision boundary better at separating these two datasets by moving away from the straight line, as shown in Fig. 1.4(c); here we can effectively achieve perfect separation as the boundary conforms to the data. However, this is an extreme example of overfitting, where the model fits the training data very well but is likely to perform very poorly in a general case. Fig. 1.4(d) presents a compromise, fitting the data well but not overfitting. We would expect this to perform best in general, when applied to "unseen" data.

1.4.5 Parameter spaces

We should make a quick note about parameter spaces. In the example above, we highlighted the value of combining two parameters together, brightness and length. Typical applications of ML will have multiple input parameters and may produce multiple outputs. There are scenarios where these inputs could be in the millions, for example, when dealing with images where each pixel value itself may be fed into the algorithm. Visualisation of these higher-dimensional spaces can be a significant challenge, so we often illustrate things with two parameters in a 2D space, with the understanding that this will generalise to higher dimensional spaces, as in the example above. Throughout this course, we will often consider two-parameter inputs to aid the visualisation.

1.4.6 Definitions

C1I

To conclude this section on basic machine learning concepts, we will outline some of the mathematical descriptions used for the data.

A single data point for a model with n parameters can be expressed by the vector $\boldsymbol{x} = (x_1, x_2 ... x_n)$. For a complete dataset of m multiple points, this is expressed as a matrix:

$$\boldsymbol{X} = \begin{pmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,n} \\ x_{2,1} & x_{2,2} & \dots & x_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \dots & x_{m,n} \end{pmatrix} \tag{1.1}$$

Considering a supervised learning problem, for each of these data points, there will be a corresponding output. In some cases this is a single value, in which case the output can be expressed as

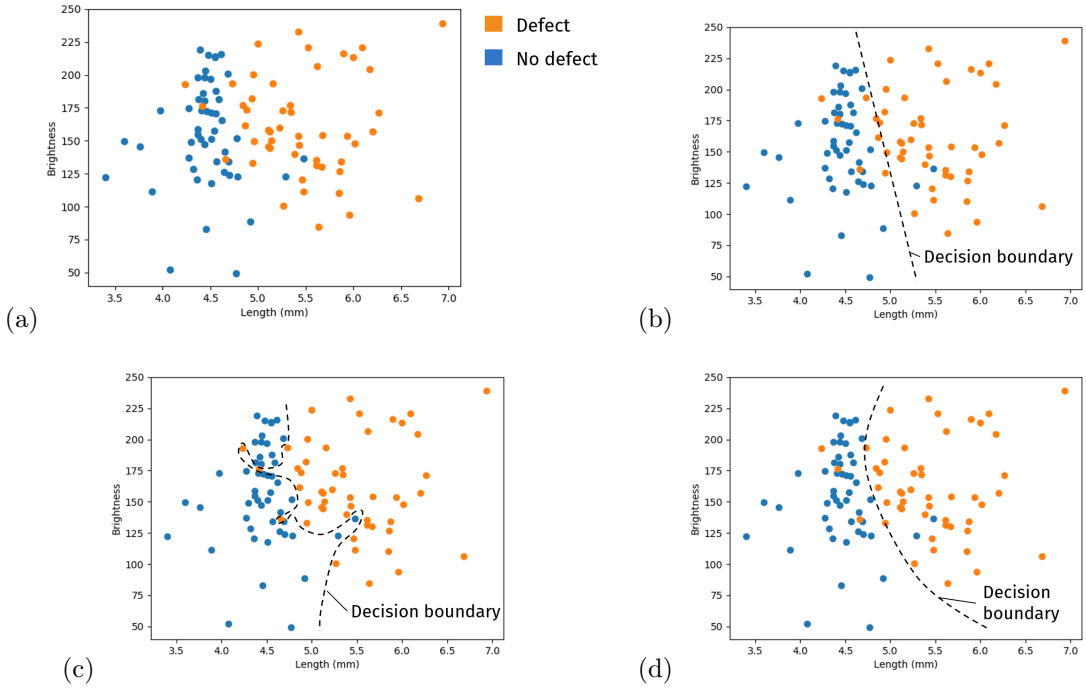


Figure 1.4: (a) Scatter plot of brightness vs length of indications for defect and non-defect classes. Potential decision boundaries are marked on (b)-(d); linear in (b), highly complex, conforming in (c) and a smooth curved boundary in (d).

a vector

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} \tag{1.2}$$

but if there are p multiple parameters in each output then these form a matrix

$$\boldsymbol{Y} = \begin{pmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,p} \\ y_{2,1} & y_{2,2} & \cdots & y_{1,p} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m,1} & y_{m,2} & \cdots & y_{m,p} \end{pmatrix}. \tag{1.3}$$

Examples of multiple outputs include coordinate values in 2D or 3D, or a football match score. Note that these outputs can be either continuous or discrete for a regression or classification problem. For unsupervised learning, there is no $\boldsymbol{Y}$ output, and we try to identify patterns in the $\boldsymbol{X}$ matrix itself. Also note that it is possible for $\boldsymbol{X}$ to have a single parameter, in which case it will drop down to a vector of length m. This is quite a rare scenario in practice.

1.5 Resources

Some potentially useful textbooks are:

- Pattern Classification by Duda, Stork and Hart
- Pattern Recognition and Machine Learning by Bishop
- A First Course in Machine Learning by Rogers and Girolami
- The Elements of Statistical Learning by Hastie, Tibshirani and Friedman

- Data-Driven Science and Engineering by Brunton and Kutz

Duda, Stork and Hart is an excellent resource for much of the material, and I found the regression section in Brunton and Kutz to be useful, although much of that book moves into control theory. Bishop is seen as one of the main textbooks by many in the field (this was recommended to me by someone from the Maths Department). Rogers and Girolami is based around a course given to undergrad computing students at University of Glasgow, so should be a good level, however I did find it quite statistics-heavy[6].

1.6 Practical bits

1.6.1 Tools

C1J

It will be assumed that you have knowledge of mathematical tools taught in the previous years of the MEng degree[7], including linear algebra; you are expected to revise this as necessary. This includes use of matrices, including matrix multiplication, index notation and eigenvectors. Regarding statistics, you will need to understand the normal distribution including for the multivariate situation (i.e. multiple input parameters). You should also understand conditional probability. Information about multivariate normal distributions is included in this chapter.

1.6.2 Computational tools

In theory, any language could be used for implementing machine learning tools, programming the concepts up from scratch should be possible where a library does not exist. However, for this course, Python is selected. The language is widely used for ML (arguably the most widely used), and has a huge range of tools and support available. It is also a very general language, so has broad applications outside machine learning, across engineering and elsewhere.

MEng students in the department should have studied Python in ME1 and ME2, so students should have some prior experience. It is recognised, however, that programming is often recognised as a challenge for our students (and our MSc students may not have studied Python at all) so the first tutorial is maintained as a relatively gentle introduction. Students are expected to independently learn the basics themselves from the many resources available online if they need it. A fairly short guide to Python for Matlab users is at https://towardsdatascience.com/python-for-matlab-users-ac3e0b8463a5, and I would strongly encourage everyone to go through part 1 (the free part) of the course here if they are familiar with Matlab but not Python: https://www.datacamp.com/courses/python-for-matlab-users.

1.6.2.1 Python IDE and environments

Python has a huge range of IDEs (Integrated Development Environment – essentially a text editor with the ability to run code and some other handy features). We will not be too prescriptive on this course, however, we encourage students to use Google's Colab – https://colab.research.google.com/, with some getting-started instructions here: https://youtu.be/YJP4JME-zS8. This is a full web-based approach, which avoids having to install and manage packages yourself. It should be noted that the final exam will be in Colab so you may wish to use this to gain familiarity (although picking it up is very easy).

While you may use your own computer and whatever environment you like, we will not be able to support with technical issues, such as installation; this goes beyond what we can provide. Note that the packages used are scikit-learn, numpy, pandas, matplotlib and keras, so you will need to install these.

1.6.3 Lectures

Ten 1-hour lectures will cover material from the course. There will be some additional online content for particular lectures where the material does not fit into the standard slot. All material

[6] Girolami was at Imperial until recently and did suggest his book for this course.
[7] Or equivalent undergraduate degree if you are an MSc student.

may be examined (as well as other parts of the course, such as tutorials).

1.6.4 Tutorials

C1K

There will be ten 2-hour tutorial sessions in the computer room. You will be given tutorial sheets to work through, electronically through blackboard. The intention of these is to provide practical experience of using the ML libraries available in Python, as well as implementing some simpler examples yourself.

You are expected to look up function definitions yourself online, rather than trying to memorise every single command; this is the way that most programmers work. You need to learn the structure of the language, including what approaches there are, then use references for the bits you need. The tutors will be able to guide you, but should not be expected to do the job of Google! Note that the scikit-learn library has excellent online documentation that you should use (which is one reason why it is the main library selected for the course). I have made a video about looking up the material at https://youtu.be/GMexl8U2T3o which you may find useful, particularly if you find it intimidating at first.

1.6.5 Notes

This document forms the notes to accompany the course. Generally, the order matches that of the lectures and tutorials. There is a slight discrepancy in the addition of Chapter 11. This contains the general material relevant to all areas of machine learning, such as validation techniques, variance and bias. These sections are often covered in a number of different lectures (depending on timing) but for these notes have been combined in one location. At the end of each chapter there are also some blank pages which you can use for adding your own notes.

Marker points have been added in the margin. These are labelled e.g. C5B; this example would be in Chapter 5, and B indicating it is the second label. Certain slides in the lecture will have the corresponding markers on to enable students to find the corresponding sections of the notes[8].

Each chapter does have some questions you may use to further your understanding of the subject – these are more theoretical written questions as opposed to the computational questions you will cover in the tutorials. This understanding may be helpful for the exam (and many have been taken from past exam papers). Note that these questions are "work in progress" and they will be added to and expanded through the years. It is emphasised that these questions should not be expected to fully cover all areas of the syllabus; questions in the final exam may come from any area of the course regardless of whether there is an end-of-chapter question on that specific topic.

1.6.6 Assessment

C1L

Assessment will be split between practical application (which is the focus of the submitted coursework) and the theoretical understanding (assessed primarily in the exam). While closely linked, the tutorials will be more focused on the practical implementation, while the lectures will be more focused on the theoretical principles. The provisional allocation of marks between the exam and different sections of coursework will be provided on blackboard.

1.7 Useful concepts

From previous years, it has been recognised that there are some commonly used concepts which are not widely understood by students starting the course; specifically these are covariance matrices and the use of the "meshgrid" command in Python.

1.7.1 Covariance matrices

C1M

If you have a single random variable, conforming to the normal distribution, this distribution can be described by simply the mean and the standard deviation. If you extend this to a distribution

[8] This is new for 2023, so please let us know if there are any errors.

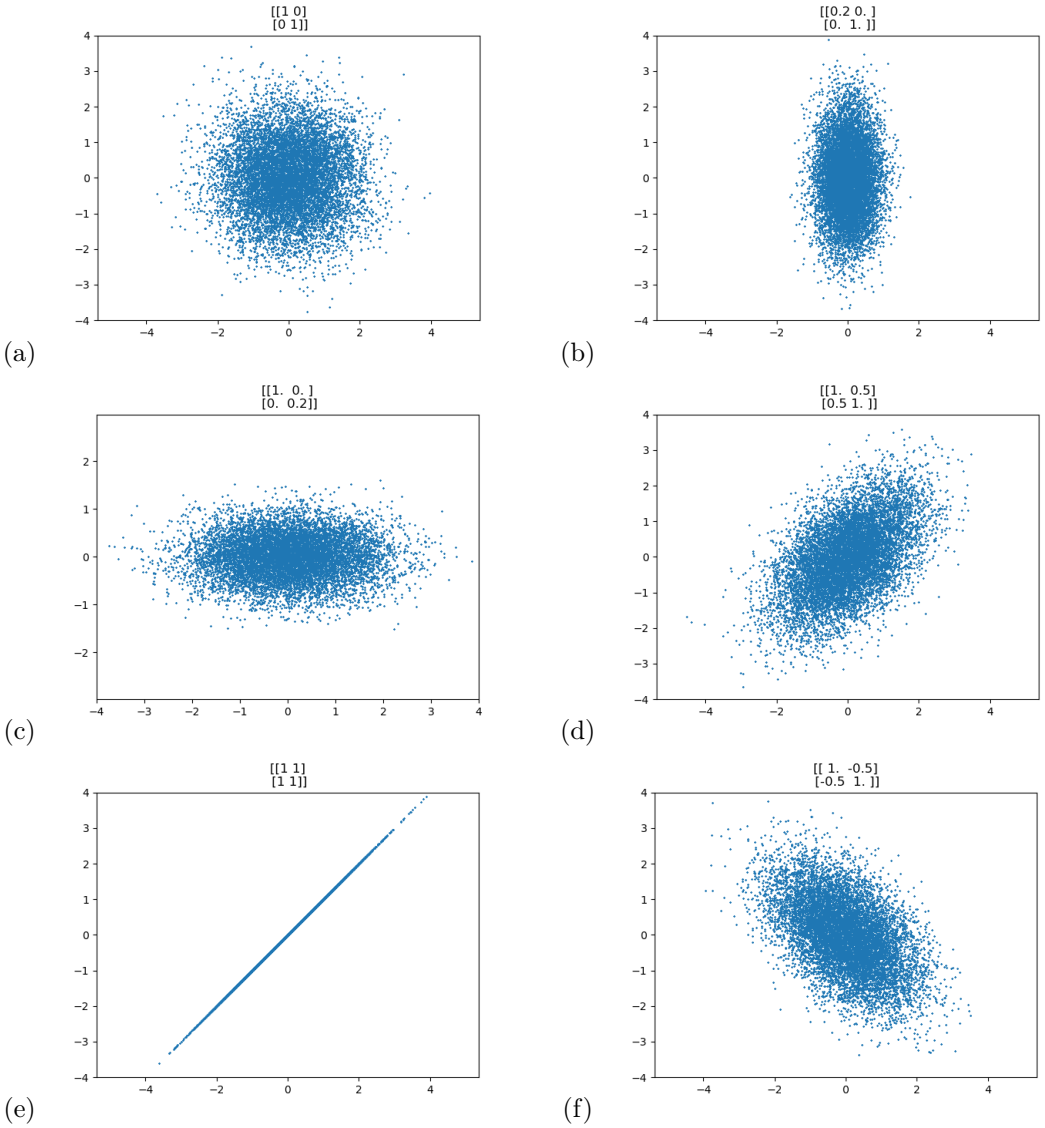


Figure 1.5: Examples of distributions from different covariance matrices.

in 2D space, i.e. consisting of two variables, you can then assume that you have two means and two standard deviations. This is correct, to an extent, however, there is also an element of how each variable varies with each other. If the variables have a positive correlation, then if variable 1 is high, variable 2 will also typically be high. This is illustrated in Fig. 1.5. In Fig. 1.5(a), (b) and (c), there is no correlation, i.e. each variable has its own standard deviation and can be described independently. In Fig. 1.5(d), (e) and (f), there is strong correlation, and the variables are not independent.

Statistically, the way this is described is with the covariance matrix. For 2 variables, this is a 2×2 matrix (as in Tutorial 1) - or for n variables it becomes $n \times n$. Each term in the matrix describes how one variable varies with respect to another. The diagonal terms correspond to variance, i.e. how each variable varies with respect to itself. A value of zero at each point in the matrix indicates no correlation, and a large value (up to the variances of the two respective variables) indicates a strong correlation. The covariance of variable a with b is the same as the covariance of variable b with a, so the matrix must be symmetric.

The variance of a dataset can be estimated by subtracting the mean from every point, squaring the result and summing. The covariance matrix can be calculated in an analogous manner as

$$\boldsymbol{C} = (\boldsymbol{X} - \boldsymbol{\mu})^t (\boldsymbol{X} - \boldsymbol{\mu})/m \tag{1.4}$$

where m is the number of points in the dataset $\boldsymbol{X}$.

For reference, several examples are shown in Fig. 1.5, with the corresponding covariance matrices listed as the titles. In the first three cases there is no correlation between the two variables. The standard deviations (variances in the matrix) are changed, as you can see from the diagonal terms, in Fig. 1.5(a) both are 1, in Fig. 1.5(b) the x variance is changed to 0.2, and in Fig. 1.5(c) the y variance is changed to 0.2. In Fig. 1.5(d) we have nonzero covariance between the two variables, so you can see there is a correlation - this is covariance of 0.5. In Fig. 1.5(e) we have an extreme correlation where the covariance equals the variance of both variables, and you can see that we get a straight line. Finally, Fig. 1.5(f) shows a negative covariance, corresponding to a negative correlation.

My Python code to generate these is as follows (note that this is just for the covariance matrix shown in Fig. 1.5(c)):

```
import numpy as np
import matplotlib.pyplot as plt

covar = np.array([[1, 0], [0, 0.2]])
distvals = np.random.multivariate_normal([0, 0], covar, 10000)

fig, ax = plt.subplots()
ax.scatter(distvals[:,0], distvals[:, 1], marker='x', s=1)
plt.axis('equal')
plt.title(np.array2string(covar))
plt.xlim([-4, 4])
plt.ylim([-4, 4])
plt.show()
```

1.7.2 Meshgrid

C1N

Visualising ML performance is a challenge; for a set of input parameters, we obtain a set of output parameters, and trying to understand what is going on is not straightforward. A common approach for simpler cases is to consider two input parameters and a single output; we looked at an example of this earlier in Fig. 1.4. We can produce scatter plots, as in that example. However, this is limited to specific random realisations. Instead, we could systematically sweep through every combination of the input parameters and use these to produce an image plot, where each pixel corresponds to the output from the algorithm for the corresponding two input values on the x and y axes. The colour of the pixel indicates the output.

To produce such an image, we need to define suitable sets of parameters to do this sampling throughout the 2D space. If we were doing this is 1D, i.e. with one parameter, it would be straightforward; we could just define values of each parameter using linspace()[9] or similar, with an array of numbers such as

$$\boldsymbol{x}_1 = (1, 2, 3, 4) \tag{1.5}$$

which can be inserted into the model to test what the output would be, then plot this output. Note that in practice we would have many more numbers than this, typically hundreds of points, but 4 have been used here for illustrative purposes. In 2D this is a bit more complex. We may have a second parameter we want to vary in the same way, i.e.

$$\boldsymbol{x}_2 = (1, 2, 3, 4) \tag{1.6}$$

[9]Note that all the functions mentioned in this section are part of the numpy package, so should be preceded by np. as appropriate.

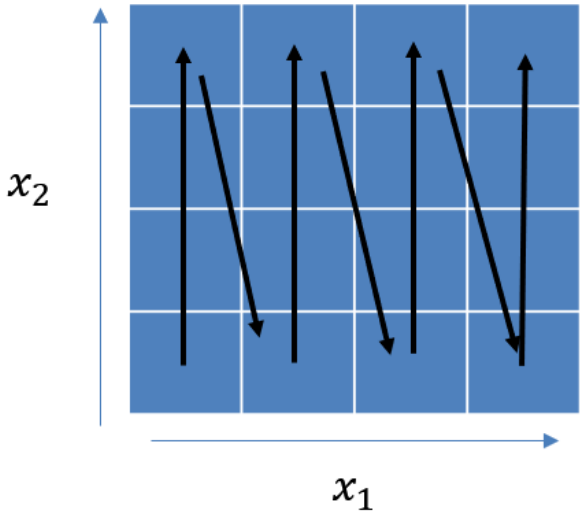


Figure 1.6: How to loop through all points in a 2D space using meshgrid.

where we define the length here as 4, as before (but this need not be the case). The issue here is that we need all the combinations of both parameters; if we vary both simultaneously then we will test the model at

$$(\boldsymbol{x}_1, \boldsymbol{x}_2) = ((1,1),(2,2),(3,3),(4,4)) \tag{1.7}$$

i.e. down the diagonal, so we won't include parameter combinations as (1, 2), (4, 1) or anything else not on the diagonal. Instead we need an approach where we take the first value of parameter 1, loop through all values of parameter 2, increment parameter 1, loop through parameter 2 again and repeat until all the 4×4 combinations have been covered. This would produce values of

$$\boldsymbol{x}'_1 = (1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4) \tag{1.8}$$
$$\boldsymbol{x}'_2 = (1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4) \tag{1.9}$$

These would now sample at every point in the 2D space. Figure 1.6 illustrates this. Note that it is possible to loop the other way, i.e. make $\boldsymbol{x}_1$ "fast" and $\boldsymbol{x}_2$ "slow" so that we hold $\boldsymbol{x}_2$ constant while we loop through $\boldsymbol{x}_1$.

The meshgrid function takes in the inputs $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ and outputs $\boldsymbol{x}'_1$ and $\boldsymbol{x}'_2$, which can then be used to sample the model. The output will then be a 1D linear array where each value is from the corresponding combinations of $\boldsymbol{x}'_1$ and $\boldsymbol{x}'_2$. We need to use the reshape() function to convert the 1D array into a 2D matrix, which can then be plotted as an image. The reshape function can reshape any array (provided the total number of elements remain the same) but the default behaviour is to move through to populate all values of the last dimension first (note that you can change this behaviour if you wish, but I wouldn't recommend it); this matches the desired behaviour to convert our output vector into the image grid. In the example above, the output will be a 1D array of length 16 elements; reshape will be used to populate a 4×4 matrix, looping through the $\boldsymbol{x}_2$ direction first. This 4×4 matrix can be plotted as an image. As mentioned, in practice, many more points would be used to produce an image.

1.8 Conclusions

This chapter has been an introduction to the machine learning course. You should now understand the motivation behind the course, some of the basic concepts of machine learning, what tools you will need for the course and how the course is structured.

1.9 Questions

1. Variable a has standard deviation 3 and mean 0, and variable b has standard deviation 2 and mean 1. The covariance between them is -0.3. Write out the covariance matrix.

1.9.1 Solutions

1.

$$cov = \begin{pmatrix} 9 & -0.3 \\ -0.3 & 4 \end{pmatrix}$$

Chapter 2

Bayesian decision theory and maximum likelihood

At its very core, Machine Learning depends on statistics. The training data will have a particular probability distribution associated with it, and, as we saw in the last chapter, by assuming that the statistical behaviour of the training data is representative of the general data that our algorithm will encounter, our algorithm can be designed to make decisions about general data.

Most engineering subjects are highly deterministic; there is a particular set of input values, you decide which is the best equation to use and you calculate the answer. While engineers may deal with uncertainty in some places (for example, tolerances in design, or the use of safety factors), a full use of statistical methods is usually not considered. A full understanding of statistics is often not seen as particularly useful[1]. However, it can be very powerful in many scenarios, and it is very useful for machine learning. This chapter will focus on statistical methods for machine learning, but will try not to cover unnecessary statistical detail.

2.1 Prerequisites

It is assumed that you have a basic knowledge of statistics. You should know what a probability distribution is, and about the Normal distribution. Some assistance was provided in the previous chapter about multi-variate Normal distributions; it is assumed that you understand this. You should also understand basic probability, including conditional probability, and the accompanying notation.

2.2 Bayesian decision theory

C2A

We will again consider the dye penetrant classification problem discussed in Sect. 1.4 for our example, where we have a particular indication visible in the image and we wish to make an assessment about whether or not this is an image of a defect. This brings us to our first definition (there will be many of these throughout this chapter), the STATE OF NATURE, represented by ω. We can define $\omega = \omega_1$ as the case that our indication is a defect, and $\omega = \omega_2$ as the case that our indication is not a defect. It is our goal to use machine learning to obtain the best possible estimation of the state of nature.

C2B

Probability is a key element of statistics, and we can have a probability associated with each of these states of nature, i.e. the probability that a particular indication is a defect is $P(\omega_1)$ and the probability that it's not a defect is $P(\omega_2)$. This probability is called the PRIOR, and describes the probability before an observation has occurred. Clearly these sum together to make 1, since there are no other possible states of nature. If we know these probabilities (and we could estimate them by simply counting cases for a training dataset) we could use them to make a decision rule,

[1] I won't get into a discussion about whether this is valid or not!

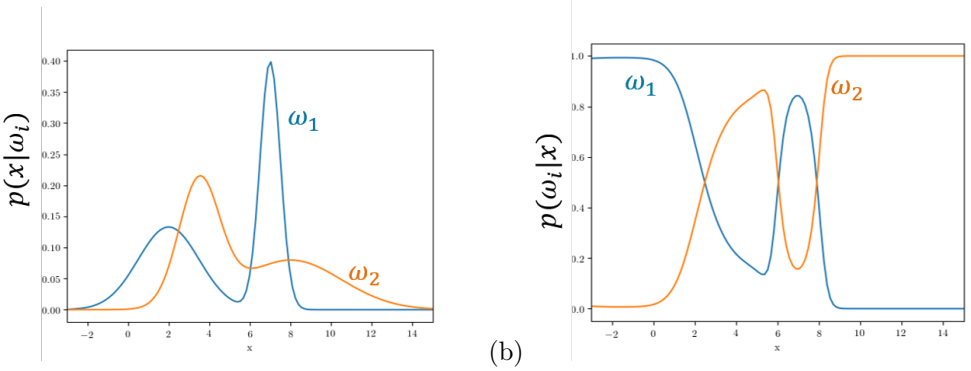


Figure 2.1: (a) Example plot of likelihood and (b) corresponding posterior.

to classify as ω_1 if $P(\omega_1) > P(\omega_2)$ otherwise decide it is ω_2. Have a think for a minute about the consequences of this and whether it is a sensible approach in practice.

Hopefully you have identified that your probabilities will be constant, i.e. there is no dependence on any observations. This means that you can look at any future indication and if your training set indicated that the majority of indications are not defects, all indications will be classed as non defects. This behaviour is clearly not desired. However, the concept of utilising probability is useful, but we do need make the probability dependent on the variables we get from an observation.

Let us consider one such variable for the dye penetrant problem: indication length. From our training dataset, we have two classes, and for each of those, we have a particular probability distribution as a function of length, which can be written as $P(x|\omega_j)$ where x can be any parameter (or set of parameters), but we will take x as length in this scenario. This is therefore the probability distribution of x given the particular state of nature ω_j, and this is called the LIKELIHOOD.

The likelihood is a step in the right direction, but on its own it is not particularly helpful. When we take a measurement we have the observation value x, and we want to know what state of nature caused that, or at least the probability that each state of nature caused that, such that we can perform a classification, which is the opposite to what the likelihood gives us: the probability of an observation given a state of nature. The desired probability is written $P(\omega_j|x)$ and is known as the POSTERIOR probability, the probability of a state of nature for a given observation. We could then employ a rule to classify as ω_1 if $P(\omega_1|x) > P(\omega_2|x)$ and ω_2 otherwise in a similar manner to what was proposed above, but now this would be dependent on the observation, x. C2C

Bayes' theorem is used to calculate the posterior probability. Basic probability rules give

$$P(\omega_j \cap x) = P(\omega_j|x)p(x) = P(x|\omega_j)p(\omega_j) \tag{2.1}$$

which can be rearranged to

$$P(\omega_j|x) = \frac{P(x|\omega_j)p(\omega_j)}{p(x)} \tag{2.2}$$

which enables us to calculate the posterior probability, assuming that the likelihood $P(x|\omega_j)$ is known, as well as the two probabilities $p(\omega_j)$ and $p(x)$[2]. An example showing the likelihood and corresponding posterior terms are shown in Figs. 2.1(a) and (b) respectively. C2D

The term $p(x)$ is known as the EVIDENCE and can be calculated as

$$p(x) = \sum_{j=1}^{J} P(x \cap \omega_j) = \sum_{j=1}^{J} P(x|\omega_j)p(\omega_j) \tag{2.3}$$

[2]It is noted here that this is expressed in terms of the probability density function for x, which is a continuous variable, rather than probabilities of a discrete variable. We will not go through the details on this course, but Bayes' law is valid as used here for continuous PDFs. Note that this case is mixed, i.e. $p(\omega_j)$ is a discrete probability (since ω_j corresponds to a discrete class) while $p(x)$ is the continuous probability density. It is also noted that the prior may be a PDF as well, in which case the output posterior will be a PDF rather than discrete probability.

from the likelihood and prior terms. This term is effectively a scaling term and ensures that the posterior probabilities sum to 1. If just comparing posterior probabilities, this term can be neglected.

When we compare the probabilities like this, we are minimising the error, i.e. getting $P(error|x) = min[P(\omega_1|x), P(\omega_2|x)]$. This is not always the most sensible thing to do. Section 1.4.4 highlighted a potential issue it is important to consider with machine learning, that of the expense (financial or otherwise) of making a particular decision, and what happens if this decision is incorrect. Scrapping a part which has no defect is much less expensive than having an accident because a defect was classified incorrectly; this asymmetry requires particular attention to account for. C2E

We introduce the definition of an ACTION, denoted α_i, where several actions are possible for each state of nature ω_j. As an example, if ω_1 is the state of finding a defect, action α_1 could be to bin the part, whereas α_2 may be to allow it into service. There are then LOSS FUNCTIONS associated with these, defined as $\lambda(\alpha_i|\omega_j)$, which describe the loss for each action taken depending on the underlying state of nature. Following our previous example, the loss $\lambda(\alpha_2|\omega_1)$, i.e. allowing a component with a defect into service, is potentially very high. RISK is then the expected loss[3] associated with taking a particular action, based on an observation x:

$$R(\alpha_i|x) = \sum_{j=1}^{J} \lambda(\alpha_i|\omega_j) P(\omega_j|x). \tag{2.4}$$

Note that to calculate risk we need a matrix of loss values $\lambda(\alpha_i|\omega_j)$ to cover the different combinations of actions and states of nature. C2F

We can consider the following example of this.

> If the cost of missing a defect is £1M, compared to binning a component with or without a defect of £1k, and there are no costs associated with keeping a non-defective component, what are the risks associated with the two actions if a measurement suggests there is a 5% chance it's a defect?

Let's start by making some definitions. ω_1 is the state that there's no defect, and ω_2 is the state that there is a defect. α_1 is the action to bin the part while α_2 is to keep it. Interpreting the loss terms from the question, we have

$$\lambda(\alpha_1|\omega_1) = 1000 \tag{2.5}$$
$$\lambda(\alpha_1|\omega_2) = 1000 \tag{2.6}$$
$$\lambda(\alpha_2|\omega_1) = 0 \tag{2.7}$$
$$\lambda(\alpha_2|\omega_2) = 1000000 \tag{2.8}$$

and the posterior probabilities are

$$P(\omega_1|x) = 0.95 \tag{2.9}$$
$$P(\omega_2|x) = 0.05 \tag{2.10}$$

The risks can then be calculated as

$$R(\alpha_1|x) = 1000 \times 0.95 + 1000 \times 0.05 = £1000 \tag{2.11}$$

and

$$R(\alpha_2|x) = 0 \times 0.95 + 1000000 \times 0.05 = £50000. \tag{2.12}$$

In the same way as minimising error, we can act to minimise risk, i.e. take action α_1 if $R(\alpha_1|x) < R(\alpha_2|x)$, otherwise take α_2.

2.3 Training a Bayesian learning algorithm

C2G

Previously we have assumed that we know the likelihood and prior probability distributions, but in reality these must be estimated from the training data. It is possible to provide a full estimation

[3] Remember from statistics that expected value is $\sum xP(x)$.

of these functions (and we will look at this in a later chapter) but it is simpler to assume the data conforms to a particular probability distribution and estimate parameters such as the mean and standard deviation for the training data.

The field of Bayesian learning is briefly mentioned at this point, but will not be covered on this course. Under this model, the parameters mentioned are themselves considered to be random variables conforming to particular distributions. Adding more training samples then "sharpens" the distribution around a particular mean, indicating more strongly that the unknown true value lies there. C2H

Instead we will consider the Maximum Likelihood approach, which is conceptually simpler. In this each of the parameters (mean, standard deviation) has a single unknown value and we will try to estimate these. If we have samples taken from a state of nature ω_j then we can assume that $p(\boldsymbol{x}|\omega_j)$ has a known parametric form, such as the Normal distribution[4], $N(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$. We wish to estimate the parameters $\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j$ and we can combine them together into a single general set of parameters $\boldsymbol{\theta}$, which we then wish to determine. To do this, we treat each class separately, using the assumption that a sample falling into one class says nothing about the other classes, so, given a set of training samples drawn from a distribution based on $\boldsymbol{\theta}$, what is $\boldsymbol{\theta}$? Or which value $\boldsymbol{\theta}$ gives the highest probability that this dataset would be selected?

Say that the data set D contains n samples $\boldsymbol{x}_1$, ..., $\boldsymbol{x}_n$, the likelihood of $\boldsymbol{\theta}$ with respect to the samples is given by

$$P(D|\boldsymbol{\theta}) = \prod_{k=1}^{n} P(\boldsymbol{x}_k|\boldsymbol{\theta}). \tag{2.13}$$

Let us briefly examine what this equation means, starting with the left hand side term $P(D|\boldsymbol{\theta})$. We have a probability distribution which is described by the knowledge of the underlying function (e.g. "it's a normal distribution") and the parameters contained in $\boldsymbol{\theta}$, and we then take the probability that our particular dataset D is produced from this distribution as $P(D|\boldsymbol{\theta})$; the literal words behind this term would be: "the probability that dataset D is produced given that the distribution described by $\boldsymbol{\theta}$ is used".

Moving to the right hand side, the probability that an individual data point $\boldsymbol{x}_k$ would come from the dataset is described as $P(\boldsymbol{x}_k|\boldsymbol{\theta})$. Combining these together, we can assume they have been independently sampled, and therefore a product of the probabilities can be used to calculate the union of all the terms. The symbol $\prod$, if you are not familiar, means take the product of all the terms (in the same way that $\sum$ corresponds to a summation).

We define our target value to be $\hat{\boldsymbol{\theta}}$, which is the value to maximise $P(D|\boldsymbol{\theta})$. Note the definition of likelihood means that we maximise the probability of producing our dataset D given $\boldsymbol{\theta}$ by varying $\boldsymbol{\theta}$, which is subtly different from maximising the posterior, $P(\boldsymbol{\theta}|D)$, which is the probability of our parameter set being $\boldsymbol{\theta}$ given the dataset D. From earlier in the chapter you will know that the difference is a factor of $p(\boldsymbol{\theta})$, i.e. the inclusion of the prior, and in practice this often makes little difference, so we focus on maximising the simpler likelihood.

To simplify some of the maths, we will maximise the log likelihood

$$l(\boldsymbol{\theta}) = \ln P(D|\boldsymbol{\theta}) = \sum_{k=1}^{n} \ln P(\boldsymbol{x}_k|\boldsymbol{\theta}). \tag{2.14}$$

This works because the log function is monotonically increasing, so a maximum in the input will be a maximum in the output. We will maximise this by setting the differential to zero (we will assume without proof that this is a maximum rather than a minimum; the Normal distribution is convenient in that it is well behaved in this regard). Differentiating with respect to one of the parameters θ_p gives

$$\frac{\partial l}{\partial \theta_p} = \sum_{k=1}^{n} \frac{\partial}{\partial \theta_p} \ln P(\boldsymbol{x}_k|\boldsymbol{\theta}) \tag{2.15}$$

[4] As mentioned previously, we will only consider the Normal distribution on this course; this is applicable to a wide range of engineering problems. The central limit theorem states that the sum of many random variables (regardless of their distributions) will tend towards a normal distribution, and many engineering processes could be considered as such sums. Crack growth could be such an example, leading to crack lengths having such a distribution. The use of the Normal distribution is not without its issues though; Chapter 10 discusses some of these.

which should be zero.

To take this further we need to consider the form of the normal distribution. The univariate (single variable) case with mean μ and standard deviation σ, written as a function of x, can be taken from the literature as

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]. \tag{2.16}$$

This extends into d dimensions as

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{d/2}\left|\boldsymbol{\Sigma}\right|^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^t \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right] \tag{2.17}$$

where $\boldsymbol{x}$ is a d-component column vector of data values, $\boldsymbol{\mu}$ is the d-component vector of mean values, $\boldsymbol{\Sigma}$ is the d-by-d covariance matrix, $|\boldsymbol{\Sigma}|$ and $\boldsymbol{\Sigma}^{-1}$ are its determinant and inverse respectively and ^{t} represents the transpose. C2I

Let us consider the case where we have a single parameter problem and wish to estimate both the mean ($\theta_1 = \mu$) and the standard deviation/variance ($\theta_2 = \sigma^2$). The equation then can be written as

$$P(x_k|\boldsymbol{\theta}) = (2\pi\theta_2)^{-\frac{1}{2}} \exp\left[-\frac{1}{2\theta_2}\left(x_k-\theta_1\right)^2\right]. \tag{2.18}$$

Simplifying through the use of ln, noting standard log rules that products become sums and power terms will be multiplied (bringing the $-\frac{1}{2}$ term outside the ln)

$$\ln P(x_k|\boldsymbol{\theta}) = -\frac{1}{2}\ln\left(2\pi\theta_2\right) - \frac{1}{2\theta_2}\left(x_k-\theta_1\right)^2 \tag{2.19}$$

or to split out the constant 2π term

$$\ln P(x_k|\boldsymbol{\theta}) = -\frac{1}{2}\ln\left(2\pi\right) - \frac{1}{2}\ln\left(\theta_2\right) - \frac{1}{2\theta_2}\left(x_k-\theta_1\right)^2. \tag{2.20}$$

Then we differentiate, with the first row as $\frac{\partial l}{\partial \theta_1}$ and the second as $\frac{\partial l}{\partial \theta_2}$:

$$\frac{\partial l}{\partial \boldsymbol{\theta}} = \sum_{k=1}^{n}\begin{bmatrix} \frac{1}{\theta_2}\left(x_k-\theta_1\right) \\ -\frac{1}{2\theta_2} + \frac{(x_k-\theta_1)^2}{2\theta_2^2} \end{bmatrix}. \tag{2.21}$$

We set both of these to zero. Considering the first row first:

$$\sum_{k=1}^{n}\frac{1}{\hat{\theta}_2}\left(x_k-\hat{\theta}_1\right) = 0 \tag{2.22}$$

Assuming $\hat{\theta}_2$ is not infinite, $\frac{1}{\hat{\theta}_2}$ can be dropped and then the summation expanded:

$$\begin{aligned} \sum_{k=1}^{n} x_k - n\hat{\theta}_1 &= 0 && (2.23)\\ n\hat{\theta}_1 &= \sum_{k=1}^{n} x_k && (2.24)\\ \hat{\theta}_1 &= \frac{1}{n}\sum_{k=1}^{n} x_k. && (2.25) \end{aligned}$$

Since we have defined $\hat{\theta}_1 = \hat{\mu}$, this means that the maximum likelihood estimate of the mean of the underlying distribution is the same as the mean of the sample set. Now consider the second row

$$\sum_{k=1}^{n} -\frac{1}{2\hat{\theta}_2} + \frac{\left(x_k-\hat{\theta}_1\right)^2}{2\hat{\theta}_2^2} = 0$$

multiplying all terms by $2\hat{\theta}_2^2$ will eliminate the denominator

$$\sum_{k=1}^{n} -\hat{\theta}_2 + \left(x_k - \hat{\theta}_1\right)^2 = 0$$

so

$$\hat{\theta}_2 = \frac{1}{n}\sum_{k=1}^{n}\left(x_k - \hat{\theta}_1\right)^2 \tag{2.26}$$

or

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{k=1}^{n}\left(x_k - \hat{\mu}\right)^2. \tag{2.27}$$

This means that the maximum likelihood estimate of the standard deviation of the underlying distribution is the same as that of the sample set, matching the mean. This can also be written[5]

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{k=1}^{n} x_k^2 - \hat{\mu}^2. \tag{2.28}$$

Note that some people often question the $1/n$ factor rather than the $1/(n-1)$ which many associate with the population estimate from a sample of size n. Note that we are calculating something subtlely different here: we're not estimating the population statistics from a sample dataset, but rather estimating the distribution which would have produced our dataset. In practice, because we typically deal with very large datasets in machine learning, the difference becomes negligible.

Although we have demonstrated this with a fairly simple example, maximum likelihood can be more broadly applied to multi-parameter cases and with other distributions. It should be noted that while exactly the same principles apply, the mathematical equations become much longer when having to differentiate matrix multiplications, and we will usually end up with simultaneous equations in the various θ parameters, so we will not cover the maths for this.

2.4 Summary

From this chapter you should know how to calculate the posterior and hence classify data points. You should be able to extract distribution parameters for a given training dataset assuming a normal distribution, and you should understand how these parameters define the likelihood and can therefore be used to obtain classification decisions.

In this chapter we have met many new definitions. These are summarised below.

- State of nature, ω_j
  - The class or underlying value we are trying to estimate with machine learning
- Prior probability, $P(\omega_j)$
  - Probability of a particular state of nature in the absence of any observations
- Posterior probability, $P(\omega_j|x)$
  - Probability of a particular state of nature, given a specific observation
- Likelihood, $P(x|\omega_j)$
  - Probability of an observation occurring given a particular state of nature
- Evidence, $P(x)$
  - Probability of observation occurring not accounting for state of nature

[5] $\frac{1}{n}\sum_{k=1}^{n}(x_k-\hat{\mu})^2 = \frac{1}{n}\sum_{k=1}^{n}(x_k^2 - 2x_k\hat{\mu} + \hat{\mu}^2) = \frac{1}{n}\sum_{k=1}^{n}x_k^2 - 2\frac{\hat{\mu}}{n}\sum_{k=1}^{n}x_k + \hat{\mu}^2 = \frac{1}{n}\sum_{k=1}^{n}x_k^2 - 2\hat{\mu}^2 + \hat{\mu}^2 = \frac{1}{n}\sum_{k=1}^{n}x_k^2 - \hat{\mu}^2$.

- Action, α_i
  - Anything which can be undertaken in response to an observation, such as discarding or keeping a part
- Loss, $\lambda(\alpha_i|\omega_j)$
  - The cost (possibly financial, but not necessarily) associated with taking a particular action for a particular state of nature
- Risk, $R(\alpha_i|x)$
  - The expected loss, based on the various underlying states of nature which would be expected for a specific action

2.5 Questions

1. I have a two parameter problem in (x_1, x_2). Taking just 3 samples: (1, -3), (0, 1) and (2, 1), what is the mean and the covariance matrix of the Normal distribution, under the maximum likelihood approach, which fits this best? [NB do not spend long on this question - it is mainly to get you thinking. See solution for discussion of this.]

2. My company has been making carbon fibre components and I have found that 1 in 10 are breaking. I am aware that there are variations in our production process, and I wish to use information from this to predict failure. In particular I wish to use the temperature at which the carbon fibre has been cured. Below is a representative sample of the curing temperatures for components which did not break:
   252.9 212.0 229.4 267.2 256.0 170.7 228.5 195.5 196.9 212.3 204.3 243.6 222.8 203.7 213.3 210.0 244.8 193.8 209.4 174.4 123.4 219.6 225.9 177.7 268.1 156.4 201.4 194.4 246.0 244.1 204.6 211.3 173.4 140.6 189.6 204.7 236.9 236.1 188.4 190.9 168.5 157.4 148.8 258.5 184.7 186.9 162.4 223.3 151.6 193.6 173.1 211.6 184.7 164.6 199.2 212.8 202.0 209.1 181.0 189.1 179.8 189.2 175.6 148.2 205.3 187.9 151.1 213.9 172.8 201.6 221.9 203.9 234.2 163.0 212.1 179.5 173.9 182.6 190.7 201.7 165.0 227.0 214.0 153.9 244.6 256.9 235.4 194.6 167.9 231.6 187.9 236.7 206.2 229.3 210.7 221.2 200.3 253.6 203.8 212.1
   The following values are derived from this dataset and you may find these useful:
   Sum: 20179.5, Sum of squares: 4163531.8, Median: 202.8, Number of points: 100
   Below is a representative sample of the curing temperatures of a sample of components which broke:
   334.7 189.4 192.8 293.6 197.2 337.5 231.4 216.4 336.5 316.6 334.0 290.8 211.2 336.0 237.9 286.1 292.6 243.0 277.6 291.5 266.9 200.5 263.4 309.7 218.7 243.3 230.4 333.2 280.3 268.3 215.4 274.3 219.7 251.4 221.4 280.4 275.9 240.6 267.8 200.8 182.9 269.8 257.5 278.6 357.2 292.5 208.9 300.3 190.8 229.2 246.9 327.1 216.5 212.8 245.6 220.1 300.7 201.4 198.4 230.3 227.6 336.8 292.7 253.9 194.9 288.0 205.0 180.5 303.5 264.3 291.4 264.3 288.6 220.7 203.5 280.7 213.8 219.0 229.5 250.8 234.1 188.1 221.0 149.9 278.1 177.9 200.3 252.3 216.7 319.4 191.8 262.0 248.2 197.4 273.5 242.3 284.7 287.1 347.3 310.1
   The following values are derived from this dataset and you may find these useful:
   Sum: 25368.4, Sum of squares: 6652312.5, Median: 251.1, Number of points: 100
   All values are in °C (and $(°C)^2$ for the sum of squares).
   Assuming that both of the probability distributions for breaking and non-breaking cases can be well captured by a normal distribution, if I take a component where the curing temperature was 240 °C, estimate the probability that the component will break.

3. I sample data for a two class problem. I estimate data within the first class as having a probability distribution of $1 - \cos \pi x$ in the range $0 < x < 1$ and zero outside, and data within the second class as being uniform in the same range and zero outside. The probability of any data being in class 1 is 0.6. If I have a sampled point at a position x in the range $0 < x < 1$, what is the probability that it comes from the first class?

2.5.1 Solutions

1. In theory you should be setting this up as a 5 parameter problem (2 means, 2 variances, 1 covariance) and solving in the same way we did for the single parameter case. You should discover this is very difficult to do. You can make an assumption following the single parameter case that the parameters correspond to those calculated just for the sample, i.e., calculate the covariance matrix using the formula given in Chapter 1.

2. (Q1 2021) Estimate mean and standard deviations for both distributions using maximum likelihood. Use sum and sum of squares from question.

$$mu_1 = \frac{sum_1}{n} = 201.8, \qquad mu_2 = \frac{sum_2}{n} = 253.7$$

$$std_1 = \sqrt{\frac{ssq_1}{n} - mu_1^2} = 30.23, \qquad std_2 = 46.56$$

$$p(\omega_1) = 0.9 \qquad p(\omega_2) = 0.1$$

$$p(x|\omega_1) = \frac{1}{\sqrt{2\pi}\sigma_1} \exp\left(-\left(\frac{x-\mu_1}{\sigma_1}\right)^2/2\right) = 0.00594$$

$$p(x|\omega_2) = 0.00821$$

$$p(x) = p(x|\omega_1)p(\omega_1) + p(x|\omega_2)p(\omega_2) = 0.00617$$

$$\text{So}: \quad p(\omega_2|x) = \frac{p(x|\omega_2)p(\omega_2)}{p(x)} = \mathbf{0.1331}$$

3. (Q4(c) 2020)

$$p(x|\omega_1) = 1 - \cos(\pi x)$$

$$p(x|\omega_2) = 1$$

$$p(\omega_1|x) = \frac{p(x|\omega_1)p(\omega_1)}{p(x)}$$

$$\begin{aligned} p(x) &= p(x|\omega_1)p(\omega_1) + p(x|\omega_2)p(\omega_2) \\ &= (1 - \cos(\pi x)) \cdot 0.6 + 1 \cdot 0.4 \\ &= 1 - 0.6 \cdot \cos(\pi x) \end{aligned}$$

$$p(\omega_1|x) = \frac{(1 - \cos(\pi x)) \cdot 0.6}{1 - 0.6 \cdot \cos(\pi x)}$$

Chapter 3

Regression

C3A

Regression is one area of Machine Learning which many students will have met before, even if it was not described as such. So far we have looked at classification; based on data, which discrete class (often of two, but can be any number) does the item fall into. Now we instead consider the case where the output is a continuous value. It could be that we want to estimate time to failure given a particular heat treatment time, or any number of examples. These can include stress-strain curves, shear behaviour in a viscous fluid, coefficient of friction, X-ray attenuation, heat transfer from a surface. A lot of engineering relies on producing a continuous numerical output[1], so regression is a powerful tool.

3.1 Linear regression

C3B

In classification, each sample consisted of a set of input parameters $\boldsymbol{x} = (x_1, x_2 ... x_n)$ and an output class (or possibly a set of classes) y. The distinction with regression is that the output y is now a continuous variable. To begin with we will consider a 1D input and 1D output, with a set of m training points (x_1, y_1),(x_2, y_2)...(x_m, y_m). We wish to approximate this with a linear function $f(x) = \beta_1 x + \beta_2$, where our goal is to find the constants β_1 and β_2 which best approximate the training data.

To do this, we can insert our training points into the linear function to give (for the ith point):

$$f(x_i) = \beta_1 + \beta_2 x_i = y_i + E_i \tag{3.1}$$

where we have introduced an error term E_i. To get the best fit, we wish to minimise this error; since we need to do this simultaneously for all these terms, we want to find one error metric which combines all of these independent metrics. Section 11.4 discusses metrics in full, but here the specific use of the common metrics used as a measure of error to be minimised are discussed.

3.1.1 Error metrics

C3C

One approach is to square each term then sum, then take the square root - this is the least squares error, known as the l_2 error metric:

$$E_2 = \sqrt{\sum_{i=1}^{m} [f(x_i) - y_i]^2}. \tag{3.2}$$

This is one of the most common error metrics and is used widely across engineering and elsewhere for all sorts of applications. The approach matches Pythagorean rules on distances (travel distance x in the x direction, and y in the y direction and you are $\sqrt{x^2 + y^2}$ from your starting point) so this is a logical one to select if the error components are distances.

[1] In fact, it is a struggle to think of many discrete values produced within engineering

There are alternative measures though, such as the l_1 norm

$$E_1 = \sum_{i=1}^{m} |f(x_i) - y_i| \,. \tag{3.3}$$

This is a sum of the distances in each direction. This is often called the 'Manhattan' distance because the distance is the same that would be travelled between two points on a grid system, where one could only travel in the x or y directions. An alternative is the maximum error, the l_∞ norm

$$E_\infty = \max |f(x_i) - y_i| \,. \tag{3.4}$$

This will clearly be very sensitive to the largest error (and completely insensitive to everything else). These norms can be generalised as the l_p norm:

$$E_p = \left(\sum_{i=1}^{m} |f(x_i) - y_i|^p \right)^{1/p} \,. \tag{3.5}$$

It should be straightforward to see how this describes both the l_1 and l_2 norms; have a look and see if you can satisfy yourself that it also describes l_∞.

Which is best does depend on the problem. We should also consider a number of aspects, including how easy it is to train and how fast, how good the model is at fitting the training data, how well it generalises, how sensitive it is to errors and how well it performs with limited training data. Throughout this course we will focus on the l_2 (least squares) error. This is often algebraically neat, although can be more sensitive to outliers than the l_1 error. When using this, since our goal is simply to minimise it, and since this process will be unaffected the square root we usually simplify it to

$$E_2 = \sum_{i=1}^{m} [f(x_i) - y_i]^2 \,. \tag{3.6}$$

3.1.2 Minimising the error

C3D

Start from the above definition of error and substitute in the linear function $f(x) = \beta_1 + \beta_2 x$

$$E_2 = \sum_{i=1}^{m} [\beta_1 + \beta_2 x_i - y_i]^2 \tag{3.7}$$

To minimise this we calculate the derivatives with respect to our parameters β_1 and β_2 and set these to zero

$$\frac{\partial E_2}{\partial \beta_1} = \sum_{i=1}^{m} 2 \left[\beta_1 + \beta_2 x_i - y_i\right] = 0 \tag{3.8}$$

$$\frac{\partial E_2}{\partial \beta_2} = \sum_{i=1}^{m} 2 \left[\beta_1 + \beta_2 x_i - y_i\right] x_i = 0 \tag{3.9}$$

Multiplying these out

$$m\beta_1 + \beta_2 \sum_{i=1}^{m} x_i - \sum_{i=1}^{m} y_i = 0 \tag{3.10}$$

$$\beta_1 \sum_{i=1}^{m} x_i + \beta_2 \sum_{i=1}^{m} x_i^2 - \sum_{i=1}^{m} y_i x_i = 0 \tag{3.11}$$

which, as a linear set of simultaneous equations, can be expressed as a matrix

$$\begin{pmatrix} \sum x_i & \sum x_i^2 \\ m & \sum x_i \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix} = \begin{pmatrix} \sum y_i x_i \\ \sum y_i \end{pmatrix} \tag{3.12}$$

where for conciseness we have simplified $\sum_{i=1}^{m}$ to $\sum$. This can be expressed in matrix form as

$$\boldsymbol{A}\boldsymbol{\beta} = \boldsymbol{b} \tag{3.13}$$

with $\boldsymbol{A}$, $\boldsymbol{\beta}$ and $\boldsymbol{b}$ defined appropriately. Any number of techniques can be used to invert to find $\boldsymbol{\beta}$.

3.1.3 Multi-dimensional data

C3E

So far we have considered a function of a single variable. In many cases, however, the underlying function could be one of multiple parameters, i.e. $f(\boldsymbol{x})$, for example, shear stress in a fluid is related to the shear strain rate, but temperature is also a parameter. The resulting function (for a linear case) can be expressed as $f(\boldsymbol{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n$. The error to be minimised becomes $E_2 = \sum_{i=1}^{m} \left[f(\boldsymbol{x}_i, \boldsymbol{\beta}) - y_i\right]^2$, which can be achieved in a similar way to before. Writing this out for a three-parameter case (β coefficients in reverse for ease later):

$$\frac{\partial E_2}{\partial \beta_2} = \sum_{i=1}^{m} 2\left[\beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} - y_i\right] x_{2i} = 0 \tag{3.14}$$

$$\frac{\partial E_2}{\partial \beta_1} = \sum_{i=1}^{m} 2\left[\beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} - y_i\right] x_{1i} = 0 \tag{3.15}$$

$$\frac{\partial E_2}{\partial \beta_0} = \sum_{i=1}^{m} 2\left[\beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} - y_i\right] = 0 \tag{3.16}$$

Multiplying out the terms gives:

$$\beta_0 \sum x_{2i} + \beta_1 \sum x_{1i} x_{2i} + \beta_2 \sum x_{2i}^2 - \sum x_{2i} y_i = 0 \tag{3.17}$$

$$\beta_0 \sum x_{1i} + \beta_1 \sum x_{1i}^2 + \beta_2 \sum x_{1i} x_{2i} - \sum x_{1i} y_i = 0 \tag{3.18}$$

$$m\beta_0 + \beta_1 \sum x_{1i} + \beta_2 \sum x_{2i} - \sum y_i = 0 \tag{3.19}$$

And writing in matrix form leads to:

$$\begin{pmatrix} \Sigma x_{2i} & \Sigma x_{1i} x_{2i} & \Sigma x_{2i}^2 \\ \Sigma x_{1i} & \Sigma x_{1i}^2 & \Sigma x_{1i} x_{2i} \\ m & \Sigma x_{1i} & \Sigma x_{2i} \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{pmatrix} = \begin{pmatrix} \Sigma x_{2i} y_i \\ \Sigma x_{1i} y_i \\ \Sigma y_i \end{pmatrix}. \tag{3.20}$$

3.2 Higher order polynomials

C3F

We can extend this to quadratic function fitting using $f(x) = \beta_1 + \beta_2 x + \beta_3 x^2$. Then (doing β coefficients in reverse for ease later)

$$\frac{\partial E_2}{\partial \beta_3} = \sum_{i=1}^{m} 2\left[\beta_1 + \beta_2 x_i + \beta_3 x_i^2 - y_i\right] x_i^2 = 0 \tag{3.21}$$

$$\frac{\partial E_2}{\partial \beta_2} = \sum_{i=1}^{m} 2\left[\beta_1 + \beta_2 x_i + \beta_3 x_i^2 - y_i\right] x_i = 0 \tag{3.22}$$

$$\frac{\partial E_2}{\partial \beta_1} = \sum_{i=1}^{m} 2\left[\beta_1 + \beta_2 x_i + \beta_3 x_i^2 - y_i\right] = 0 \tag{3.23}$$

Leading to

$$\beta_1 \sum x_i^2 + \beta_2 \sum x_i^3 + \beta_3 \sum x_i^4 - \sum x_i^2 y_i = 0 \tag{3.24}$$

$$\beta_1 \sum x_i + \beta_2 \sum x_i^2 + \beta_3 \sum x_i^3 - \sum x_i y_i = 0 \tag{3.25}$$

$$m\beta_1 + \beta_2 \sum x_i + \beta_3 \sum x_i^2 - \sum y_i = 0 \tag{3.26}$$

This can be written as before in matrix form:

$$\begin{pmatrix} \Sigma x_i^2 & \Sigma x_i^3 & \Sigma x_i^4 \\ \Sigma x_i & \Sigma x_i^2 & \Sigma x_i^3 \\ m & \Sigma x_i & \Sigma x_i^2 \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix} = \begin{pmatrix} \Sigma x_i^2 y_i \\ \Sigma x_i y_i \\ \Sigma y_i \end{pmatrix} \tag{3.27}$$

This generalises to polynomials of degree D with a $(D+1)\times(D+1)$ system of equations, following the same form.

3.2.1 Expressing higher order polynomials as multi-parameter problems

C3G

Consider $f(x) = \beta_1 + \beta_2 x + \beta_3 x^2$. We can change this into a multi-parameter problem $f(x) = \beta_1 + \beta_2 x_1' + \beta_3 x_2'$ where the parameter $\boldsymbol{x}' = (x, x^2)$. The step to exploit this would be to calculate the new parameter values, then fit these using the linear approach. In the simple case given here, it is trivial to use the derived $\boldsymbol{\beta}$ values directly, but in general, when using this on general data, we would need to calculate the new parameter values for this data then apply the coefficients to these.

By comparing eqs. (3.20) and (3.27), it is clear that there are significant similarities. There is no computational benefit from this approach, but it is important to be aware of this method because it is commonly applied, particularly in the Scikit-learn toolbox as you will see in the tutorial.

3.3 Fitting exponentials

C3H

We may suspect that the function $f(x) = \beta_1 e^{\beta_2 x}$ would fit a particular dataset better for some engineering application. This would lead to the following equations

$$\beta_1 \sum_{i=1}^{m} x_i e^{2\beta_2 x_i} - \sum_{i=1}^{m} x_i y_i e^{\beta_2 x_i} = 0 \tag{3.28}$$

$$\beta_1 \sum_{i=1}^{m} e^{2\beta_2 x_i} - \sum_{i=1}^{m} y_i e^{\beta_2 x_i} = 0 \tag{3.29}$$

which are nonlinear, i.e. cannot be rewritten in a linear matrix equation. Solving these directly is not straightforward.

Instead we can apply logs to both sides of $y = \beta_1 e^{\beta_2 x}$, giving

$$\ln y = \ln \beta_1 + \beta_2 x \tag{3.30}$$

which is a linear equation in parameters $\ln \beta_1$ and β_2 (although note that you may need to write these out in terms of some new variables). If it is possible to find such a transform, then standard linear regression approaches as described above are suitable.

It should be noted that the error term will be formulated in the transformed space, not the original space. i.e. in

$$\ln y_i + E_i' = \ln \beta_1 + \beta_2 x_i \tag{3.31}$$

and

$$y_i + E_i = \beta_1 e^{\beta_2 x_i} \tag{3.32}$$

the two error terms are not equal, i.e. $E_i \neq E_i'$. We are doing least squares in the transformed space, not the original one.

3.4 Fitting non-linear functions and gradient descent

C3I

In many cases it is not possible to perform linear fitting, or find a transform which would allow linear fitting to be performed. In this scenario we can express the fitting function as

$$f(x) = f(x, \boldsymbol{\beta}) \tag{3.33}$$

to show dependence on parameters $\boldsymbol{\beta}$ which describe the non-linear functions. The error to be minimised is then

$$E_2(\boldsymbol{\beta}) = \sum \left[f(x_i, \boldsymbol{\beta}) - y_i \right]^2 . \tag{3.34}$$

In general, taking

$$\frac{\partial E_2}{\partial \beta_j} = 0 \tag{3.35}$$

will result in intractable non-linear equations. There may be multiple (even infinite numbers of) solutions, or no solutions at all.

Gradient descent methods are popular numerical tools for solving these problems. A starting point is selected, then the gradient of the error function E_2 is calculated. A small step is taken in the opposite direction (i.e. to minimise E_2). This is then repeated until the gradient becomes flat. One specific clear issue with gradient descent is that one can become stuck in local minima, so we therefore need to start close to the global minimum.

For illustration purposes we will define an example error function as $e(x_1, x_2) = 3x_1^2 + x_2^2$, and we will start at point $(2, 2)$ and try to find the minimum. The derivatives can be calculated as

$$\frac{\partial e}{\partial x_1} = 6x_1 \tag{3.36}$$

$$\frac{\partial e}{\partial x_2} = 2x_2 \tag{3.37}$$

In this case, clearly there is a trivial solution where the gradient becomes zero ($x_1 = x_2 = 0$), so we do not need to use gradient descent, however, we will do so for illustrative purposes. Expressing these two derivatives as a combined vector, $\boldsymbol{\nabla} e = (6x_1, 2x_2)$, we must now step in the opposite direction, i.e. at each step do the following

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \delta \boldsymbol{\nabla} e(\boldsymbol{x}_k). \tag{3.38}$$

For this simple case of determining two parameters, we can plot the resulting path of descent in the 2D space, along with the underlying function. Figure 3.1 shows this for four values of step length.

In Fig. 3.1(a) we can see that the descent step has not converged. It will do, but the step size ($\delta = 0.03$) is too small and it will take a long time to do this. By comparison, Fig. 3.1(b) has a larger step size ($\delta = 0.1$) and convergence is achieved within the 8 steps. Increasing the step size further to $\delta = 0.3$, as shown in Fig. 3.1(c), we can see that we are overshooting the value, which is undesired behaviour, although convergence is still occurring overall. Finally we have a slightly larger step size of $\delta = 0.4$, and we can see that it no longer converges; the step size is too large and at each step we overshoot and move further away from the true value.

Selecting the step size is a significant challenge of gradient descent methods. In many cases trial and error is used, or experience from similar problems, to establish something suitable. There are line search algorithms which can repeatedly sample the underlying function to minimise it, however, this can be computationally expensive to do since the calculation must be performed each time. It is often better to pick a fixed step which is smaller than optimal and iterate rather than recalculate.

In some cases it may not be possible to calculate the gradient analytically (although in many cases it is). Instead we may calculate this numerically, using a finite difference approximation, i.e. for each parameter β_j, step a small distance δ_{β_j} in that parameter direction and note difference in the underlying function. The gradient can then be approximated as

$$\frac{\partial E_2}{\partial \beta_j} \approx \frac{E_2(\beta_j + \delta_{\beta_j}) - E_2(\beta_j)}{\delta_{\beta_j}}, \tag{3.39}$$

which is a truncation of the Taylor series. Note that you can also calculate this as

$$\frac{\partial E_2}{\partial \beta_j} \approx \frac{E_2(\beta_j + \delta_{\beta_j}/2) - E_2(\beta_j - \delta_{\beta_j}/2)}{\delta_{\beta_j}}, \tag{3.40}$$

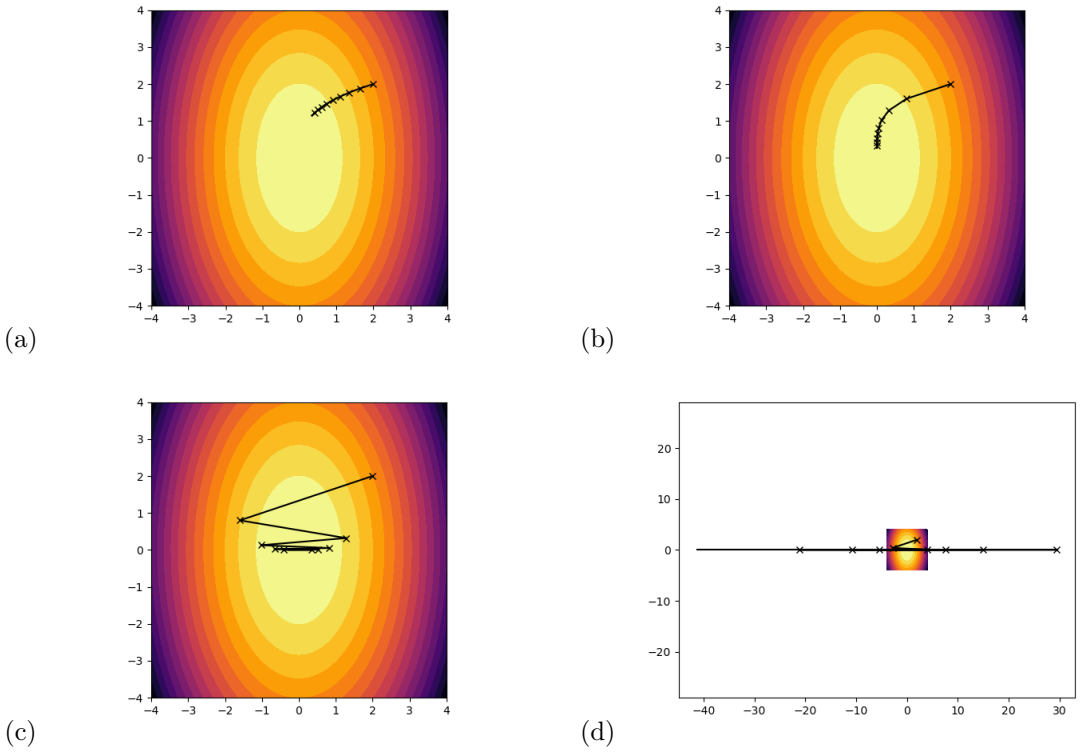


Figure 3.1: Examples of gradient descent with different step sizes. In (a) $\delta = 0.03$, in (b) $\delta = 0.1$ in (c) $\delta = 0.3$ and in (d) $\delta = 0.4$. In all cases 8 steps were taken. Note that in (d) the scale has been adjusted because the point is moving outside the main region.

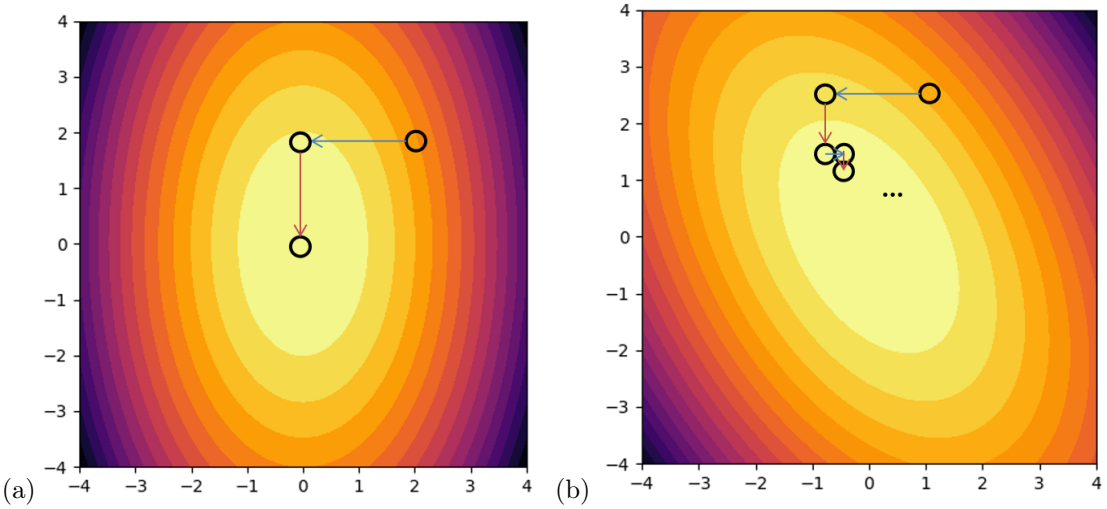


Figure 3.2: Alternating descent algorithm, applied to a simple 2D function. (a) shows a straightforward scenario, while (b) presents an example where alternating descent performs less well.

which is centralised at $E_2(\beta_j)$ and may be slightly more accurate; however, it is unlikely this will have huge impact. It should be noted that this is quite an inefficient approach; it is necessary to calculate the underlying function repeatedly for each parameter (and repeat this at every step of the descent routine), so this is expensive for multi-parameter cases and analytical gradient calculation is often much preferred. It should be noted that the finite difference approximation here can be very valuable to verify analytical gradient calculation.

3.4.1 Alternating descent

C3J

Another, related, approach is alternating descent. In this case, each parameter is minimised in turn. This is often efficient for very simple scenarios. Figure 3.2(a) shows the result for a straightforward gradient descent approach example; this is for the case $e(x_1, x_2) = 3x_1^2 + x_2^2$ from before, and it should be noted that the minimum in each variable is independent of the other variable as shown in eqs. (3.36) and (3.37). By contrast, in Fig. 3.2(b) the plot is rotated. Now the descent approach is not as straightforward, and many steps are required. Alternating descent works best for simple cases, ideally where the minimum for each parameter is independent of the other parameters.

3.5 Ridge regularisation

C3K

One issue that all machine learning approaches need to be aware of is overfitting, i.e. fitting too closely to the training dataset and therefore not having good generalisation performance. There are various approaches which can be used to address this. The first is limiting the number of parameters in the model – in the case of regression this could be limiting the number of polynomial coefficients which can be varied – but this could mean that important true behaviour is missed. Another option, when performing an iterative fitting approach like gradient descent, is not to let the routine fully converge. However, this is also not ideal; how do we know when to stop?

Regularisation is a commonly used approach to address this. It appears in many areas of engineering, including inversion and optimisation, and it is also very important in fitting machine learning models. Regularisation involves penalising particular solutions which have high complexity. In the case of regression, we know that having more parameters means more complexity, but it is also more complex if these parameters are large. Therefore, we would prefer solutions where the parameters are small. We could rewrite our error value as

$$E_2 = \sum \left(\beta_1 + \beta_2 x_i + \beta_3 x_i^2 + \ldots - y_i\right)^2 + \lambda \sum \beta_j^2$$

where we are now increasing the error term when we have large values of the parameters β_j, and λ is a scalar value indicating the relative importance of this term. This is called ridge regularisation[2].

Selecting λ is a trade-off between bias and variance, which are discussed in Sect. 11.1. A low value leads to overfitting, which means a low bias but a high variance. A high value means the opposite; the model is not well fitted to the training dataset and we have high bias, but low variance. In order to select a suitable value of λ (or any similar weighting function for a regularisation term) it is necessary to perform validation studies. Section 11.2 discusses validation approaches.

3.6 Conclusions

This chapter has introduced regression, fitting a continuous output to a set of input data (in contrast to classification, where the output is discrete). Regression is one of the key approaches of machine learning, and particularly relevant to engineering. It should also be highlighted that regression approaches can always be used for classification, by defining a specific threshold on the output. The opposite is true in many cases too, with the majority of classification models having the flexibility to produce continuous outputs instead.

3.7 Questions

1. Mathematically show that the l_∞ metric is the same as taking the maximum error.

2. Analytically calculate the gradient of $x^4 - 4$, then compare to the numerical approximation around $x = 3$ with a step of 1×10^{-2} (use the central approach from eq. (3.40)). What happens when you now consider $x^2 - 4$ instead? Can you explain why this is happens?

3. I have the following points to which I wish to fit a second-order polynomial:

x_1	x_2	y
5.3	8.3	6.2
2.1	2.8	5.6
3.9	5.5	7.2
1.8	2.3	3.4

   (a) We wish to do this fitting by adding additional parameters then fitting a linear function to all of these parameters. Write out algebraically what these additional parameters are.

   (b) Now assume we wish to just fit a linear function to the data, i.e. we just have the parameters x_1 and x_2. Take the linear function as $y = \beta_1 + \beta_2 x_1 + \beta_3 x_2$. Define a cost function based on the L2 norm and calculate what the gradient is, taking a starting point as $\beta_1 = 2$, $\beta_2 = -1$ and $\beta_3 = 0$. For a step length of 0.001, what are the values for β_1, β_2 and β_3 after the first step?

4. Derive the matrix equation which should be solved in order to fit the function $f(x) = \beta_1 + \beta_2 x + \beta_3 x^2 + \beta_4 x^3$.

[2] I won't go into the reasons why here, but you can look this up online if you wish.

3.7.1 Solutions

1.
$$l_\infty = \left(\sum_{i=1}^{m} |x_i|^p \right)^{\frac{1}{p}} \quad \text{as p} \to \infty.$$

As p increases, the largest x_i term (which we will call x_{max}) will dominate more and more. At the limit, all other terms will be insignificant by comparison and can be neglected, i.e.,:

$$l_\infty = (|x_{max}|^p)^{\frac{1}{p}} \quad \text{as p} \to \infty.$$
$$l_\infty = |x_{max}|$$

2. Say $y = x^4 - 4$
Then $\frac{dy}{dx} = 4x^3, \quad = 108$ when $x = 3$
With a finite step of 0.01, take two values: $x_1 = 3.005$ and $x_2 = 2.995$.
Then $y_1 = 77.5413515006$, $y_2 = 76.4613485006$

$$\frac{dx}{dy} \approx \frac{y_1 - y_2}{x_1 - x_2}$$
$$\frac{dx}{dy} \approx \frac{1.080003}{0.01} = 108.0003$$

Now say $y = x^2 - 4$
Then $\frac{dy}{dx} = 2x, \quad = 6$ when $x = 3$
As before, take two values: $x_1 = 3.005$ and $x_2 = 2.995$.
Then $y_1 = 5.030025$, $y_2 = 4.970025$

$$\frac{dx}{dy} \approx \frac{y_1 - y_2}{x_1 - x_2}$$
$$\frac{dx}{dy} \approx \frac{0.060050}{0.01} = 6$$

i.e. we get an exact result - why? Remember, we are doing a Taylor series expansion:

$$f(x) = f(a) + f'(a)(x - a) + \frac{f''(a)(x - a)^2}{2} + ...$$

Note that $f'''(a)$ and higher order terms drop to zero for this particular function, so actually this is exact for our specific case:

$$f(b) = f(a) + f'(a)(b - a) + \frac{f''(a)(b - a)^2}{2}.$$

Letting $b = x - \frac{\delta x}{2}$and $a = x$:

$$f\left(x - \frac{\delta x}{2}\right) = f(x) + f'(x)\left(-\frac{\delta x}{2}\right) + \frac{f''(x)\left(\frac{\delta x}{2}\right)^2}{2},$$

then repeating this equation but swapping the sign for $\frac{\delta x}{2}$:

$$f\left(x + \frac{\delta x}{2}\right) = f(x) + f'(x)\left(\frac{\delta x}{2}\right) + \frac{f''(x)\left(\frac{\delta x}{2}\right)^2}{2}.$$

Take the difference:

$$f\left(x + \frac{\delta x}{2}\right) - f\left(x - \frac{\delta x}{2}\right) = 2f'(x)\left(\frac{\delta x}{2}\right)$$

where several terms have cancelled out. Rearranging this, it's fairly clear to see:

$$f'(x) = \frac{\left[f\left(x + \frac{\delta x}{2}\right) - f\left(x - \frac{\delta x}{2}\right)\right]}{\delta x}.$$

In this case the approximate solution becomes exact. Important reminder that this only works here because we only have terms up to x^2 in $f(x)$.

3. (Q11 2020)
   a) $x_1^2, x_2^2, x_1 x_2$
   b)

$$c = \sum \left(y_i - (\beta_1 + \beta_2 \cdot x_{1i} + \beta_3 \cdot x_{2i})\right)^2 \tag{3.41}$$

$$r_i = y_i - (\beta_1 + \beta_2 \cdot x_{1i} + \beta_3 \cdot x_{2i}) \tag{3.42}$$

$$\frac{\partial c}{\partial \beta_1} = \sum 2 \cdot r_i \cdot (-1) = -2 \sum r_i \tag{3.43}$$

$$\frac{\partial c}{\partial \beta_2} = \sum 2 \cdot r_i \cdot (-x1i) = -2 \sum r_i \cdot x_{1i} \tag{3.44}$$

$$\frac{\partial c}{\partial \beta_3} = -2 \sum r_i \cdot x_{2i} \tag{3.45}$$

x_1	x_2	y	y,	r_i
5.3	8.3	6.2	-3.3	9.5
2.1	2.8	5.6	-0.1	5.7
3.9	5.5	7.2	-1.9	9.1
1.8	2.3	3.4	0.2	3.2

$$\frac{\partial c}{\partial \beta_1} = -2 \cdot 27.5 = -55 \quad \Rightarrow \quad \beta_1 : 2.055 \tag{3.46}$$

$$\frac{\partial c}{\partial \beta_2} = -207.14 \quad \Rightarrow \quad \beta_2 : -0.792 \tag{3.47}$$

$$\frac{\partial c}{\partial \beta_3} = -304.44 \quad \Rightarrow \quad \beta_3 : 0.304 \tag{3.48}$$

4.

$$\begin{pmatrix} \sum x_i^3 & \sum x_i^4 & \sum x_i^5 & \sum x_i^6 \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 & \sum x_i^5 \\ \sum x_i & \sum x_i^2 & \sum x_i^3 & \sum x_i^4 \\ m & \sum x_i & \sum x_i^2 & \sum x_i^3 \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{pmatrix} = \begin{pmatrix} \sum x_i^3 y_i \\ \sum x_i^2 y_i \\ \sum x_i y_i \\ \sum y_i \end{pmatrix}$$

Chapter 4

Linear discriminant functions

C4A

In Sect. 1.4.4, we observed a training dataset and attempted to find a suitable line to separate the two classes. This was firstly done in a 1D sense, then we saw that it may be easier in 2D, combining two parameters together. This generalises into arbitrary numbers of dimensions for multi-parameter problems. Linear discriminant functions are a way of describing these separation functions in arbitrary space; they are linear in the parameters $\boldsymbol{x}$ or a given set of functions of $\boldsymbol{x}$. These form the basis of neural networks which we will look at later in Chapter 6, as well as outlining the concepts for support vector machines studied in Chapter 5.

C4B

Linear discriminant functions are described mathematically as

$$g(\boldsymbol{x}) = \boldsymbol{w}^t\boldsymbol{x} + w_0 \tag{4.1}$$

where $\boldsymbol{w}$ is the WEIGHT VECTOR and w_0 the BIAS or THRESHOLD WEIGHT. The first term ($\boldsymbol{w}^t\boldsymbol{x}$) is effectively a dot product between the weight vector and the feature vector $\boldsymbol{x}$ – each of the individual values in $\boldsymbol{w}$, w_i, are multiplied by the corresponding x_i term, and summed together, i.e. $\boldsymbol{w}^t\boldsymbol{x} = \sum_{i=1}^{m} w_i x_i$.

Figure 4.1(a) illustrates a linear discriminant function for a 2D case, with $\boldsymbol{w} = (2, -1)^t$ and $w_0 = -1$. This can be used for classification purposes by using the point $g(\boldsymbol{x}) = 0$ as the discriminant boundary, i.e. decide ω_1 if $g(\boldsymbol{x}) > 0$ and ω_2 if $g(\boldsymbol{x}) < 0$ (we will not worry about the case where $g(\boldsymbol{x}) = 0$, where the class is undefined) – Fig. 4.1(b) illustrates this. Figure 4.1(c) then presents the 2D plot with the two resulting regions marked.

4.1 Linear discriminant function properties

C4C

We take our linear discriminant function as defined above (decision surface at $g(\boldsymbol{x}) = 0$) and we consider two points ($\boldsymbol{x}_1$ and $\boldsymbol{x}_2$) on the line as illustrated in 3D and 2D in Fig. 4.2.

Then

$$g(\boldsymbol{x}_1) = g(\boldsymbol{x}_2) = 0 \tag{4.2}$$

so

$$\boldsymbol{w}^t\boldsymbol{x}_1 + w_0 = \boldsymbol{w}^t\boldsymbol{x}_2 + w_0 \tag{4.3}$$

and therefore

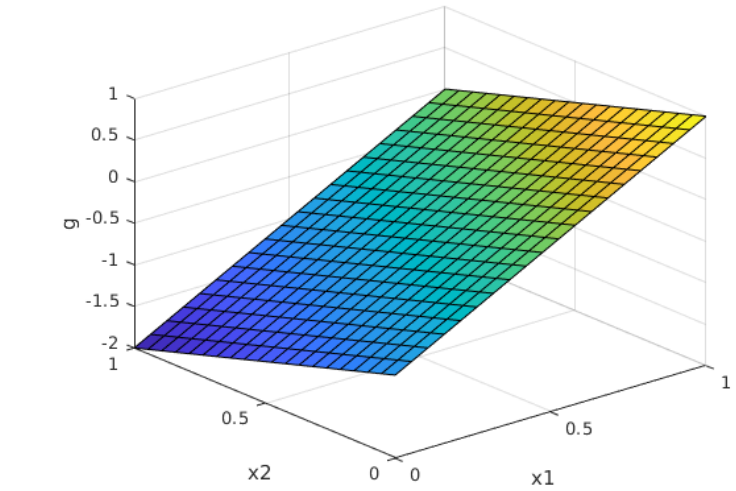
$$\boldsymbol{w}^t(\boldsymbol{x}_1 - \boldsymbol{x}_2) = 0. \tag{4.4}$$

We now have a dot product between two vectors, and the result is zero; ignoring trivial cases where one of the vectors is zero, physically this means that the two vectors $\boldsymbol{w}$ and $(\boldsymbol{x}_1 - \boldsymbol{x}_2)$ must be perpendicular to each other, i.e. the vector $\boldsymbol{w}$ is perpendicular/normal to the decision boundary. Figures 4.3(a) and (b) illustrate this.

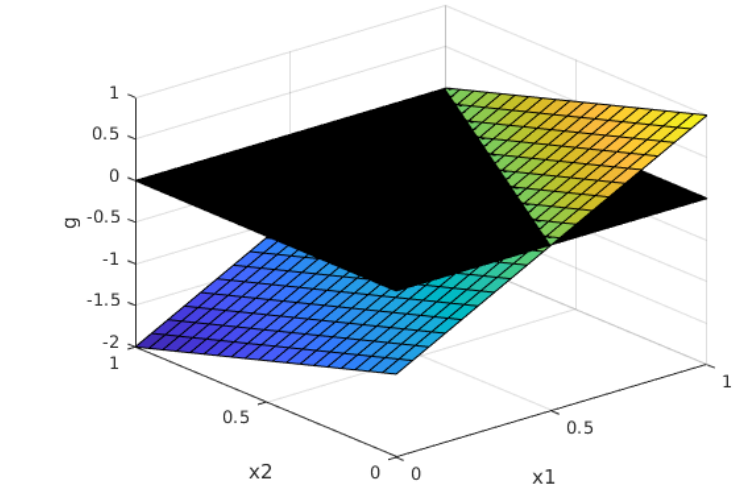
C4D

Now we can consider a point at a distance r from the discriminant surface, illustrated in Fig. 4.3(c). The nearest point on the surface is labelled $\boldsymbol{x}_p$; this can be considered the projection of point $\boldsymbol{x}$ onto the surface. Take the linear discriminant function of point $\boldsymbol{x}$

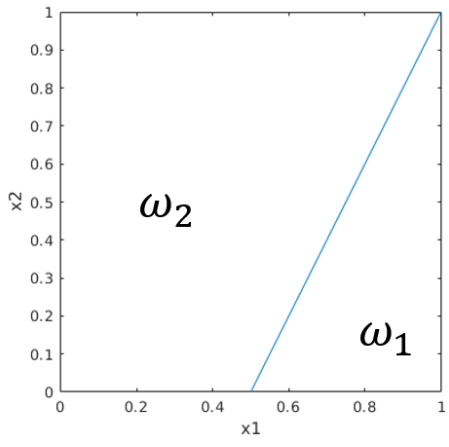
$$g(\boldsymbol{x}) = g(\boldsymbol{x}_p + r\hat{\boldsymbol{w}}) \tag{4.5}$$



(a)



(b)



(c)

Figure 4.1: Linear discriminant function example, where $\boldsymbol{w} = (2, -1)^t$ and $w_0 = -1$. (a) just shows the function, (b) then shows the function with $g(\boldsymbol{x}) = 0$ marked as well, giving a discriminant line. (c) shows the resulting 2D plot and the classification regions.

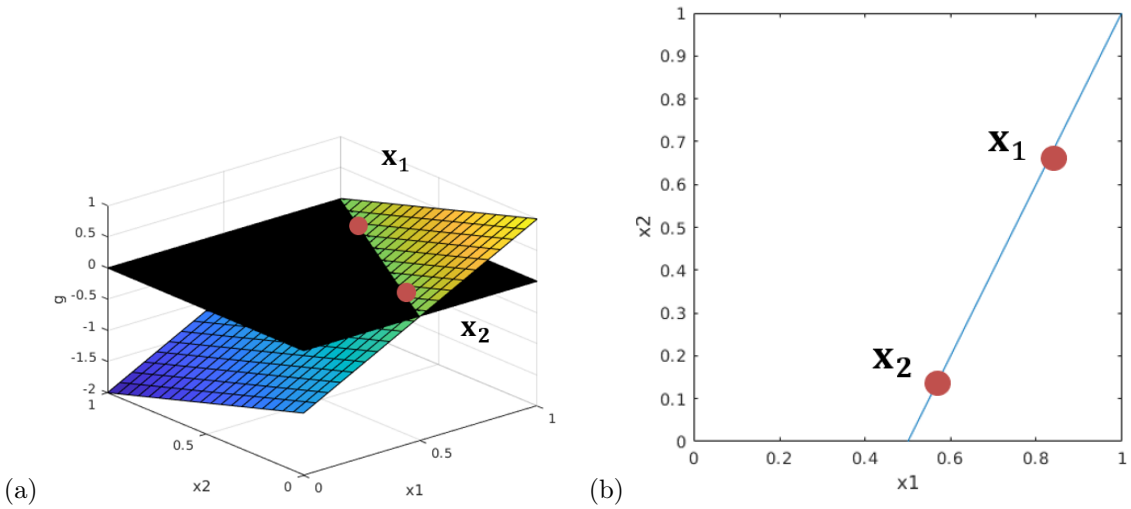


Figure 4.2: Identification of two points on the decision boundary of a linear discriminant function, (a) plotted in 3D and (b) in 2D.

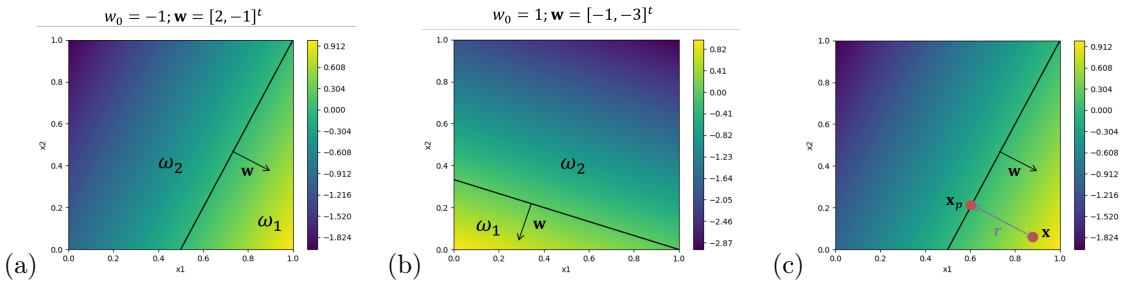


Figure 4.3: (a) and (b) give two illustrations of the weighting vector being perpendicular to the discriminant line, as shown in eq. (4.4). (c) shows definitions for calculating the distance from the line.

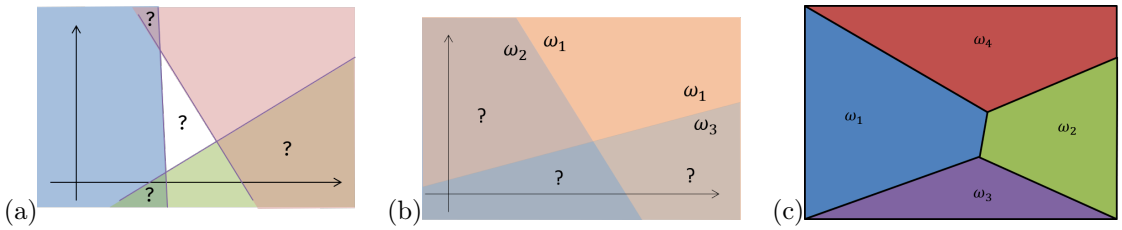


Figure 4.4: Classifying multiple regions. (a) shows the potential of having one discriminant function for each class, which leaves ambiguity in some points defined many times and some others not defined at all. (b) presents the solutions of having a function for each pair of classes, but this then introduces ambiguity with points potentially being classified many times.

where $\hat{\boldsymbol{w}} = \frac{\boldsymbol{w}}{|\boldsymbol{w}|}$ is the unit vector in direction $\boldsymbol{w}$. Expanding out the function g

$$g(\boldsymbol{x}) = \boldsymbol{w}^t \left(\boldsymbol{x}_p + r\frac{\boldsymbol{w}}{|\boldsymbol{w}|} \right) + w_0 \tag{4.6}$$

$$= \boldsymbol{w}^t \boldsymbol{x}_p + w_0 + r\frac{\boldsymbol{w}^t \boldsymbol{w}}{|\boldsymbol{w}|} \tag{4.7}$$

Note that since $\boldsymbol{x}_p$ is on the surface, $g(\boldsymbol{x}_p) = 0 = \boldsymbol{w}^t \boldsymbol{x}_p + w_0$. We can substitute this into (4.7) to eliminate the first two terms and give

$$g(\boldsymbol{x}) = 0 + r\frac{|\boldsymbol{w}|^2}{|\boldsymbol{w}|} \tag{4.8}$$

$$= r\,|\boldsymbol{w}| \tag{4.9}$$

so

$$r = \frac{g(\boldsymbol{x})}{|\boldsymbol{w}|}. \tag{4.10}$$

Described simply, the linear discriminant function divided by the magnitude of the weighting vector gives the distance from the decision surface. If the weighting vector is a unit vector then $g(\boldsymbol{x})$ directly gives that distance. It should be noted that this is a signed distance, i.e. it will be negative on the other side of the discriminant line.

4.2 Multi-category classification

C4E

So far we have considered linear discriminant functions for binary (two-class) decisions. How does this expand to multiple classes ω_i? One option is to define one classifier for each class, identifying which is within each. An example of this is illustrated in two-parameter space in Fig. 4.4(a). The issues from this are clear: there are several points which would fall into multiple classes, and one clear region which would fall into none. In Fig. 4.4(b) we have another solution, where we define a classifier between each pair of classes. This avoids having unclassified regions, but then introduces many points which will fall into multiple classes.

The commonly taken solution is to define one linear discriminant function for each class and classify each point as the one which is highest; this is a linear machine. Mathematically this can be described as

$$g_i(\boldsymbol{x}) = \boldsymbol{w}_i^t \boldsymbol{x} + w_{i0} \tag{4.11}$$

then assign point $\boldsymbol{x}$ to ω_i if $g_i(\boldsymbol{x}) > g_j(\boldsymbol{x})$ for all $j \neq i$.

4.3 Generalised linear discriminant functions

C4F

Consider the case where the straight lines and flat planes considered so far are unsuitable for our purposes. Instead we want to move to non-linear boundaries which would enable curved surfaces

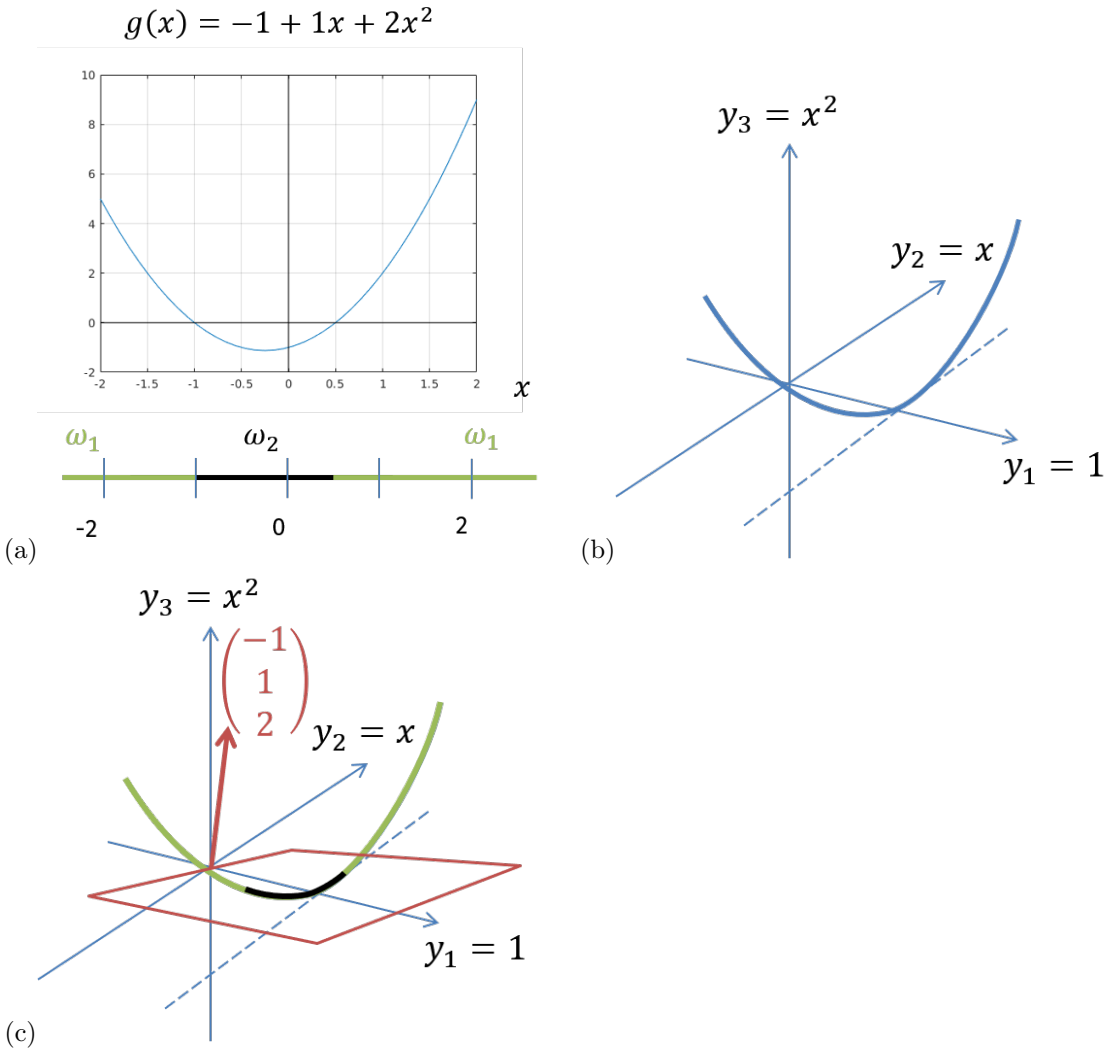


Figure 4.5: Generalised linear discriminant function. (a) shows an example of a 1D polynomial decision function $g(x) = -1 + 1x + 2x^2$. (b) shows the general mapping from the function x to $\boldsymbol{y} = (1, x, x^2)^t$ in 3D space. (c) then shows how this mapping can be used with specific coefficients $\boldsymbol{a} = (-1, 1, 2)^t$, which form the plane marked, which then intersects the curve $\boldsymbol{y}(x)$ and identifies the decision boundary. This matches the same function as shown in (a).

and lines to be able to describe the decision boundary. To achieve this we need higher order (e.g. quadratic) terms. We can therefore introduce the generalised linear discriminant function

$$g(\boldsymbol{x}) = \boldsymbol{a}^t \boldsymbol{y} \tag{4.12}$$

where $\boldsymbol{a}$ is a $\hat{d}$-dimensional weight vector and the $\hat{d}$ functions $y_i(\boldsymbol{x})$ can be arbitrary functions of $\boldsymbol{x}$. By incorporating these arbitrary functions y_i, we can allow for greater complexity.

We can illustrate this with an example. If $g(x) = -1 + 1x + 2x^2$ then $g(x) > 0$ if $x < -1$ or if $x > 0.5$, i.e. we have multiple boundaries which are not possible through linear functions. This is illustrated in Fig. 4.5(a).

To express this in the form of a generalised linear discriminant function, we can utilise the mapping

$$\boldsymbol{y} = \begin{pmatrix} 1 \\ x \\ x^2 \end{pmatrix} \tag{4.13}$$

which represents

$$g(x) = a_1 + a_2 x + a_3 x^2. \tag{4.14}$$

This is a 1D function in x, but $\boldsymbol{y}$ maps it into a parabola in 3D space. Fig. 4.5(b) illustrates this mapping: $y_1 = 1$, $y_2 = x$ and $y_3 = x^2$.

To look at the specific case of $g(x) = -1 + 1x + 2x^2$, take $\boldsymbol{a} = (-1, 1, 2)^t$. This introduces a new plane into the domain which is described by its normal, $(-1, 1, 2)^t$. This is illustrated in Fig. 4.5(c). This plane intersects the general parabola to split the function into two regions, corresponding to the two classes. This matches the original discriminant behaviour of Fig. 4.5(a). C4G

In theory, this could be a very powerful tool; from a single set of functions in $\boldsymbol{y}$ it is possible to train a number of different vectors $\boldsymbol{a}$ which should be able to fit the training data, and these functions can enable significant complexities to exist in the model, potentially matching the underlying behaviour. As we will see in the next chapter, it is this very concept which gives support vector machines their power. However, generalised linear discriminant functions are themselves vary rarely used, because they are not an efficient tool. The curse of dimensionality means that when moving to higher dimensional spaces, with many input parameters (remember that in this case we just had a single input parameter, x, which mapped to three parameters in $\boldsymbol{y}$), we will end up with large numbers of coefficients which need to be established. It is also difficult to determine which function set can be used.

In practice, therefore, standard linear discriminant functions are used more often. However, the convenience of incorporating the bias term is sometimes utilised with $\boldsymbol{y} = (1, x_1, ...x_n)^t$, $\boldsymbol{a} = (w_0, w_1 ... w_n)^t$, although this is mathematically identical to the standard functions, and this convention will be relied upon when we study neural networks in Chapt. 6.

4.4 Training

C4H

As for other problems we've studied we can assume we have an input dataset $\boldsymbol{X}$ consisting of m sets of n parameters. Matching these, we have m corresponding class values, stored in a vector $\boldsymbol{y}$. For mathematical convenience we will set the y values all to be 1 for ω_1 or -1 for ω_2 to indicate the class. We will assume that the bias term has been incorporated into $\boldsymbol{X}$ and the weighting vector $\boldsymbol{w}$ here. Based on this, we can produce the equation

$$\boldsymbol{X}\boldsymbol{w} = \boldsymbol{y} \tag{4.15}$$

where we would like to solve to find the weighting values $\boldsymbol{w}$ which best fit this equation. To do this we can utilise the l_2 error metric

$$E_2 = \sum \left(\boldsymbol{w}^t \boldsymbol{x}_i - y_i\right)^2 \tag{4.16}$$

and minimise this via a gradient descent approach, where the gradient is

$$\boldsymbol{\nabla} E_2 = \sum 2\left(\boldsymbol{w}^t \boldsymbol{x}_i - y_i\right) \boldsymbol{x}_i. \tag{4.17}$$

This approach was described extensively for regression in Chapter 3.4.

4.5 Conclusions

Linear discriminant functions provide an important basis for many machine learning techniques including neural networks and support vector machines. They can be used for many classification problems where a simple linear boundary is needed. They can be extended to more complex boundaries by incorporating higher order nonlinear functions, however, this is typically complex to achieve reliably and not efficient – the next chapter on support vector machines will provide a more efficient framework to achieve this. Linear discriminant machines can be produced to perform multi-category classification by taking the largest linear discriminant function at each point.

Chapter 5

Support vector machines

C5A

We begin this chapter by considering what it is that we are trying to achieve when we train a classifier. The last chapter looked at linear discriminant functions, which could separate the parameter space into two domains. One question arises in how to select the best possible line; Figs. 5.1(a) and (b) illustrate two potential options for an example training dataset. The goal is to achieve good generalisation performance, so that as many subsequent points are classified correctly as possible. In this case, the test point marked by the square is classified in different classes depending on the two lines. By observation, we can say that the square is slightly closer to the red circles on the lower right, so probably Fig. 5.1(a) is a better line to have. But it is important to consider how to actually express this mathematically.

C5B

Support vector machines try to address this by considering the training points at the extremes of their classes, and placing the decision boundary as far from these training points as possible. Figures 5.2(a), (b) and (c) to illustrate this concept. In Fig. 5.2(a), we can see that three such points have been identified and labelled as support vectors. A boundary is then placed between these such that the margin – the shortest distance between the support vectors and the boundary – is maximised. It is possible to select several different sets of support vectors which give different margins, and alternatives are shown in Figs. 5.2(b) and (c). The boundary where the margin is maximised is called the Maximal Margin Hyperplane (MMH). When used as a decision boundary to classify points, it becomes the Maximal Margin Classifier (MMC).

Returning to our previous question, where we were trying to decide where to place the boundary in Fig. 5.1, it was suggested that the solution from Fig. 5.1(a) was superior because the square point was classified according to the set it was closest to. The MMC tries to capture this behaviour to give a good generalisation performance, by placing the boundary at the furthest possible distance from the nearest points in both datasets.

These nearest points, the support vectors, are critical in defining the boundary position. All other values are ignored (other than exploiting the knowledge that these are further from the boundary). One useful intuitive way of thinking about these points is that, being closest to the boundary, they are the most difficult to classify, so they are therefore the most informative for classification (we will discuss the consequences for overfitting later on).

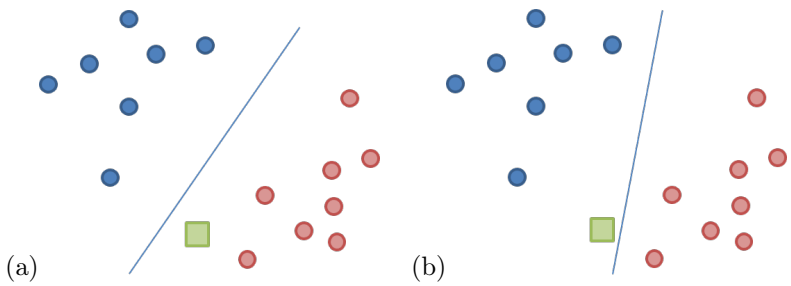


Figure 5.1: Examples of choosing a classification line, and the consequence for subsequently classifying the square marked.

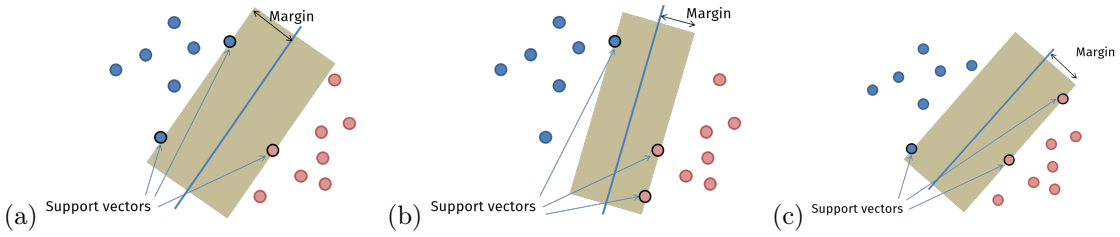


Figure 5.2: Selecting support vectors to produce different margins.

5.1 What do we mean by "vector"?

C5C

The use of the term "vector" is often confusing for students, thinking about physical vectors which could symbolise direction, or the travel from one position to another. In this case, vector is simply used to refer to a training sample, which can be described as $\boldsymbol{x} = (x_1, x_2, x_3 ... x_n)$ (as discussed in Sect. 1.4.6). A vector is just a set of numbers arranged in a line (in the same way a matrix is a set of numbers arranged in a grid). We often represent these as physical points in a 2D (or sometimes 3D) plot for illustration, but this does not necessarily mean that the underlying data corresponds to physical points in space.

5.2 Support vector machines

C5D

We will utilise concepts from the linear discriminant functions described in Chapter 4. We will start by forcing the weighting vector in this function to have unit length[1], i.e.

$$|\boldsymbol{w}| = 1 \tag{5.1}$$

so

$$|\boldsymbol{w}|^2 = \boldsymbol{w}^t \boldsymbol{w} = 1. \tag{5.2}$$

We then wish to maximise the margin value M such that, for all m training points $\boldsymbol{x}_i$ with corresponding classifications $y_i = \{1, -1\}$:

$$y_i(\boldsymbol{w}^t \boldsymbol{x}_i + w_0) \geq M. \tag{5.3}$$

This can be illustrated as in Fig. 5.3. Figure 5.3(a) shows the arrangement for a simple set of data in two classes in 2D space, and (b) shows how this is projected into 1D space by multiplying by the weighting vector $\boldsymbol{w}$. Figure 5.3(c) then multiplies by y_i to illustrate how eq. (5.3) must hold.

The algorithms behind training such a classifier are not readily accessible and beyond the scope of this course, so will not be outlined here. Of course, you may look these up online if you wish. However, eq. (5.3) captures the key principles of the method, which is important for understanding how support vector machines work.

The key advantage of support vector machines is that they are generally efficient, since only the support vectors need to be stored to define the classifier. A disadvantage of them is that they are highly sensitive to the support vectors, and therefore any outliers in the training data could greatly skew the classifier. Figure 5.4 provides a visual example of this sensitivity. Ignoring outliers, the thick solid line shows where the classification boundary would be, which seems to match the general behaviour well. In contrast, the thin line marks the position of the boundary when incorporating the outlier on the bottom left. This is much less representative of the true behaviour that we would expect. We would like to have a mechanism to enable us to deal with outliers when establishing the boundary position and avoid overfitting.

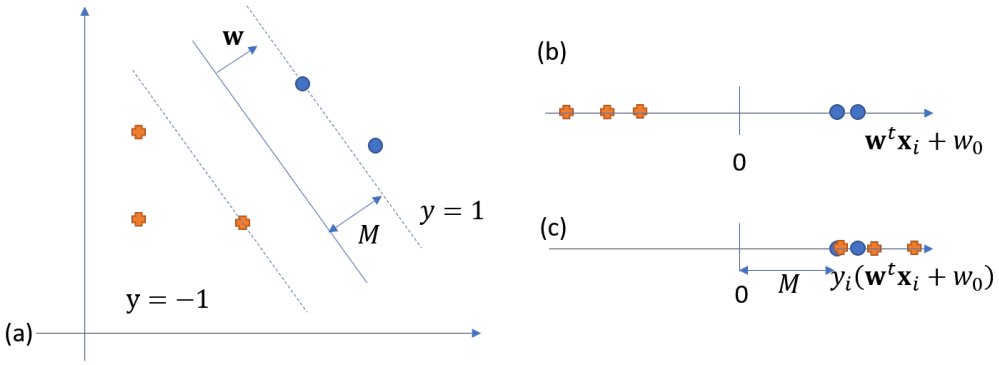


Figure 5.3: Construction of the support vector classifier. (a) shows the decision boundary and margins for a small set of training data in 2D space. (b) shows this projected onto a 1D line via the linear discriminant function. (c) then illustrates the definition of eq. (5.3) by multiplying by y_i.

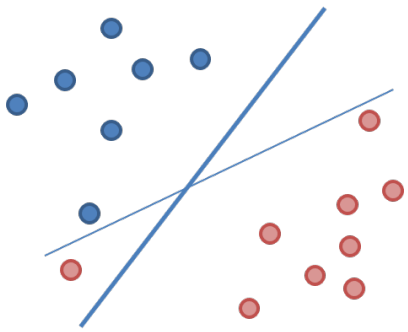


Figure 5.4: Illustrating the potential effect of outliers for support vector machines. The thick line marks the most likely boundary based on the data ignoring the bottom left outlier, but the thin line shows where this would be placed without making any allowances for outliers.

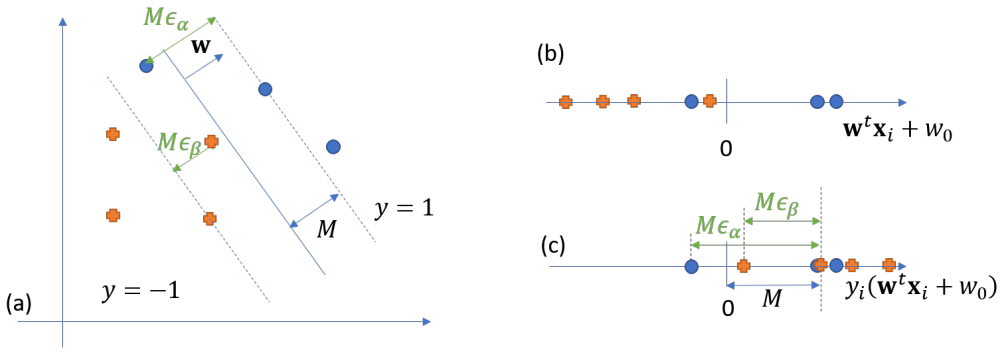


Figure 5.5: Illustration showing how soft margins work with support vector machines. (a) shows the 2D parametric space, with two points within the margin region. (b) presents this projected into 1D. (c) then multiplies this by the class indicator, and highlights how the two points lie to the left of the margin location, i.e. $y_i(\boldsymbol{w}^t\boldsymbol{x}_i + w_0) < M$, by the distances $M\epsilon_\alpha$ and $M\epsilon_\alpha$. This is permitted by (5.4).

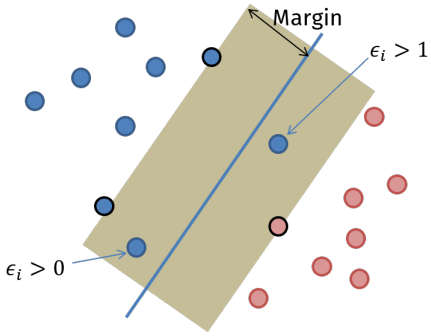


Figure 5.6: Definition of epsilon for particular points in a soft margin classifier.

5.3 Soft margin classifiers

C5E

We can reduce our sensitivity to individual support vectors by relaxing the requirement that all points are outside the margin region. To do this we can introduce a set of values ϵ_i and incorporate these as

$$y_i(\boldsymbol{w}^t\boldsymbol{x}_i + w_0) \geq M(1 - \epsilon_i). \tag{5.4}$$

These values are restricted to be non-negative, i.e. $\epsilon_i \geq 0$, and all the values are required to sum to less than the budget C

$$\sum_{i=1}^{n} \epsilon_i \leq C. \tag{5.5}$$

Figure 5.5 presents a rendering of the soft margin classifier. In Fig. 5.5(a), the margin region is encroached upon by two points, one from each class, with one even exceeding the classification line itself. The soft margin classifier permits this. Figure 5.5(b) illustrates this as a function of the 1D parameter $\boldsymbol{w}^t\boldsymbol{x}_i + w_0$, then Fig. 5.5(c) projects the points according to their classified values y_i, i.e. $y_i(\boldsymbol{w}^t\boldsymbol{x}_i + w_0)$ on the x axis. This last plot shows how the two points highlighted exceed the margin, by $M\epsilon_\alpha$ and $M\epsilon_\beta$.

C5F

The values for ϵ_i correspond to different cases, as illustrated in Fig. 5.6:

- $\epsilon = 0$ means that the training observation $\boldsymbol{x}_i$ is on the correct side of the margin
- $\epsilon > 0$ means that the observation $\boldsymbol{x}_i$ is on the wrong side of the margin

[1] Although note that while we do this in the derivation, it does not necessarily mean that we have to do this in all cases.

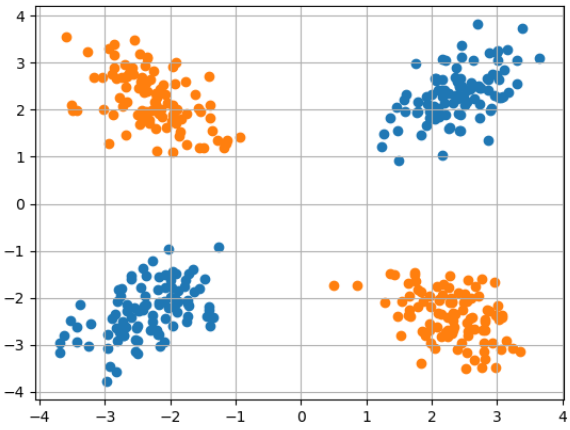
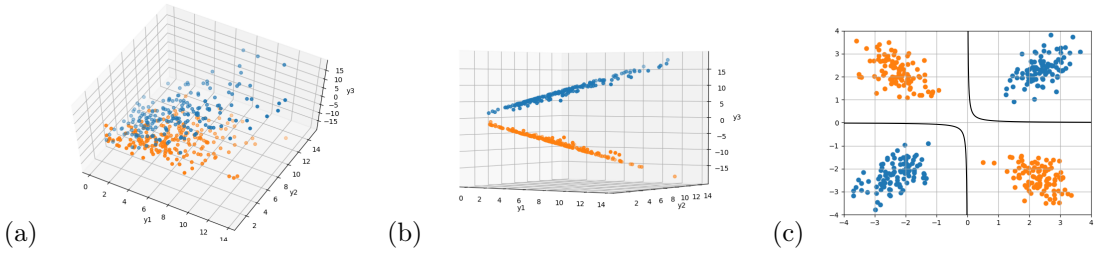


Figure 5.7: A challenging classification problem; it is not possible to perform a linear separation of this data. This data comes from the XOR operator.



(a) (b) (c)

Figure 5.8: (a) The dataset from Fig. 5.7 projected into a higher dimensional space as given in eq. (5.6) and in a different view in (b). (c) shows the final decision boundary back in the original space.

- $\epsilon > 1$ means that the observation $\boldsymbol{x}_i$ is on the wrong side of both the margin and the classification hyperplane

C5G

The budget C defines the maximum permissible value of the sum of the ϵ_i values, i.e. the extent to which we permit points to be on the wrong side of the margin. Setting $C = 0$ will produce standard hard boundary initially studied, then increasing C will produce weaker fitting but less overfitting[2]. The use of C is a form of regularisation, which was first discussed in Sect. 3.5 for regression. We are reducing the tightness of fit to the training data such that overfitting should be reduced. This would increase bias but decrease variance, as discussed in Sect. 11.1. It is important to perform validation to ensure a suitable value of C has been selected; Sect. 11.2 discusses this.

5.4 Nonlinear mapping

C5H

In some cases, it is not possible to perform a linear separation of the data; Fig. 5.7 presents an example of a particularly tricky dataset, based on XOR behaviour. We need to use higher order function to be able to separate the data. The concept is to project the data into a higher order space in order to find a suitable separation hyperplane. This matches the proposed approach of generalised linear discriminant functions of Sect. 4.3, although support vector machines have some significant advantages which can avoid many of the drawbacks.

[2]It is noted that the SVM implementation used in the tutorials in scikit-learn has the inverse definition for this parameter – small values mean a very soft boundary and high values mean a hard boundary. You are encouraged to look this behaviour up in the documentation if it is important for your application.

We can define a new space as

$$\boldsymbol{y} = (x_1^2, x_2^2, \sqrt{2}x_1x_2) \tag{5.6}$$

and a 3D rendering of this is given in Fig. 5.8(a). Initially this does not seem to provide much benefit, but when we rotate the plot as in Fig. 5.8(b) we can see that the y_3 parameter is able to distinguish the two classes quite clearly. We can see that $y_3 = 0$ should be able to do it, but we will take $y_3 = \sqrt{2}x_1x_2 = 0.1$ to avoid a singularity. From this, C5I

$$x_1 = \frac{0.1}{\sqrt{2}}\frac{1}{x_2} \tag{5.7}$$

which is then rendered in Fig. 5.8(c)[3]. C5J

The projection in the previous case was conveniently selected to match the problem. It is not always clear which space is best to project into. However, libraries (including scikit-learn, which we will use in the tutorials) typically have toolboxes of general projection spaces which can be utilised for different problems. It should be noted that data is more likely to be separable when projected into more dimensions, so this is often sensible to do, although note that this is increasing complexity of the model so can lead to overfitting.

5.5 Kernels

In Sect. 4.3, we highlighted that increasing the parameter space to include higher order functions can lead to a significant increase in the complexity of the problem, given the resulting high dimensionality when considering the potential combinations of all the different parameters. Support vector machines will suffer from this, however, kernels are a powerful way of efficiently managing this. C5K

At this point we will take a slight detour into dot products. It is possible to reformulate the entire training process in terms of dot (or inner) products between the different training points (we won't!). This means that there will be $m(m-1)/2$ distinct values for the different combinations of dot products (although in practice, we will typically only need a small fraction of these corresponding to the support vectors). Although this is an increase in the number of values, note that the dot products are scalar values so this will compensate to an extent, particularly for high-parameter problems (or those projected into high-dimensional spaces). C5L

One approach to training support vector machines is as follows

1. Obtain training data in original space
2. Project into higher dimensional, higher order space
3. Calculate dot products there
4. Generate resulting support vector machine classifier

This approach is useful to help understanding, but is not particularly efficient.

Kernels are functions which enable dot products in higher order space to be calculated quickly from the base parameters. This means that there is no need to calculate the values in the higher order space at all, and meaning that point 2 above can be completely skipped. This approach enables support vector machines to capture complex behaviour, without losing efficiency. C5M

There are a range of kernels available in most libraries. The polynomial kernel is common, and given by

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i.\boldsymbol{x}_j + c)^d = \left(\sum_{k=1}^{n} x_{ik}x_{jk} + c\right)^d \tag{5.8}$$

where c and d (representing the polynomial degree) are user-selectable tuning parameters. In the prior example we effectively used $c = 0$ and $d = 2$. The standard linear classifier has $c = 0$ and $d = 1$ to give the standard dot-product.

[3] If we had set $y_3 = 0$ the two lines would meet in a sharp corner at (0,0), but dealing with the maths would have been tricky.

The radial kernel (RBF for Radial Basis Function) is given by

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left[-\gamma \sum_{k=1}^{n} (x_{ik} - x_{jk})^2\right]^d, \gamma > 0. \tag{5.9}$$

This is localised, i.e. the underlying functions disappear to zero away from the given point. This can be beneficial over polynomial functions, which do not have this behaviour; you can recognise that all of the x^n terms in the polynomial form will be non-zero (and potentially large) across all values in the range x. This means that RBFs can be more suitable for capturing local complexities, although clearly they are more susceptible to overfitting and hence variance increasing, while polynomials will have more global behaviour.

You should also note that there are typically a range of other kernels available in most packages, including Scikit-learn.

5.6 Conclusions

To apply support vector machines in practice, it is necessary to select a suitable kernel, as well as the degree of that kernel. Then a suitable soft-margin parameter needs to be selected to balance bias and variance. A library should then be able to fit a classifier to the training data based on this. This will identify the support vectors, which will classify subsequent datasets. As usual, we should always validate!

From this chapter, you should understand how maximum margin hyperplanes are defined and from that maximum margin classifiers. You should also understand how this margin can be relaxed for certain points in the training set, making a soft margin support vector classifier, which should reduce overfitting and variance. You should recognise that projecting into higher order space can make it easier to subdivide data, and that kernels can calculate the support vector machine much more efficiently without having to actually project the data.

5.7 Questions

1. A linear support vector classifier has points $(2, 1)$, $(0, -1)$ as the support vectors classified as 0 and 1 respectively.

   (a) Expressing the resulting decision function in the form $g(\boldsymbol{x}) = \boldsymbol{w}^t\boldsymbol{x} + w_0$ with $g(\boldsymbol{x}) > 0$ being class 1 and $g(\boldsymbol{x}) < 0$ corresponding to class 0, what are $\boldsymbol{w}$ and w_0?

   (b) Now assume it is a soft-margin classifier. Taking a training point at $(0.5, 1)$ classified as 1, what is the value of ϵ_i for this point? If it was now classified as 0, what would the value of ϵ_i be?

5.7.1 Solutions

1. a)

$$\boldsymbol{w} = \begin{pmatrix} 0 \\ -1 \end{pmatrix} - \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} -2 \\ -2 \end{pmatrix}$$

$g(\boldsymbol{x}) = 0 \quad$ when $\boldsymbol{x} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad$ so: $0 = \begin{pmatrix} -2 & -2 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} + w_0 \quad \Rightarrow \quad w_0 = 2$

b)

$g \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} -2 & -2 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \end{pmatrix} + 2 = -6 + 2 = -4 \quad$ so margins range from -4 to 4

$g \begin{pmatrix} 0.5 \\ 1 \end{pmatrix} = \begin{pmatrix} -2 & -2 \end{pmatrix} \begin{pmatrix} 0.5 \\ 1 \end{pmatrix} + 2 = -3 + 2 = -1$

$\epsilon_i = 1.25$ if classified as 1, $\epsilon_i = 0.75$ if classified as 0

Chapter 6

Neural networks

C6A

So far on this course we have discussed a range of models. These generally have a similar approach; they assume the behaviour of the data follows some function, e.g. a particular probability distribution, a polynomial regression, or a particular set of basis functions with support vector machine. The parameters defining these forms are then learnt from the training data available. This poses a challenge to the designer of such a system; for arbitrarily complex data, how should we select suitable functions to describe the behaviour?

Neural networks are often described as having the ability to fit both the functional forms as well as the coefficients describing how they should be combined. They have the power, within a concise set of coefficients, to capture complex non-linear behaviour. Much of the modern power of machine learning has come from the development of neural networks, particularly in the form of deep networks (containing hundreds or even thousands of layers).

6.1 Neural network layout

C6B

In Chapter 4.3 we studied linear discriminant functions, which can be mathematically expressed as

$$g(\boldsymbol{x}) = \sum_{i=1}^{n} w_i x_i + w_0. \tag{6.1}$$

This can be represented graphically as shown in Fig. 6.1. The inputs are on the left - x_1 and x_2 and are combined with their respective weightings marked, w_1 and w_2. The bias weighting w_0 is represented as being multiplied by 1.

Previously we used the output from the linear discriminant function to classify values; we can represent classifying values as -1 or 1 by applying the sign function to the weighted sum

$$f(g(\boldsymbol{x})) = \frac{g(\boldsymbol{x})}{|g(\boldsymbol{x})|} \tag{6.2}$$

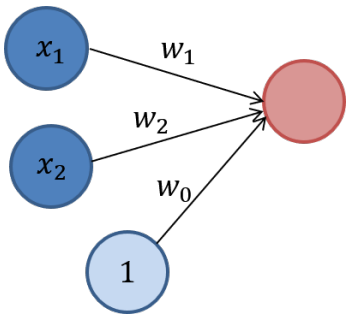


Figure 6.1: Graphical representation of a linear discriminant function. Values x_1 and x_2 are inputs and w_0, w_1 and w_2 are weightings.

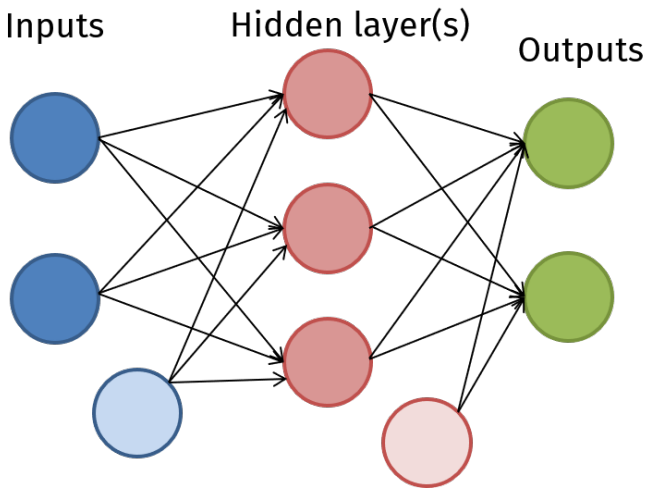


Figure 6.2: Neural network layout, with three layers. The bias nodes are marked in a lighter colour.

i.e. dividing the function by its absolute. Considering both positive and negative values, satisfy yourself that this does produce the required output of -1 when $g < 0$ and $+1$ when $g > 0$. An important aspect is that this function introduces nonlinearity into the neural network, which is key to it working.

We will quickly review what nonlinear functions mean. A mathematical description of a linear function can be given as

$$L(a) + L(b) = L(a + b). \tag{6.3}$$

This means that if you double your input to function L, then $L(2a) = L(a + a) = L(a) + L(a) = 2L(a)$. Within stress analysis, the stress-strain relationship is assumed linear (in many cases), which is a valid assumption for many materials we use provided the strain is small. If you double the stress, the strain will double. In the stress-strain example, linearity is only valid for small strains; we know that the stress-strain curve is only a straight line for a certain length, and after a certain point plastic deformation occurs. It is common in engineering that we can approximate behaviour as linear within a limited range (and this is in fact the first term of the Taylor series approximation).

Nonlinearity is everything else. Nonlinearity within an amplifier will distort the higher amplitudes of a signal making the sound change,[1] and there are many other examples of nonlinearity throughout engineering.

The linear discriminant functions discussed above are linear combinations of the input parameters; each weighting term w_i simply scales each parameter. This limits the functional forms which can be captured by this tool (although their simplicity is highly valuable, computationally speaking). The sign function of eq. (6.2) is a nonlinear function (satisfy yourself of this based on the description above). This introduction of nonlinearity into the output provides power to describe increasingly complex behaviour. Neural networks combine many different linear discriminant functions with nonlinearity introduced across multiple layers, ultimately reaching a set of outputs. C6C

Figure 6.2 illustrates a neural network. A neural network consists of many linear discriminant functions combined together, with the output from each fed through a nonlinear function, then on to other linear discriminant functions. In this figure, the two bias terms are marked in a lighter colour, although note that I will not always explicitly draw out the bias terms each time from this point forwards. This network has three layers, consisting of an input layer, hidden layer and output layer, as marked. The arrows mark links between nodes in the network; each of these has a weight associated with it which can be modified to train the network.

To demonstrate the use of a neural network, we will step through with the example from

[1] This may be desired if you are a guitarist in heavy metal band "Killer Rage Anger Scream" but less so if you're listening to Bach's Air on the G String. A lot of things are about perspective.

Fig. 6.2. Inputs are provided to the two input values, labelled as x_i. These are then multiplied by weighting terms along each linking arrow, and summed to provide NET ACTIVATION VALUES at each of the hidden layer nodes, described as

$$net_j = \sum_{i=0}^{n} w^1_{ji} x_i. \tag{6.4}$$

It should be noted that we now have additional numbers describing the positions of the weighting terms; superscript indicates the layer number (do not get confused with exponents) and the two subscript terms describe the start and end nodes of the link. Often these weightings are expressed as a matrix for each layer in the network. The hidden layer nodes have nonlinear functions associated with them, giving

$$y_j = f(net_j). \tag{6.5}$$

This process is repeated to give the output values, described as z_k. Mathematically, we can describe this whole process as C6D

$$z_k = f\left[\sum_{j=0}^{nh} w^2_{kj} f\left(\sum_{i=0}^{n} w^1_{ji} x_i\right)\right]. \tag{6.6}$$

It should be noted that we have described a fairly simple example here; in practice, more layers can be used, and more nodes in each layer. The nonlinear functions themselves can be varied at each layer.

In this approach, we are summing many functions together, which has similarities with Taylor series and Fourier series. However, we rarely get anywhere close to defining a "full" set of parameters in the way we would do with these approaches; we have fewer weighting terms available. The neural network provides a very concise expression of behaviour, such that even with a limited number of weighting terms we can get a huge range of functional outputs. We will explore this now.

6.1.1 Combining multiple functions

C6E

We will introduce another activation function here, the rectified linear function, described as

$$f(net_j) = \begin{cases} net_j & net_j > 0 \\ 0 & otherwise. \end{cases} \tag{6.7}$$

This is abbreviated to ReLU for Rectified Linear Unit. "Unit" is synonymous with node, which reflects that the activation function is the defining feature when selecting a node. There are many other functions which are referred to as units in this way.

This is illustrated in Fig. 6.3(a). Figure 6.3(b) then illustrates how these functions can be applied in a simple 3-layer neural network. The net activation term for the top node of the hidden layer is given by

$$net_1 = w^1_{10} + w^1_{11} x \tag{6.8}$$

where x is the single input to the network and w^1_{10} and w^1_{11} represent the weighting terms for the bias and the input respectively, leading to that top node. The ReLU function can then be applied to that node as

$$y = f(net_1) = f(w^1_{10} + w^1_{11} x).$$

One can calculate the zero intercept of $net_1 = 0$ as

$$w^1_{10} + w^1_{11} x = 0$$

so

$$x = -\frac{w^1_{10}}{w^1_{11}}.$$

When the ReLU function is applied to this net activation term, the zero point defines where the gradient discontinuity in the function is. The gradient in the non-zero section of the function is

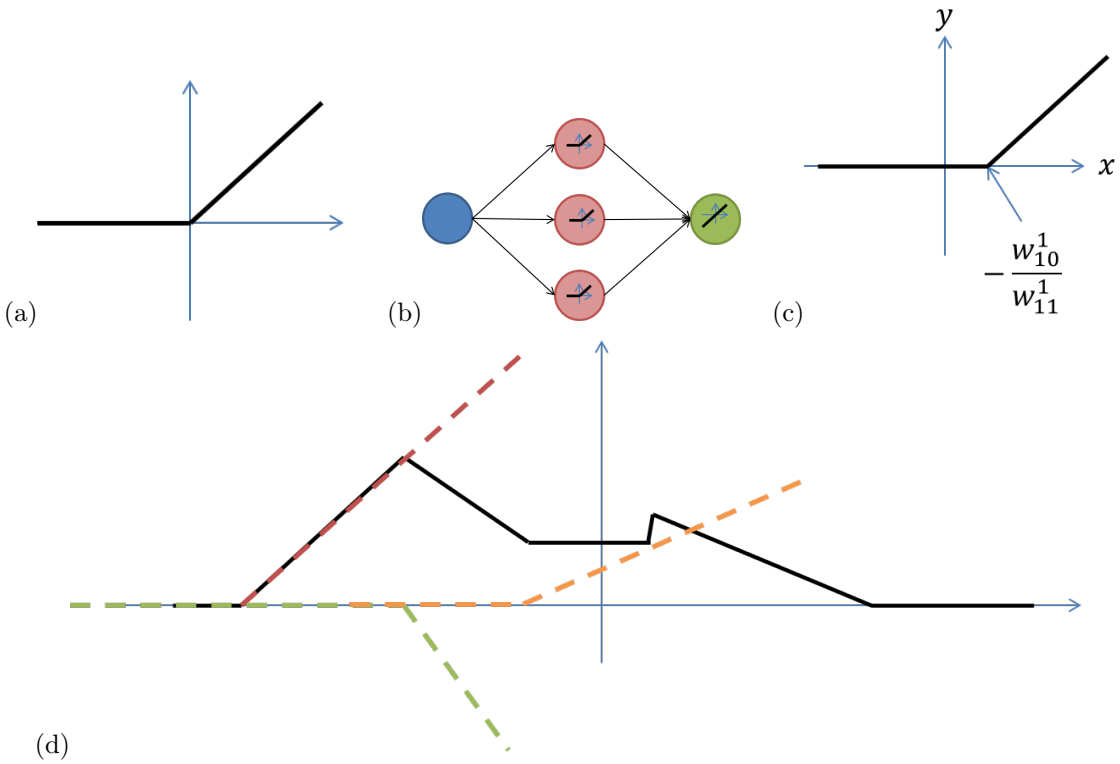


Figure 6.3: (a) shows the ReLU (recified linear) activation function, and (b) shows an example neural network incorporating these. (c) illustrates how the ReLU activation function will behave depending on the input. (d) shows how multiple ReLU functions with different weightings can be combined to generate an arbitrary function.

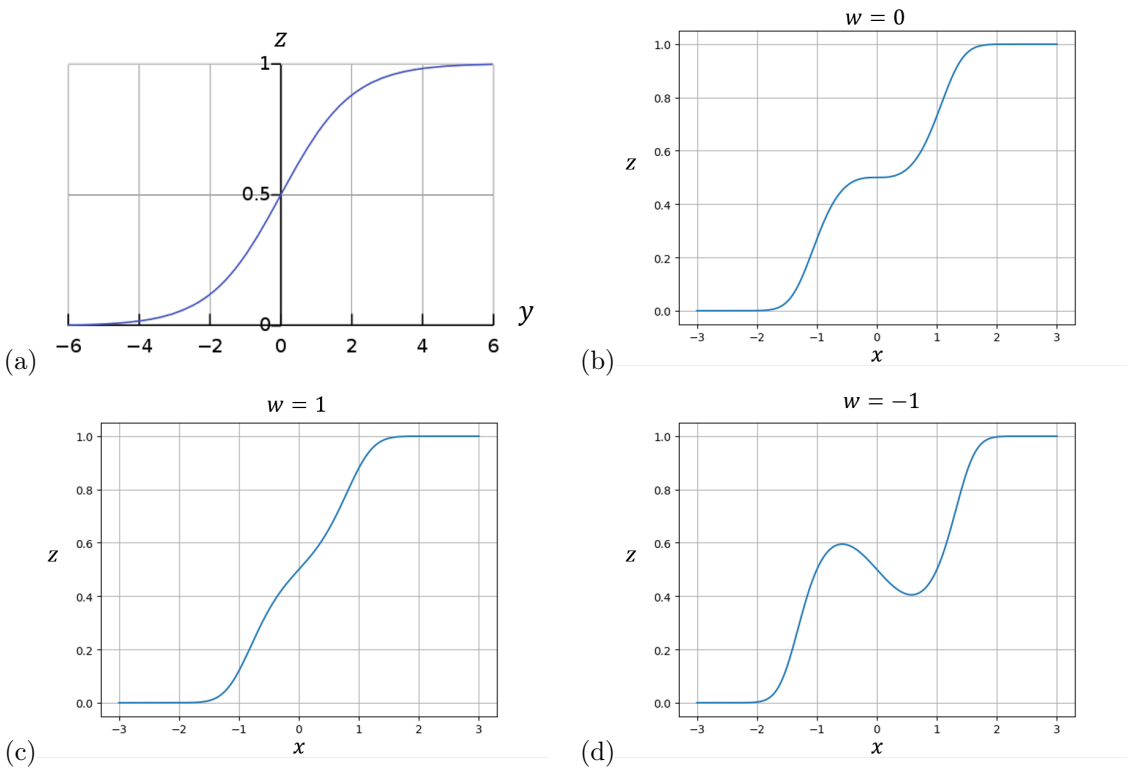


Figure 6.4: Function of a function. (a) illustrates the sigmoid function on its own. (b)-(d) illustrate z described in eq. (6.10) as a function of x, for different values of the weighting term w, (b) $w = 0$, (c) $w = 1$ and (d) $w = -1$.

then given by w^1_{11}. This gives us a powerful tool for adjusting the function shape and position, just from two parameters.

The second set of calculations takes us from the hidden layer to the output layer. Here we take the ReLU outputs, multiply them by more linear weights and sum again. In the design of the network I've actually made the final node a linear function, such that the output is a weighted sum of the hidden layer outputs. It is noted that if any of the weighting terms between the hidden and output layer are negative, the ReLU function will be flipped upside down.

This summing behaviour is illustrated for a simple example in Fig. 6.3(d). Here different ReLU functions have been summed up to produce a much more complex function. The power of combining multiple nodes together is that we can generate arbitrary functions like this, and increasing the number of nodes will increase the complexity we can capture.

6.1.2 Function of a function

C6F

Another key feature of the Neural Network can be observed by considering eq. (6.6). The neural network is a function of a function, at each point scaled, offset and summed. This further increases the expressive capability of the network. We can illustrate this with a simple 1D example.

We will define our first function as a simple polynomial

$$y = x^3 + wx \tag{6.9}$$

where we have used a single arbitrary weighting term w, which we will use to adjust the function. For our second function, we will utilise the sigmoid function (which we will discuss more about later). This is written as

$$z = \frac{1}{1 + e^{-y}} \tag{6.10}$$

and illustrated in Fig. 6.4(a). In Figs. 6.4(b) to (d) we see z as a function of x, for $w = 0$, $w = 1$ and $w = -1$ respectively. This has introduced a significant variation into the resulting function, just from changing a single parameter. It is clear that combining multiple functions like this across several layers can enable significant complexity. This is further enhanced by the combination of functions from multiple nodes within each layer, as discussed in Sect. 6.1.1.

6.1.3 Linear transfer functions

C6G

Previously I have highlighted the power of neural networks for expressing complex functional forms. We have emphasised the nonlinear nature of the functions as being key to this. To illustrate this further, let us consider a network purely consisting of linear transfer functions. This will be a three-layer network, i.e. will have an input, a hidden layer and an output. Taking a linear activation function $f(net_j) = A + Bnet_j$ for the hidden layer,

$$y_j = f(net_j) = A + Bnet_j \tag{6.11}$$

$$= A + B\sum_{i=0}^{n} w_{ji}^{1} x_i \tag{6.12}$$

we can express this function in terms of a new set of weighting terms

$$w_{ji}'^{1} = \begin{cases} Bw_{ji} & i > 0 \\ Bw_{ji} + A & i = 0 \end{cases} \tag{6.13}$$

giving

$$y_j = \sum_{i=0}^{n} w_{ji}'^{1} x_i. \tag{6.14}$$

Now let's take this forward to the second layer, which will have a similar linear activation function, so we will simply write out the equation

$$z_k = \sum_{j=0}^{nh} w_{kj}'^{2} y_j \tag{6.15}$$

which becomes

$$z_k = \sum_{j=0}^{nh} w_{kj}'^{2} \sum_{i=0}^{n} w_{ji}'^{1} x_i \tag{6.16}$$

$$= \sum_{j=0}^{nh} \sum_{i=0}^{n} w_{kj}'^{2} w_{ji}'^{1} x_i \tag{6.17}$$

$$= \sum_{i=0}^{n} w_{ki}'' x_i \tag{6.18}$$

where

$$w_{ki}'' = \sum_{j=0}^{nh} w_{kj}'^{2} w_{ji}'^{1}. \tag{6.19}$$

So we have effectively expressed our output z_k as a weighted linear sum of the input x_i, which removes almost all the benefit of neural networks. This serves to make the point that neural networks rely on non-linearity to achieve their results. It should be noted that linear functions can be used within neural networks, often as a scaling solution (one example where these were used was in Sect. 6.1.1), but having a network purely of linear functions serves no value.

6.2 Training neural networks

C6H

As with many machine learning algorithms the training approach is one of taking the training dataset and updating the coefficients such that when the training inputs are fed through the algorithm, they produce an output as close as possible to the corresponding training outputs. We will consider the conceptually simple concept of gradient descent of the l_2 norm to achieve this fitting. Most routines will be variations of this approach, so understanding this will be useful. It is noted at this point that the starting point is often to assign random numbers to each weighting value, and gradient descent steps from that towards the point that minimises the l_2 norm. For this least squares solution, the cost function is defined as[2]

$$J = \sum_{m=1}^{M} \frac{1}{2} \left(t_m - z_m\right)^2 . \tag{6.20}$$

Here we are using the terminology from the neural network field which differs from that learnt in Sect. 3.1.1, although the definition is mathematically identical[3]. t_m is the true value to which we are trying to fit data, and z_m is the current output from the neural network, both for the mth output value, from a total of M outputs. The cost function J matches the error E_2 from before. To undertake gradient descent we must first calculate the gradient of this cost function with respect to the weighting parameters.

6.2.1 Gradient calculation

C6I

Consider first that we calculate the gradient with respect to a weighting between the last hidden layer and the output layer. Following Fig. 6.2, we will say that this is the second set of weighting terms w^2_{kj}, so we wish to calculate $\frac{\partial J}{\partial w^2_{kj}}$. Differentiating eq. (6.20) gives

$$\frac{\partial J}{\partial w^2_{kj}} = \sum_{m=1}^{M} - \left(t_m - z_m\right) \frac{\partial z_m}{\partial w^2_{kj}} \tag{6.21}$$

where we have applied the chain rule. Note that we have both m and k parameters in this equation, which both are used to indicate nodes in the final layer. m is used when we iterate through to sum up the squares to generate the cost function, as described in eq. (6.20), while k indicates (along with j) the specific weighting term we are differentiating with respect to. Since $z_k = f(net^2_k)$, we can again apply the chain rule to give

$$\frac{\partial z_m}{\partial w^2_{kj}} = f'\left(net^2_k\right) \frac{\partial net^2_k}{\partial w^2_{kj}} \tag{6.22}$$

if $m = k$ and

$$\frac{\partial z_m}{\partial w^2_{kj}} = 0$$

otherwise[4]. This indicates that the derivative of the output from one particular node is only nonzero if taken with respect to a weighting which actually goes into that node, which should be unsurprising. Then, where nh is the number of nodes in the hidden layer

$$\frac{\partial net^2_k}{\partial w^2_{kj}} = \frac{\partial}{\partial w^2_{kj}} \sum_{j'=0}^{nh} w^2_{kj'} y_{j'} \tag{6.23}$$

$$= \sum_{j'=0}^{nh} \frac{\partial w^2_{kj'}}{\partial w^2_{kj}} y_{j'} + \sum_{j'=0}^{nh} w^2_{kj'} \frac{\partial y_{j'}}{\partial w^2_{kj}} . \tag{6.24}$$

[2] Note that the order of $t_m - z_m$ is irrelevant because of the squaring - these can be in either order. There is an exercise in the questions below about this.

[3] Through these notes I have generally tried to be consistent, which is a challenge given the different conventions across machine learning. Here I think it is worth being familiar with the standard for neural networks.

[4] *More detailed explanation:* Essentially the z_m value, at the m^{th} node, will be a function of the weighting values w^2_{mj} and any earlier ones in the network, and it will remain constant as any other weighting value in the second layer (w^2_{pq}) changes. So the derivative of z_m must be zero when taken with respect to any weighting value in the second layer which does not directly lead to m, i.e. when $k \neq m$.

From this, recognise that the y values are the nodal outputs in the second layer so do not depend on the weightings between the second and third, so the second term disappears to zero. The term $\frac{\partial w^2_{kj'}}{\partial w^2_{kj}}$ is one where $j' = j$, and zero otherwise (in exactly the same way we saw with m and k above). This means that the summation drops down to $y_{j'}$ for $j' = j$, so

$$\frac{\partial net^2_k}{\partial w^2_{kj}} = y_j. \tag{6.25}$$

Putting this all together gives

$$\frac{\partial J}{\partial w^2_{kj}} = -\left(t_k - z_k\right) f'\left(net^2_k\right) y_j. \tag{6.26}$$

At this point, have a think about what these terms are. We are differentiating with respect to the weighting w^2_{kj}, which is between hidden layer node j and output k. We have the residual at the output, $t_k - z_k$, the derivative of the activation function at that node, $f'\left(net^2_k\right)$, and multiply this by the output from the hidden layer node, y_j. The first and last terms are available directly from the forward calculations in the neural network, and the activation function derivative can typically be calculated easily.

We now consider differentiation with respect to weight w^1_{ji} in the first layer

$$\frac{\partial J}{\partial w^1_{ji}} = \sum_{m=1}^{M} -\left(t_m - z_m\right) \frac{\partial z_m}{\partial w^1_{ji}} \tag{6.27}$$

and as before, applying the chain rule to $z_m = f(net^2_m)$ gives

$$\frac{\partial z_m}{\partial w^1_{ji}} = f'\left(net^2_k\right) \frac{\partial net^2_m}{\partial w^1_{ji}} = f'\left(net^2_k\right) w^2_{mj} \frac{\partial y_j}{\partial w^1_{ji}} \tag{6.28}$$

where we have also differentiated $net^2_m = \sum_{j'=0}^{nh} w^2_{mj'} y_{j'}$ in a similar way to eq. (6.23). Then

$$\frac{\partial y_j}{\partial w^1_{ji}} = f'(net^1_j) \frac{\partial net^1_j}{\partial w^1_{ji}} = f'(net^1_j) x_i \tag{6.29}$$

so

$$\frac{\partial J}{\partial w^1_{ji}} = -f'(net^1_j) x_i \sum_{m=1}^{M} \left(t_m - z_m\right) f'(net^2_m) w^2_{mj}. \tag{6.30}$$

Unsurprisingly, since the earlier weighting affects a larger amount of the network, there are more terms which appear in the calculation. This could be problematic for many of the very deep multi-layered models, where computation would become prohibitively expensive during training. However, inspection shows that many of these terms are repeated from other calculations; consider eq. (6.26) which contains the $\left(t_m - z_m\right) f'(net^2_m)$ term which can then just be reused in eq. (6.30).

The gradient calculation approach through the full network then consists of

1. Performing a forward calculation, recording all the net activation values and nodal outputs
2. Calculating the residuals
3. Stepping back through each of the layers, calculating the derivatives for all the weighting values in that layer with respect to the overall cost function J, and recording the terms to then be reused in the earlier layers.

6.2.2 Training via backpropagation

C6J

For gradient descent, we wish to step a small amount η in the opposite direction to the gradient, so the updates can be written as

$$\Delta w^1_{ji} = \eta f'(net^1_j) x_i \sum_{m=1}^{M} \left(t_m - z_m\right) f'(net^2_m) w^2_{mj} \tag{6.31}$$

$$\Delta w_{kj}^2 = \eta\,(t_k - z_k)\,f'\left(net_k^2\right) y_j. \tag{6.32}$$

The approach of calculating gradient values, and hence these update terms, by stepping backwards through the network is called BACKPROPAGATION, and can be thought of as propagating the residuals at the last layer back through the network to update the weighting terms, to improve the output. It should be noted that there are alternative training approaches (and you may wish to read about these online), but they typically utilise some form of backpropagation.

6.2.3 Training approaches

C6K

The above derivation has assumed we have a single data point to fit the network to. In reality we have multiple data points, and we wish to achieve a good fitting to all of these. There are two main methods to achieve this. In STOCHASTIC TRAINING we randomly select a data point from the training set and update the weighting values according to this data point each time, then repeat the process. In BATCH TRAINING all points in the dataset are presented to the network, then the weights are updated for all of them at the same time. This is likely to be smoother, since it will minimise sensitivity to outliers, but will be slower. It is also possible to compromise between these, by presenting a subset at a time then rotate through these; we may need to select a suitable BATCH SIZE to do this.

Typically the data is presented to the network multiple times during training, to iteratively fit the network to it. An EPOCH is a measure of the relative amount of learning through these different methods; it represents the number of times the full dataset has been exposed to the model. For example, if 3 epochs have been used to train the model with a dataset of m samples, for stochastic training this would mean $3m$ updates, while for batch training there would just be 3.

6.3 Transfer functions

C6L

As highlighted in Sect. 6.1.3, neural networks rely on non-linear transfer functions to capture the complexity of the underlying functions. We will review some common choices here.

6.3.1 Step function

The STEP FUNCTION is illustrated in Fig. 6.5(a) and defined as

$$f(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases} \tag{6.33}$$

although it should be noted that alternative values can be used instead of 1 and -1 (0 and 1 are common) and the discrimination point need not be 0 as in this case. The case here is the sign function ($+1$ for positive, -1 for negative, and can also be written in a general form as $f(x) = x/|x|$) The function is conceptually simple and quick to calculate, and in producing a binary output is often useful in the final layer of a neural network to perform classification. A downside is that the gradient is zero in the flat sections, which can make training more of a challenge.

6.3.2 Sigmoid function

The SIGMOID FUNCTION can be considered a smooth version of the sign function. It is illustrated in Fig. 6.5(b), and defined as

$$f(x) = \frac{1}{1 + e^{-x}} \tag{6.34}$$

The derivative is easy to calculate (this is a question at the end of the chapter). The smoothing behaviour makes the gradient better behaved than a step function, although it should be noted that this does disappear to zero at large positive and negative inputs, which could make training difficult.

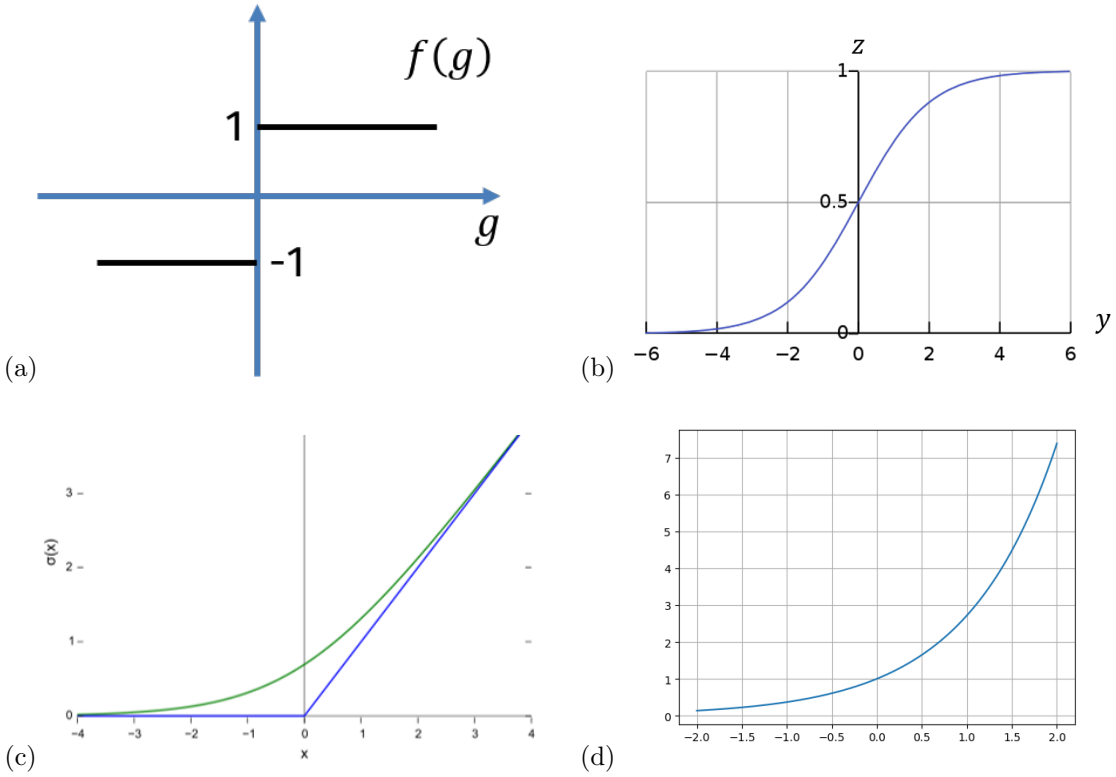


Figure 6.5: Neural network transfer functions. (a) shows the step function, (b) sigmoid, (c) ReLU and softplus and (d) softmax.

6.3.3 Rectified Linear (ReLU) function

Section 6.1.1 introduced the ReLU function, defined as

$$f(x) = \begin{cases} x & x > 0 \\ 0 & otherwise. \end{cases} \tag{6.35}$$

and is illustrated as the straight lines in Fig. 6.5(c). It is a very fast function, and popular because of this.

6.3.4 Softplus function

The curve in Fig. 6.5(c) illustrates the softplus function, a smoothed version of the ReLU. It is mathematically described as

$$f(x) = \log\left(1 + e^{x}\right). \tag{6.36}$$

It has the advantage over ReLU of being differentiable throughout the range of all possible x values:

$$f'(x) = \frac{e^{x}}{e^{x} + 1} = \frac{1}{1 + e^{-x}} \tag{6.37}$$

although the function is more computationally demanding than the standard ReLU.

6.3.5 Softmax function

The softmax is an exponential function, which is then normalised against all other outputs so they sum to one:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{nOutputs} e^{x_j}}. \tag{6.38}$$

This has value when dealing with probabilities, which must sum to one, so is often valuable as the last layer of the neural network. The softmax is closely related to the sigmoid function. If we consider the first output for a scenario with two outputs, then

$$f(x_i) = \frac{e^{x_1}}{e^{x_1} + e^{x_2}} \tag{6.39}$$

$$= \frac{1}{1 + e^{-x_1} e^{x_2}}. \tag{6.40}$$

By ignoring the scaling of the other term, e^{x_2}, it is clear to see we get the sigmoid function.

6.3.6 Choosing transfer functions

There is no clear right answer about what is the best transfer function for all cases. As with much of machine learning, it is usually necessary to test out different options and assess performance, noting that you should always validate.

Training is a significant consideration; you typically want transfer functions which are easy to differentiate and calculate values for. You may wish to have different types at different locations, particularly on the output nodes where you may want to output probability or class information. You can also include linear functions early in the network to allow scaling to occur.

6.4 Designing neural networks

C6M

Neural networks are amongst the most empirical of the machine learning algorithms[5]. They make no attempt to replicate real mechanisms in their structure and the complexity which is incorporated as described in Sects. 6.1.1 and 6.1.2 is essentially arbitrary. The success of neural networks is that this complexity is easy to generate and once designed and properly trained, the neural networks are often highly efficient. This does mean that it is not possible to define firm rules about how the network should be designed. The following tips can, however, help identify somewhere to start.

- Find an example where someone has tried to solve a similar problem; implement this then try to improve it
- For any problem you should be able to identify the numbers of inputs and outputs; this helps to define some of the architecture
- In many cases you can get away with just a single hidden layer
  - You can define a simple model based on these points by setting the hidden layer size to the mean of the inputs and outputs
- Start simple then expand if necessary, if you find that bias is high

Note that to fully capture the complexity of some problems, it is necessary to have a large amount of training data. The specific amount required is difficult to know in advance, but validation should help to confirm this.

It should be noted that this is the fastest-moving area of a fast-moving topic, so while you can expect ME1 Mechanics to remain the same for many years, the same is not true for the state of the art in neural networks and the "best practice" may well change over time.

6.5 Types of neural networks

Much attention at the moment is given to deep learning, where many-layered neural networks are used to capture significant complexity of underlying datasets. This has become possible because of the improvements in computer hardware, and also the large quantities of data. Deep learning cases are seen to incorporate feature extraction, where the early layers exact features (e.g. edges, shapes)

[5] And it should be recognised that the entire subject is, almost by definition, empirical.

and the later layers process these. Other forms of neural networks, such as recursive (e.g. for time-varying data) and convolutional (e.g. for image processing) neural networks exist too, but these are beyond the scope of the course.

6.6 Libraries

C6N

As with many of the topics studied in this course, there are libraries for implementing neural networks. These avoid the pitfalls of coding software yourself, focusing instead on implementing tools and getting results. Scikit-learn has been used for many of the tutorials, but is a general ML library; neural networks are quite specialist. The main libraries are TensorFlow (from Google) and PyTorch (from Facebook). Then there are higher level libraries which plug into these, such as Keras with TensorFlow[6] and fast.ai with Pytorch. In the tutorials, we will use Keras, because the form is very similar to Scikit-learn. Keras gives good flexibility without significant complexity.

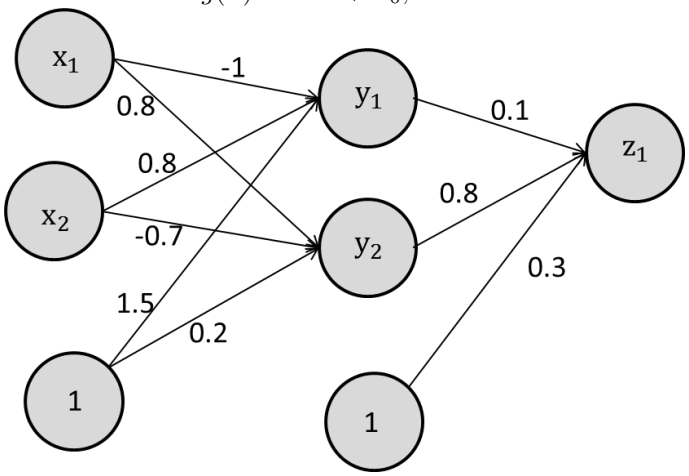
6.7 Conclusions

In this chapter we've looked at neural networks. You should have learnt what the structure is, including how they bring in complexity through the use of nonlinear functions. You should understand how they are trained, including how the gradient can be calculated to enable the cost function to be minimised, and you should know how they can be used in practice. It is good to play around with these things; https://playground.tensorflow.org gives a great visual representation of this for a two parameter problem.

6.8 Questions

1. Show that $J = \sum_{m=1}^{M} \frac{1}{2} (t_m - z_m)^2$ and $J = \sum_{m=1}^{M} \frac{1}{2} (z_m - t_m)^2$ both produce the same value when differentiated with respect to z_m.

2. I have the neural network shown below. Activation functions at each node are unscaled linear functions, i.e. $f(x) = x$. Show that this network is equivalent to a single linear discriminant function of form $g(\boldsymbol{x}) = \boldsymbol{w}^t \boldsymbol{x} + w_0$, and find what $\boldsymbol{w}$ and w_0 are.



3. The sigmoid function is defined as $f(x) = \frac{1}{1+e^{-x}}$. Calculate the derivative f' with respect to x and show that this can be expressed in terms of $f(x)$ itself.

[6] Note that Keras does support other backends

6.8.1 Solutions

1. $\frac{\partial J_1}{\partial z_m} = \sum -(t_m - z_m)$, $\frac{\partial J_2}{\partial z_m} = \sum (z_m - t_m)$, i.e. they are equal.

2. (Q13 2020)
   $(-1x_1)\cdot 0.1 + 0.8\cdot x_1 \cdot 0.8 + 0.8\cdot x_2 \cdot 0.1 - 0.7\cdot x_2 \cdot 0.8 + 1.5\cdot 0.1 + 0.2\cdot 0.8 + 0.3 = 0.54\cdot x_1 - 0.48\cdot x_2 + 0.61$
   $g(\boldsymbol{x}) = \boldsymbol{w}^t\boldsymbol{x} + w_0 \quad \Rightarrow \quad w_0 = 0.61, \quad \boldsymbol{w} = \begin{pmatrix} 0.54 \\ -0.48 \end{pmatrix}$

3. $$\sigma(x) = \frac{1}{1+e^{-x}}$$
$$\frac{d\sigma}{dx} = \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right) = \frac{d}{dx}[(1+e^{-x})^{-1}]$$
$$= -(1+e^{-x})^{-2}\cdot(-e^{-x})$$
$$= \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}}\cdot\frac{e^{-x}}{1+e^{-x}} = \frac{1}{1+e^{-x}}\cdot\left(\frac{1+e^{-x}}{1+e^{-x}} + \frac{-1}{1+e^{-x}}\right)$$
$$= \sigma(x)\cdot(1-\sigma(x))$$

Chapter 7

Non-parametric methods

Non-parametric methods are techniques which do not describe models fitted to data in parametric forms. We will discuss what this means in this chapter. Firstly, we will review some basics about probability, to establish the context.

7.1 Probability density functions

C7A

Consider that we have a particular "true" probability density function $p(x)$, which describes the probability for a random variable x. We acquire a set of data which conforms to this probability density function. As discussed in Chapter 2, if we can estimate the probability density function from this training data, we can then use this to make classification decisions in a general scenario. We can assume a normal distribution (or any distribution) and do a best-fit of the mean and standard deviation. But in some cases the shape is too complex to fit an existing distribution form so instead we must estimate from the training samples. Since we are not estimating distribution parameters, this is a non-parametric approach. While we will initially consider probability, the concept does extend into a range of non-parametric ML models.

7.2 Probability density estimation

C7B

The probability density function can be integrated to give a probability that x will lie within a range as

$$P(x_1 < x < x_2) = \int_{x_1}^{x_2} p(x)dx \tag{7.1}$$

and we can approximate the integral as

$$P(x_1 < x < x_2) = \int_{x_1}^{x_2} p(x)dx \approx p(x)(x_2 - x_1) = p\,(x)\,h \tag{7.2}$$

where $h = x_2 - x_1$ and we have made the assumption that $p(x)$ doesn't vary significantly within the range $x_1 < x < x_2$, by assuming that the distance h is small. We now consider that we have m samples, each drawn from the distribution. In each case, we have the same probability P that they lie within the range given. Then, the probability that k of the m fall within the range $x_1 < x < x_2$ is given by the binomial law

C7C

$$P_k = \left(\begin{array}{c} m \\ k \end{array} \right) P^k \left(1 - P\right)^{m-k}. \tag{7.3}$$

The binomial distribution describes the scenario where there are m repeated binary tests (e.g. coin flips or rolling six on a dice), and you wish to know the probability of getting k positive results. We do not need much detail about the binomial distribution, beyond recognising that the number of points expected within the region is given by

$$E(k) = mP. \tag{7.4}$$

For example, given 10 samples, and a 20% chance of each one individually falling within the region, the expected number of points in the region is $10 \times 0.2 = 2$. This should be fairly intuitive to understand.

If we have our training dataset, and k samples now fall within the region, we can estimate this probability by rearranging eq. (7.4)

$$P = \frac{k}{m}. \tag{7.5}$$

Then since $P = p(x)h$, the density can be estimated as

$$p(x) = \frac{k}{mh}. \tag{7.6}$$

This represents the fundamental concept we are applying when estimating probability.

7.2.1 Higher dimensional space

C7D

When n parameters are present in the data, we end up with a probability density function in n-dimensional space. Rather than the linear length h, the region of interest becomes an n-dimensional hypercube of side length h. The approach which can be taken is then to count up the number of points within that region, and estimate the probability as

$$p(x) = \frac{k}{mV} \tag{7.7}$$

where $V = h^n$ is the volume of the cube.

7.2.2 Accuracy of estimate

C7E

There are two elements to the accuracy. One is the spatial resolution, i.e. how small is the region over which we are averaging the probability. If this is small, the output will be more representative of the local value. However, we must also consider that a small volume will also contain fewer points, and therefore the ratio of k/m is likely to be less accurate. Variance will be larger for this scenario, as the output will be more dependent on the specific sample set. A larger volume will get more points and increase the probability accuracy, but this will be spread over the larger volume, so variance should be reduced. There is a trade-off between variance and spatial averaging. An alternative is to somehow acquire more training samples. Under this, k/m will converge, and we will obtain a more accurate estimation of probability. However, acquiring data can be expensive, so this is often undesirable.

7.3 Parzen windows

C7F

Parzen windows provide a mathematical framework for systematically performing the probability density estimation, based on the above description. We can define a variable

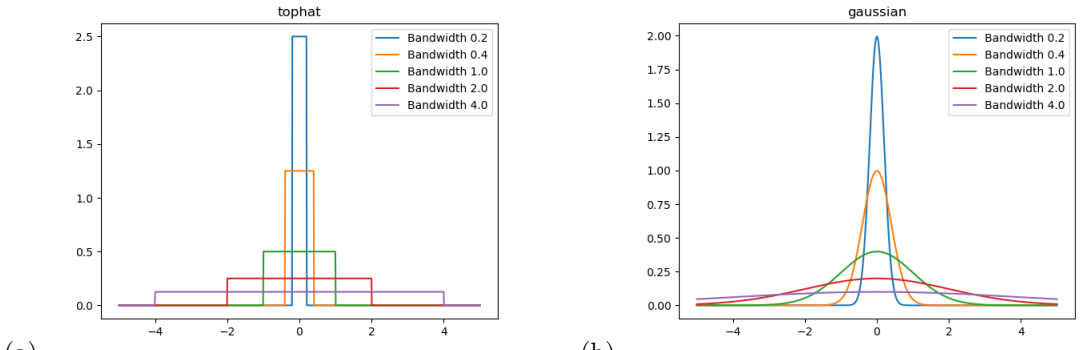
$$u = \frac{x - x'}{h} \tag{7.8}$$

where h is the width of the window and x' defines the window position.. Based on u, a window function can be defined as

$$\phi(u) = \begin{cases} 1 & |u| < \frac{1}{2} \\ 0 & otherwise \end{cases}. \tag{7.9}$$

This means that $\phi(u) = 1$ for any point within the window centred at x', and 0 otherwise. This can be used in probability estimation. We can apply the function to the samples in our dataset x_i; the number of samples in the window is calculated as

$$k = \sum_{i=1}^{m} \phi\left(\frac{x_i - x'}{h}\right) \tag{7.10}$$



(a) (b)

Figure 7.1: Parzen window examples. (a) gives the top hat window with different bandwidths (note bandwidth $= h/2$ following the definitions in the text). (b) shows the Gaussian window with different bandwidths.

which leads to our probability estimate becoming

$$p(x) = \frac{1}{m} \sum_{i=1}^{m} \left[\frac{1}{h} \phi \left(\frac{x_i - x'}{h} \right) \right]. \tag{7.11}$$

Consider the k calculation term given by (7.10). This defines the window position centred at x', and every sample which lies within the range is then added up. Note that the $1/h$ term is included with the function here; this is done to ensure that the window function integrates to 1 and hence is a valid probability density function (PDF) (note that it also must be non-negative throughout for this to hold). Provided this is the case, it will be a valid Parzen window which is required to be a valid PDF.

It is possible to conceptually consider this equation the other way around. Since ϕ is symmetrical, i.e. $\phi(u) = \phi(-u)$, we can also consider that

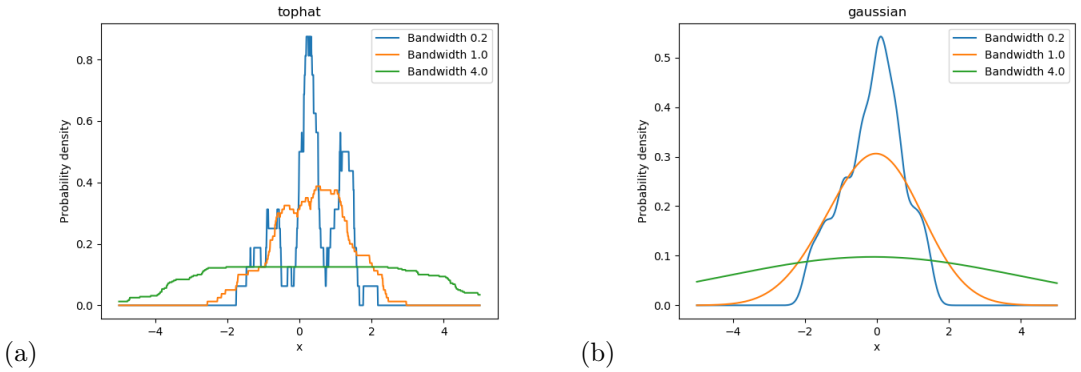
$$p(x) = \frac{1}{m} \sum_{i=1}^{m} \left[\frac{1}{h} \phi \left(\frac{x' - x_i}{h} \right) \right] \tag{7.12}$$

which instead corresponds to a function centred at each sample x_i, which are summed up to make one single function which represents the probability. The final function can be seen to be a sum of many Parzen windows. C7G

It is possible for Parzen windows to have other forms than the "top hat" considered here. Typically these will be smoother functions, which will result in smoother probability estimates. Figure 7.1 illustrates both the top hat and the Gaussian windows in (a) and (b) respectively, showing examples of different shapes. This also illustrates how the bandwidth (or h) can change; the height simultaneously adjusts to ensure that the window integrates to 1.

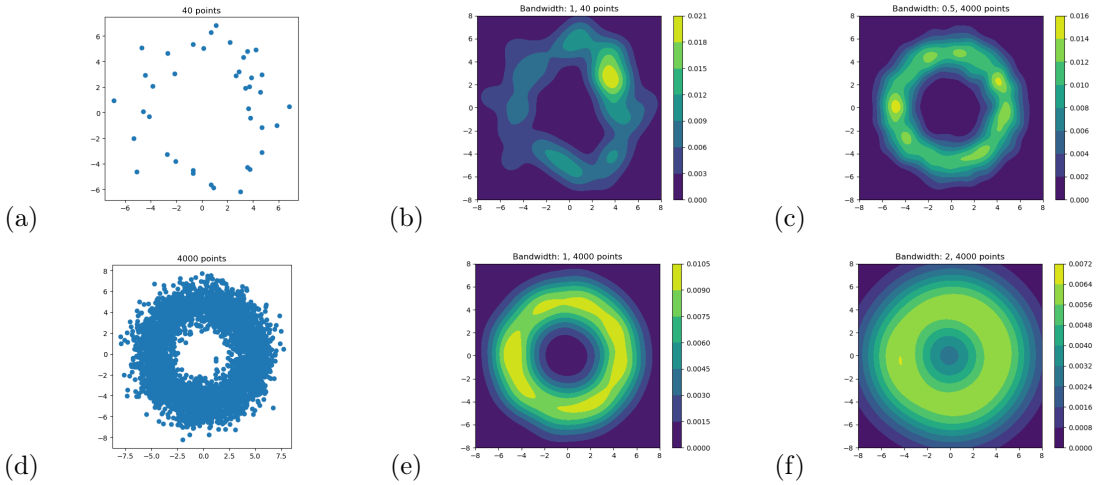
The effect of adjusting the bandwidth on the resulting estimate can be seen in Fig. 7.2. From Fig. 7.2(a), it is clear that a narrow bandwidth is very sensitive to the specific realisation of the dataset, with the 0.2 value having many spikes. By contrast, with a bandwidth of 4.0, the detail gets heavily removed. A bandwidth of 1.0 seems like a good compromise here. The smoothness of the Gaussian window in Fig. 7.2(b) seems to perform better for this problem (although it is noted that the underlying Normal distribution from which the points are sampled is itself a Gaussian). All three curves perform better than their top-hat counterparts, with the bandwidth of 1.0 appearing to form a very good estimate.

The Parzen windows have been defined so far in a single dimension; the concept works in higher parametric dimensions too. Figure 7.3 illustrates an example in 2D space with Gaussian windows. The two sets of points are illustrated in Fig. 7.3(a) and (d). Figure 7.3(b) gives the estimate from 40 points with a bandwidth of 1, showing a significant sensitivity to the specific point realisation. Figure 7.3(c) shows the bandwidth of 0.5 with 4000 points, (e) with bandwidth 1 and (f) with bandwidth 2. The conclusions are the same as for the 1D case; more points gives



(a) (b)

Figure 7.2: Estimations of a probability distribution from 40 samples, for the top hat (a) and Gaussian (b) windows. The underlying points were normally distributed ($\mu = 0$, $\sigma = 1$).



(a) (b) (c)

(d) (e) (f)

Figure 7.3: 2D Parzen windows. (a) and (d) show distributions of 40 and 4000 points respectively in 2D space. (b) shows Gaussian window estimations of the probability density for a bandwidth of 1 based on the 40 points of (a). (c) (e) and (f) use the 4000 points in (d) with the Gaussian window and bandwidths of 0.5, 1 and 2 respectively.

a better estimate, and increasing the bandwidth will increase the averaging and reduce sensitivity to the specific realisation expressed by the sample dataset, but reduce detail too.

7.3.1 Classification

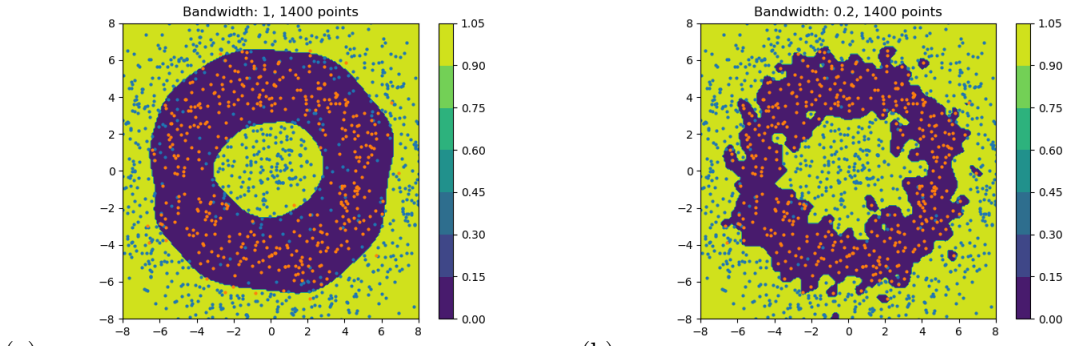
As with previous techniques, it is possible to classify based on these probability distributions simply by allocating to the class with the highest probability; the bandwidth can control the amount of fitting, i.e. the bias and variance. Figure 7.4 gives an example of this for two different bandwidths.

7.4 Nearest Neighbours

C7H

We have highlighted that the bandwidth/size of the Parzen window is not straightforward to select. A particular issue is that the optimal value at one location in the distribution will not be optimal elsewhere; since the probability density function will vary throughout the domain, the size needed to generate a smooth-yet-accurate result will vary. Nearest Neighbour approaches address this by effectively automatically adjusting the size.

The Nearest Neighbour approach is to compare the test point (which we wish to classify) against



(a) (b)

Figure 7.4: Classification based on Parzen window probability estimations for a two-category problem, given 1400 sample points. (a) shows a bandwidth of 1 and (b) a bandwidth of 0.2.

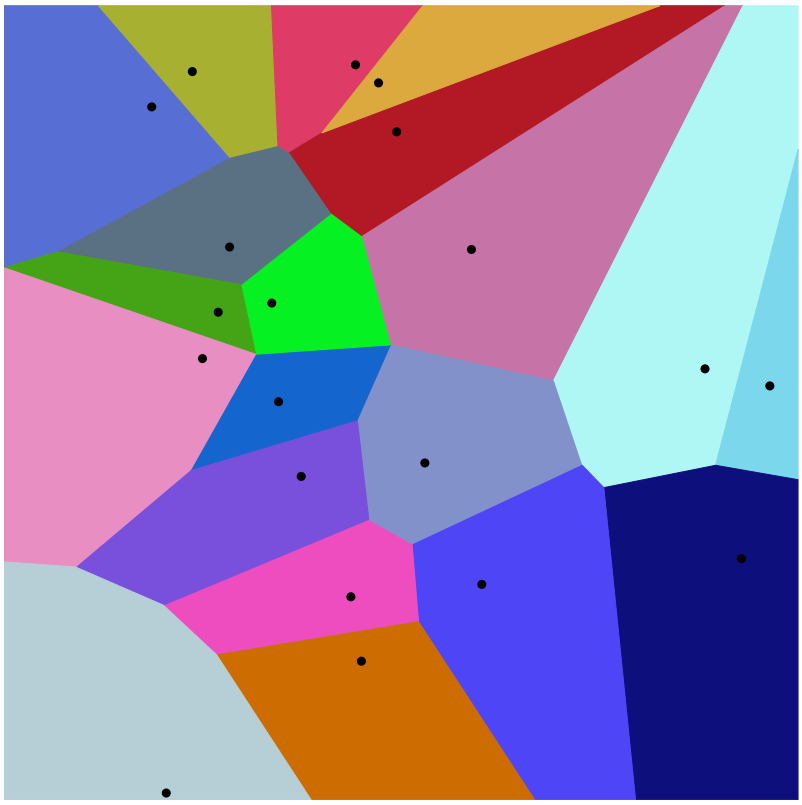
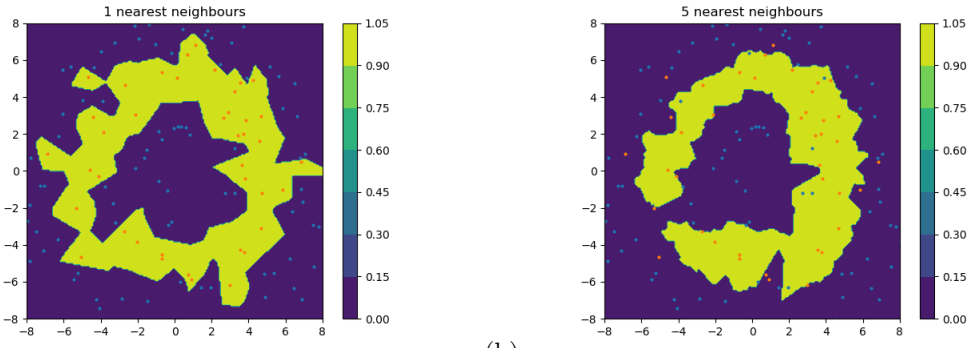


Figure 7.5: Example of a Voronoi diagram. The domain is coloured according to which point is nearest.



(a) (b)

Figure 7.6: k-Nearest-Neighbour classification for 1 and 5 nearest neighbours in (a) and (b) respectively.

each point in the training dataset. We then classify this as having the same class as the nearest training point. This is effectively partitioning the space into Voronoi cells, if you are familiar with these[1]. Figure 7.5 illustrates this; the domain is subdivided into sections, and these are coloured according to which training point is the nearest. Linking back to the window techniques, we could consider it to be a scenario where we place a window at the point of interest, then increase its size until it hits the first neighbour; in this way it adapts to the local sampling density.

Nearest neighbours rely on calculating a distance, which is dependent on the concept of a metric, i.e. the similarity between two points. Sect. 11.4 discusses metrics in full detail. Throughout this discussion we have just considered the Euclidean metric as our distance, but the concept can be extended to any metric.

7.4.1 k-nearest-neighbour

C7I

The K-Nearest-Neighbour (KNN) approach extends the Nearest Neighbour method to incorporate more than a single neighbour. This can reduce overfitting, i.e. the sensitivity to one specific neighbour. The approach is to start from the point we wish to classify, and increase the size of a window centred on it until it captures k training samples, i.e. the k nearest-neighbours to the point. These neighbours each then vote on what the class should be at the original position. Based on this, for a two-category problem, it is sensible to make k odd to avoid ties. Figure 7.6 shows the algorithm for 1 and 5 neighbours, showing how 5 neighbours is less sensitive to outliers and therefore has lower variance.

A big advantage of the KNN approach is that it is a simple algorithm - it is easy to conceptually explain and quite intuitive. It can also capture quite a lot of complexity. However, it is computationally intensive, since lots of distance measurements must be taken and comparisons made, although there are possibilities to optimise this. It is also necessary to store all training points to calculate the nearest neighbours; compare this to the linear decision boundary, which is a very lightweight classification tool.

7.5 Summary

This chapter has covered probability density estimation via Parzen windows, as well as the k-Nearest-Neighbour approach and how this effectively adjusts the window size depending on the local sampling density. These are some valuable tools in cases where the data do not conform well to specific functions.

[1] Voronoi diagrams are pretty cool. There is quite a lot on the Wikipedia page https://en.wikipedia.org/wiki/Voronoi_diagram and it's fairly accessible – for interest you might want to have a read.

7.6 Questions

1. I have a parzen window defined as

$$\phi(x) = \begin{cases} \frac{3}{4}(1-x^2) & -1 < x < 1 \\ 0 & otherwise \end{cases}$$

   (a) Show that this is a valid parzen window.

   (b) I have a dataset consisting of a single point at $x = -1$. Write out an expression for the resulting probability distribution estimated by the parzen window defined.

7.6.1 Solutions

1. (Q4 2020) a)

$$\begin{aligned}\int_{-\infty}^{\infty} \phi \, dx &= \int_{-1}^{1} \frac{3}{4}(1-x^2)\, dx \\ &= \frac{3}{4}\left[x - \frac{x^3}{3}\right]_{-1}^{1} \\ &= \frac{3}{2}\left(1 - \frac{1}{3}\right) \\ &= 1\end{aligned}$$

$\therefore$ integrates to 1.
$x^2 \leq 1$ in range, so $1 - x^2 \geq 0$, hence $\phi(x) \geq 0$
$\therefore$ valid probability distributions.
$\therefore$ valid parzen window.

b)

$$p(x) = \begin{cases} \frac{3}{4}\left[1 - (x+1)^2\right], & -2 < x < 0 \\ 0, & otherwise \end{cases}$$

Chapter 8

Nonmetric methods

C8A

Nonmetric methods are a class of machine learning models. The name "nonmetric" relates to the definition of a metric; see Sect. 11.4 for full information on metrics. Metrics measure closeness between two points in a dataset. Clearly, in machine learning, where we are trying to pull out relationships between different data points, this concept of "closeness" is very important.

However, in some scenarios, data cannot be described by a metric in this way. An example would be the manufacturer of a particular component. The concept of "closeness" does not apply well when we try to compare these manufacturer names and we must therefore derive suitable alternative approaches. It should be noted that it may be possible to find a metric, but that this may be very unsuitable; for example, one may be able to do a comparison, letter-by-letter of the written names and measure the overall alphabetical distance between the two. However, this is extremely unlikely to have any influence over whether the lawnmower from Jones Lawnmowers Inc. is likely to perform better than the equivalent model from Smiths Garden Machinery Ltd.

C8B

Nonmetric methods do not rely on metrics to function. They can operate on nonmetric quanties, such as the names discussed before. It should be recognised that nonmetric methods can be applied to metric data too; while they do not rely on the concept of metrics, this does not mean that they cannot be applied.

Credit is given to Naomi Shipway who provided diagrams used for decision trees and random forests.

8.1 Decision trees

C8C

Decision trees are one of the earliest forms of machine learning, and can deal with nonmetric data. An example of a decision tree is given in Fig. 8.1. The concept is straightforward: for a given decision tree, we will start at the top and step through the tree, taking a path depending on the answers to a series of questions.

In terms of architecture, decision trees are flexible. The same question ("Size?" for example) can appear multiple times, and at each point we can have different numbers of splits. In practice, for programming reasons, it can be better to utilise just binary decisions (i.e. each path splitting into two at each node); in practice it is possible to achieve the equivalent effect as a higher number of splits by chaining multiple binary splits across many layers. We can illustrate the use of decision trees with an example about house purchasing[1]. Table 8.1 provides some example test data.

In Fig. 8.2(a) we plot two parameters from such data, the cost and the number of bedrooms. The decision tree then aims to place decision boundaries in this data to classify it. For example, we can see that we can separate a set of points which did not sell within the month by putting a boundary at a cost of £300k, and taking the points less than that. The remaining points above can then be separated in a similar way, but this time by classifying on the number of bedrooms; above 7 or below 3 sold within a month. Finally the residual can be split by a boundary at £700k on the price, with anything above that limit selling in the month and everything else not.

[1] For some reason there are a lot of examples in machine learning (and particularly decision trees) where house prices are predicted.

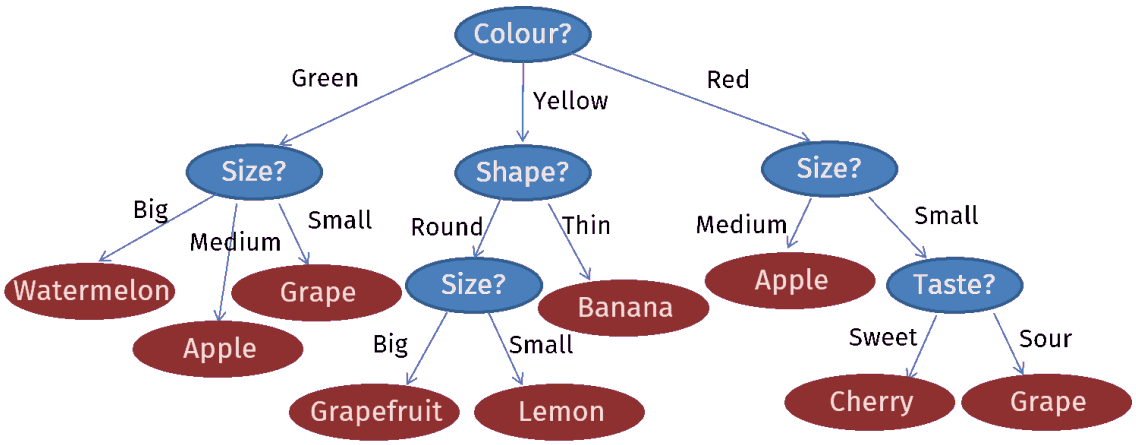


Figure 8.1: Example of a decision tree. We propagate down the tree from the top, taking a path depending on the answers to a series of questions. This ultimately classifies a particular input value.

Training data set		1	2	3	4	5	6
Sold within a month	y	N	Y	Y	Y	N	N
Cost (/£1000)	x_1	785	250	365	1,500	155	595
Number of bedrooms	x_2	7	2	3	4	2	5
Number of bathrooms	x_3	3	1	4	2	1	2
Distance to nearest city (/miles)	x_4	9	2	3	0	0.5	3
Garage?	x_5	Y	N	N	N	N	Y
Age of house (/years)	x_6	217	1	10	110	80	50

Table 8.1: House sale example data for decision tree example.

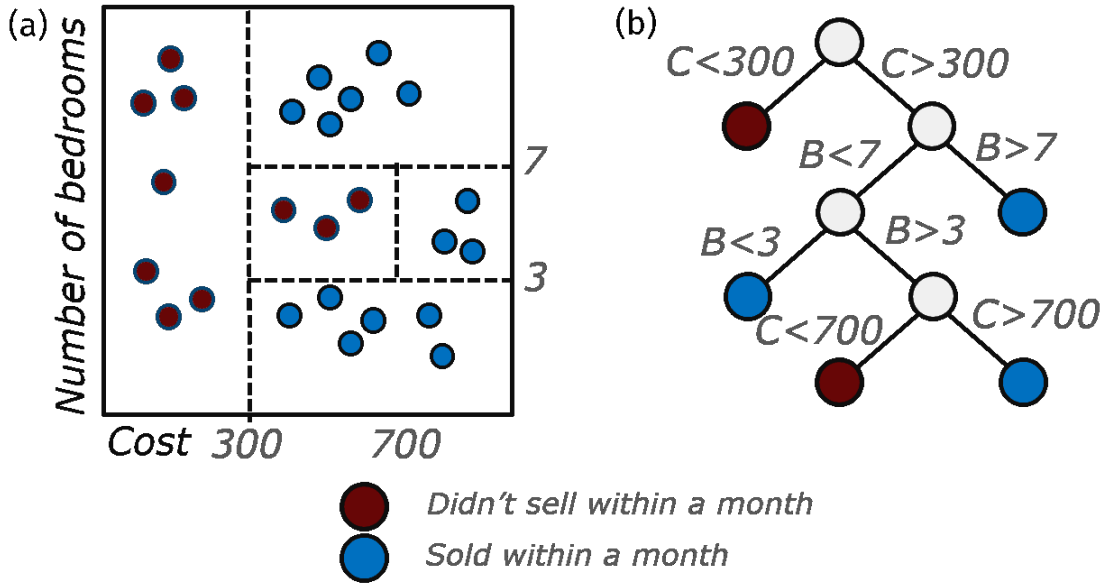


Figure 8.2: Construction of a decision tree. (a) shows the 2D parametric space for number of bedrooms vs cost, with the decision boundaries marked, and (b) gives the corresponding decision tree.

At each split, we are making a decision. This is reflected in the decision tree. The cost $<$ £300k decision we made first is at the top of the tree, immediately identifying some of the data points for houses which didn't sell in a month. The residual then passes down to other branches. C8D

Training the decision tree in practice is similar to the process described above. The algorithm will need to analyse the training data made available to it and produce suitable decision boundaries which (we hope) will generalise to other data. As a general rule, we wish to minimise complexity, i.e. the size of the tree. This minimises the computational cost, as well as the potential for the algorithm to overfit to the training data.

There are various decisions which must be made in designing the tree, including:

- How many splits at each node?
  - Binary (into two) is simpler and any decision with more splits can be generated as a chain of several splits
- Which property to split on?
- When do we stop splitting (i.e. decision made)?

As with almost all machine learning approaches, making these decisions is somewhat arbitrary, and typically established empirically, through experience with prior similar problems, and validation approaches (see Sect. 11.2). At a more fine-grained level, the fitting algorithm must decide where to place decision boundaries, and we therefore need some measure to indicate that one particular boundary position is better than another. The concept of IMPURITY can be used for this. C8E

Impurity is a measure of how impure the training values reaching a particular node are. If impurity is zero, this means that all the cases reaching that node belong in a single category - an example of this would be the coloured nodes on Fig. 8.2(b). A larger impurity indicates some mixing, to the largest values if each category is equally represented. A common example of impurity is the Gini impurity:

$$i\,(N) = \sum_{i \neq j} P(\omega_i) P(\omega_j) = 1 - \sum_j P^2(\omega_j)$$

which is the expected error rate if the category label was selected randomly from the class distribution present at the node. There are other measures, but most have the same behaviour.

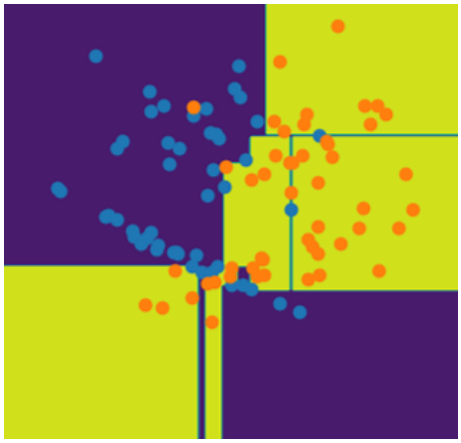


Figure 8.3: An example of a 2D parameter space populated by training data (blue and orange) upon which a decision tree has been trained. The dark blue and yellow areas are the corresponding classification regions from the decision tree, showing the blocky nature.

Impurity can be minimised by optimisation approaches; we have already discussed gradient descent in Sect. 3.4. This process is easiest with binary decisions since only a single boundary needs to be established. C8F

Potentially, it is possible to continue splitting based on the training dataset until we achieve 100% purity, but this will introduce a risk of overfitting, i.e. the method will not generalise well. Instead a typical approach is to split until a certain number of layers are defined, or a certain fraction or number of training points are left at a node. It should be noted that if it is defined as a certain number of training points, the leaves (final nodes of the tree) will scale to the problem; if there is an area dense with points then there will be more leaves in that area.

The decision tree is conceptually simple and well established, which are clear positives. As highlighted, it is a non-metric technique, so it does not require metric data. It also provides a framework for combining other classification techniques; any classifier we have looked at on the course could be applied to the data at a particular node. The downsides are that certain problems may require very complex trees to successfully classify, and the decision surfaces, typically being aligned with one classification parameter, will often result in very "blocky" boundaries which may unrealistic and not generalise well. Figure 8.3 provides an example of this.

An individual decision tree can suffer significantly from overfitting. There are some arbitrary decisions taken in how a tree is trained which can produce a very different result. Instead, a common approach is to instead design multiple trees and combine the results; these are random forests.

8.2 Random forests

C8G

Random forests train multiple decision trees based on random selections of features (and often a random subset of the data too). When applied, each tree makes a decision based on the data available, then the trees vote to give the final result. This has a number of benefits over decision trees. The method reduces overfitting, as illustrated in Fig. 8.4; while the blockiness of the individual decision trees is still clearly present, the use of multiple trees helps to smooth this. As is also visible in Fig. 8.4, it is possible to obtain a probability from the number of trees which vote each way. Knowing this uncertainty about a decision is very powerful in machine learning.

8.3 Conclusion

This chapter has discussed non-metric methods for machine learning, specifically looking at decision trees and how these are expanded to random forests. Non-metric methods can be used for (but

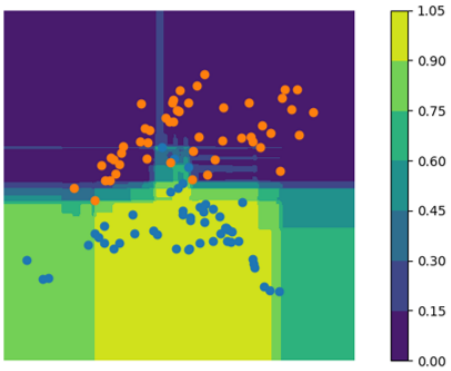


Figure 8.4: An example of classification by a random forest.

are not limited to) scenarios where non-metric data is present, i.e. data which cannot be compared by a metric, or "closeness", measure.

8.4 Questions

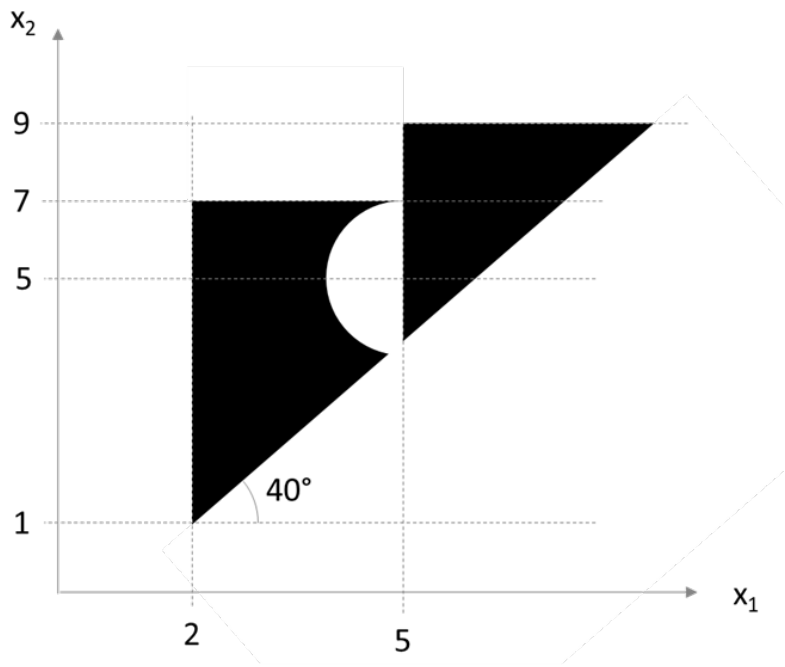
1. I have a set of work tools, and data recorded about each one. This includes the following parameters: mass (kg), length (cm), type (spanner, hammer etc.) and age (years). I have had these valued, and I wish to estimate for a general tool whether the value will be more or less than £30.

   (a) Explain why a random forest/decision tree approach may be most suitable for this task.

   (b) I now only consider a subset, consisting of just my collection of screwdrivers. Would a random forest still necessarily be the most appropriate?

2. I define a decision tree based on some training data. At one of the nodes, I have 10 points in my dataset which reach that node. There is 1 from category 1, 6 from category 2 and 3 from category 3. What is the Gini impurity at that node?

3. Sketch out a decision tree corresponding to the decision function below, where white corresponds to class A and black to class B, making clear at each node what the equations for the corresponding inequalities are. You should make your tree as concise as possible.


x2
9
7
5
40°
1
x1
2
5

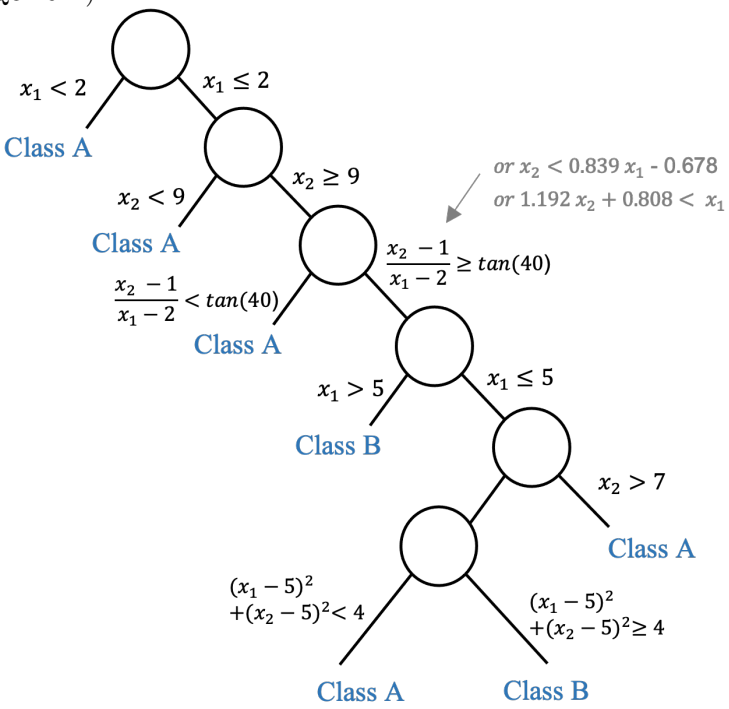
8.4.1 Solutions

1. (Q7 2020)
   a) Type is a non-metric parameter, i.e., has no 'closeness' associated with it. Decision tree-type methods work with non-metric problems. b) Other methods may be more suitable. The type parameter has effectively been moved so the data is metric.

2. (Q9 2020)

$$\begin{aligned} impurity &= 1 - P_1^2 - P_2^2 - P_3^2 \\ &= 1 - 0.1^2 - 0.6^2 - 0.3^2 \\ &= 0.54 \end{aligned}$$

3. (Q3 2021)



Chapter 9

Unsupervised learning

We briefly introduced unsupervised learning back in Sect. 1.4. Unsupervised learning is the identification of patterns within unlabelled data, which contrasts to the previous cases throughout the majority of this course, where we have data labels and we wish to use the training data to provide predictions of the labels for future cases. Mathematically, previously we had the parameters for all points $\boldsymbol{X}$ and a corresponding set of outputs, $\boldsymbol{Y}$, now we just have $\boldsymbol{X}$ and wish to identify patterns in this. Unsupervised learning attempts to pull out information about the relationships between data (similarities and differences). C9A

This can be useful for a number of reasons. In many cases it can help with labelling large datasets; this can be very expensive (think speech patterns for example), but by pulling out cases which are very similar we may be able to simplify the labelling process. In some cases there may be variations in the pattern characteristics over time (for example, seasonal variations), and unsupervised learning can identify these trends and improve the performance. Feature extraction, discussed in Sect. 11.5, can be achieved through unsupervised learning, where the key common features in a dataset are recognised and exploited, and this can also subsequently be used in data compression. And there are many more examples where unsupervised learning is very valuable.

This chapter will discuss two of the main approaches for unsupervised learning, clustering and principal component analysis.

9.1 Clustering

9.1.1 K-means clustering

C9B

Clustering splits the data into discrete separable groups - clusters. To illustrate this, we will consider K-means clustering, given by Alg. 9.1, and illustrated in Fig. 9.1.

The K-means algorithm is iterated until it converges; the solution will be one where each point is allocated to its nearest mean, and the mean position is given by the mean of all the points allocated to it. This converged situation, in an ideal scenario, would have each cluster allocated to a different mean. C9C

There are some questions which arise from this. Firstly, we must consider the starting point for each mean. Note that the algorithm does not necessarily achieve global convergence, i.e. will become stuck in local minima. Therefore the result will depend on the starting points. One option

Algorithm 9.1 K-means algorithm

1. Define K mean points in the parametric space at arbitrary positions
2. Allocate each training point to the nearest of the means just defined
3. Adjust the mean points to correspond to the mean of the points allocated to it
4. Repeat the process from 2.

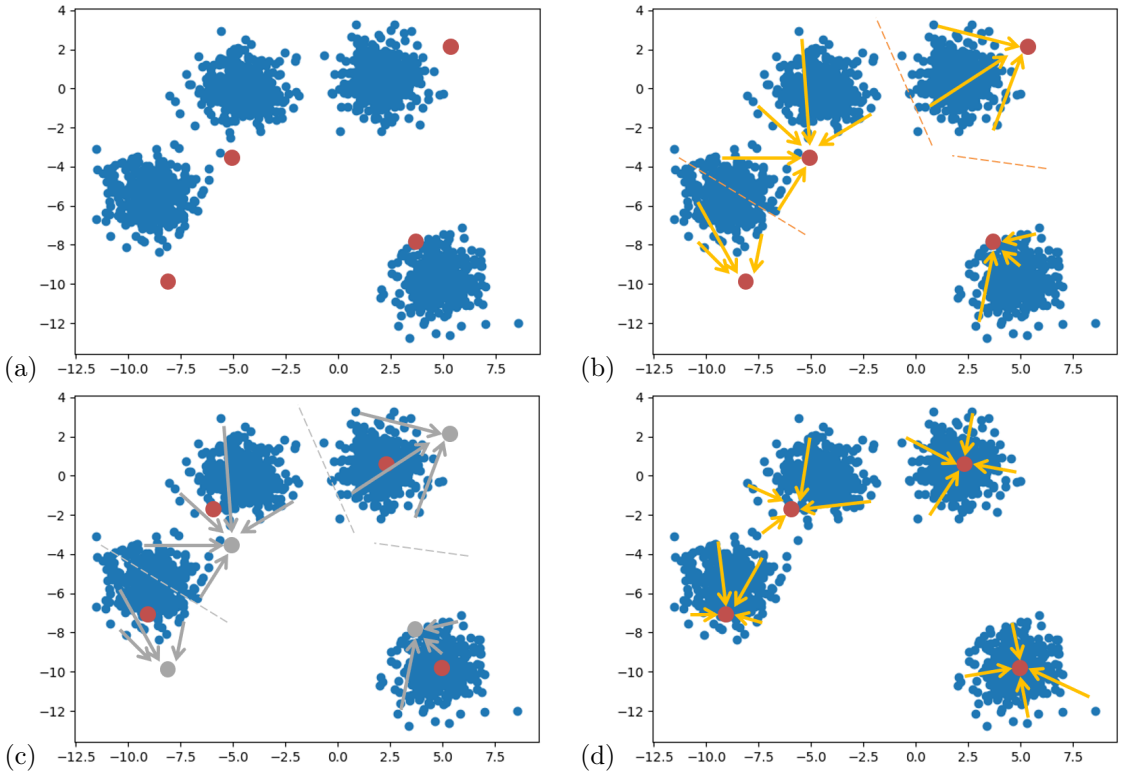


Figure 9.1: Illustration of the K-means algorithm. Means are given in red and training points in blue. In (a) the initial positions of the means are defined. (b) then shows each training point being allocated to the nearest mean. (c) then uses the allocated points to calculate a new set of means. (d) then shows the process repeating again, as the points are reallocated to each mean depending on their new positions.

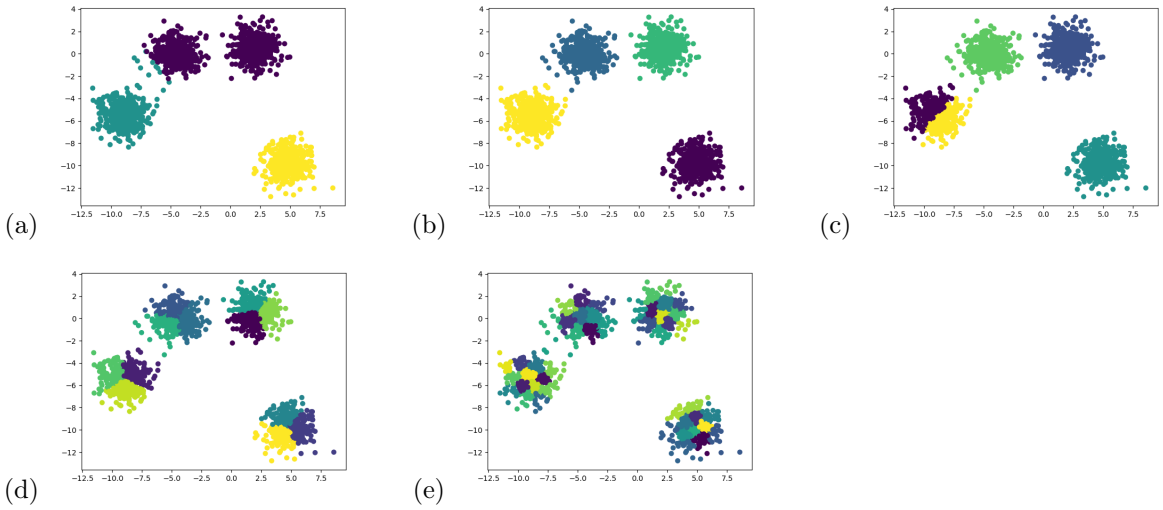


Figure 9.2: K-means clustering, with different numbers of means. (a) 3 means, (b) 4 means, (c) 5 means, (d) 12 means and (e) 48 means.

is to randomly distribute these, which is quick and straightforward to implement, but may suffer from the local minima issue. Another approach is to run the algorithm several times with different starting points, and assess which one produces the best result (typically via some measure such as the total distance from the mean); this may take longer to run, but the algorithm is fast so it should not be problematic. Scikit-learn uses an approach called K-means++ which is described as a "careful seeding" approach. Scikit-learn also automatically runs the algorithm multiple times and takes the optimal solution.

A second question is how many means should be defined. This is decided in advance. Figure 9.2 presents clustering for a simple case with four clusters, with different numbers of means. We can see that the algorithm still manages to split the points into K separate clusters, but there is no guarantee that this will be consistent with the patterns of the data. Again, an optimisation routine could be used where different numbers of means could be tested and the optimum taken, but another approach may be more suitable in this case. In some cases we may already have experience of the problem being solved, however, in practice, there are many scenarios where we have other requirements to split the data into a particular number of parts, even if this may not be the optimum.

A final point of note is that the numbering of the clusters is arbitrary. The nature of unsupervised learning means that there is no reference against which to compare, which would enable such numbers to be established. Therefore, the assigned numbers to each point are simply to distinguish different clusters. The assigned numbers may also be different each time the algorithm is run.

In summary, K-means is a lightweight, conceptually straightforward, and fast algorithm. In terms of applications within engineering, we could potentially use it for analysis of component failures based on manufacturing parameters (time of day, amount of training of operatives etc.), or population location analysis to establish the optimal position of transport hubs. Have a think yourself about whether you can come up with any other examples[1].

9.1.2 Other clustering techniques

C9D

There is a huge range of clustering methods out there; we will not evaluate all the methods here. These all have advantages and disadvantages associated with them. https://scikit-learn.org/stable/modules/clustering.html provides quite a nice graphical example of some of these implemented for a number of challenging scenarios - you can see that no one routine always performs best. Running through a few examples:

[1] I would be interested to hear about these!

- Mini batch K-means
  - Same as K-means, but optimising the means based on a small random sample of the training values
  - Speed benefit, and likely minimal accuracy loss for large datasets
- Affinity propagation
  - Compare each point with all other points then link them together accordingly
  - Do not need to specify number of clusters
  - Slower and more memory intensive
- Hierarchical clustering
  - Top-down subdivision of points, a tree with relationships between all the points, with the root being all the points and leaves each being an individual point
  - This is powerful when we expect to have different levels of relationship between the points

Overall, the K-means algorithm gives a good idea of the concepts behind clustering – that the purpose is to link data points together – we will not discuss these routines in any further depth.

9.2 Principal component analysis

C9E

We have seen how clustering can identify particular clusters within a dataset. Principal component analysis (PCA) can also be used to identify underlying behaviour within an unlabelled dataset.

PCA is a method which can be used to identify common trends between the data. For a dataset with n input parameters, PCA will produce n vectors which can be used to describe the dataset. Each of the vectors will in turn be less significant than the previous one, i.e. the first one will capture the largest variations present in the data. We will begin with an outline of a key mathematical concept upon which PCA relies.

9.2.1 Orthogonality

C9F

You should be familiar with the concept of orthogonality, particularly in a physical sense. Two vectors in 2D or 3D space are orthogonal to each other if they are at 90 degrees. This can be expressed mathematically by first considering the dot product

$$\boldsymbol{a}.\boldsymbol{b} = |\boldsymbol{a}|\,|\boldsymbol{b}|\cos\theta \tag{9.1}$$

where $\boldsymbol{a}$ and $\boldsymbol{b}$ are vectors and θ is the angle between them. If the two vectors are orthogonal, $\theta = \pi/2$, so $\cos\theta = 0$ so

$$\boldsymbol{a}.\boldsymbol{b} = 0. \tag{9.2}$$

Although this is not any form of rigorous mathematical proof, we can generalise the concept to higher dimensions, where a dot product is the sum of the individual terms of each vector multiplied together

$$\boldsymbol{a}.\boldsymbol{b} = \sum_{i=1}^{n} a_i b_i. \tag{9.3}$$

Mathematically, orthogonality implies a lack of "commonality" between two vectors. In physical space this is conceptually easy to visualise, but in higher dimensions, if two vectors are orthogonal then overall this indicates that we are unable to express any part of one vector in terms of the other.

We can now consider that we have a set of orthogonal vectors, $\boldsymbol{w}_i$, where each vector is orthogonal to all others in the set, i.e. $\boldsymbol{w}_i.\boldsymbol{w}_j = 0$ for all $i \neq j$. We will also place a further restriction that each vector should have unit "length", so $\boldsymbol{w}_i.\boldsymbol{w}_i = 1$. In 2D space, we can have two vectors

which are orthogonal to each other – you should be able to physically visualise this – if you have two orthogonal vectors in 2D space, it is not possible to add a new one which is orthogonal to the previous two. In 3D dimensions this extends to 3 orthogonal vectors which can exist, and in n dimensions, n orthogonal vectors.

These sets of vectors have some convenient properties. As a simple example, you should all be familiar with $\boldsymbol{i}$, $\boldsymbol{j}$, $\boldsymbol{k}$ unit vectors for Cartesian problems; this is a set of orthogonal vectors, each one aligned along an axis. Any position in space can be expressed as a unique linear combination of these vectors. Now consider a 2D situation, where we have two orthogonal unit vectors $\boldsymbol{i}'$,$\boldsymbol{j}'$ which, by the definitions given above, can always be expressed as the $\boldsymbol{i}$, $\boldsymbol{j}$ vectors rotated by an arbitrary angle. It is clear to see that we can similarly define any point in space with a linear combination of these two vectors. C9G

It is when we come to the question of how to express a point as a sum of weighted vectors that the real power of the orthogonal set of vectors becomes clear. Let's say we want to find the weights, t_i, in order to describe a point $\boldsymbol{x}$

$$\boldsymbol{x} = \sum_{i=1}^{n} t_i \boldsymbol{w}_i. \tag{9.4}$$

We start by doing a dot product with a second vector $\boldsymbol{w}_j$ from the same orthogonal set:

$$\boldsymbol{w}_j.\boldsymbol{x} = \sum_{i=1}^{n} t_i \boldsymbol{w}_j.\boldsymbol{w}_i$$

and recognise that $\boldsymbol{w}_i.\boldsymbol{w}_j = 1$ if $i = j$ and 0 otherwise[2], to give

$$\boldsymbol{w}_j.\boldsymbol{x} = t_j. \tag{9.5}$$

If we were instead to consider that the vectors were not orthogonal (and, less importantly, unit magnitude), then we would see that we need to solve a set of simultaneous equations, which will become increasingly difficult in problems with more dimensions.

9.2.2 Principal components

C9H

Now let us say that we have our dataset $\boldsymbol{X}$, parameterised in n dimensions. We will at this point make the assumption that the mean of each parameter in $\boldsymbol{X}$ is zero, and continue this throughout our discussion in this chapter. In practice this means that we should subtract off the mean from a dataset when performing principal component analysis. In principal component analysis we generate a set of principal components, which are a set of orthogonal vectors, which capture the main directions of variance of the data. We aim to describe each point in our dataset, $\boldsymbol{x}_i$, in terms of the orthogonal vectors $\boldsymbol{w}_j$ and weighting terms t_{ji}, and we wish to do this such that the variance t_{1i} is as large as possible. Maximising the variance will make $\boldsymbol{w}_1$ capture as much as possible of the underlying behaviour. We will discuss the other vectors later, but these will in turn maximise the residual variance not captured by the preceding vectors. To maximise the variance, we want

$$\begin{aligned}
\operatorname{argmax} \sum_{i=1}^{m} (t_{1i})^2 &= \operatorname{argmax} \sum_{i=1}^{m} (\boldsymbol{x}_i.\boldsymbol{w}_1)^2 && (9.6)\\
&= \operatorname{argmax} \|\boldsymbol{X}\boldsymbol{w}_1\|^2 && (9.7)\\
&= \operatorname{argmax} \left((\boldsymbol{X}\boldsymbol{w}_1)^t \boldsymbol{X}\boldsymbol{w}_1 \right) && (9.8)\\
&= \operatorname{argmax} \left(\boldsymbol{w}_1^t \boldsymbol{X}^t \boldsymbol{X}\boldsymbol{w}_1 \right) && (9.9)\\
&= \operatorname{argmax} \left(\boldsymbol{w}_1^t \boldsymbol{C}\boldsymbol{w}_1 \right) && (9.10)
\end{aligned}$$

[2] As an aside, this is a form of a delta function (the Kronecker delta), which you may have come across elsewhere. In fact we did actually see it in eq. (6.23) when we were discussing neural networks in Chapter 6. Convolving (i.e. multiplying by then integrating) with a continuous delta function (Dirac delta function) shifts the underlying signal. Here we multiply by the discrete delta function and sum, so we effectively shift the vector t_i to t_j. This is really just a curiosity if you are interested, and not anything for the exam (which we all know is the really important thing).

Algorithm 9.2 Principal component analysis algorithm

1. Acquire data and assemble into matrix $\boldsymbol{X}$
2. Estimate covariance matrix $\boldsymbol{C} = (\boldsymbol{X} - \boldsymbol{\mu})^t(\boldsymbol{X} - \boldsymbol{\mu})/m$
3. Get eigenvalue-eigenvector pairs for $\boldsymbol{C}$ (typically eigenvalues will be normalised and corresponding eigenvectors will be in order).

where argmax means that we select the argument which gives the maximum of the given function, and we have used the definition of the covariance matrix $\boldsymbol{C} = (\boldsymbol{X}-\boldsymbol{\mu})^t(\boldsymbol{X}-\boldsymbol{\mu})/m$, from Sect. 1.7.1, but taking zero mean[3]. We wish to get the maximum of the term $\boldsymbol{w}_1^t\boldsymbol{C}\boldsymbol{w}_1$. To achieve this we let C9I

$$\boldsymbol{a} = \boldsymbol{C}\boldsymbol{w}_1 \tag{9.11}$$

so

$$\boldsymbol{w}_1^t\boldsymbol{C}\boldsymbol{w}_1 = \boldsymbol{w}_1^t\boldsymbol{a}. \tag{9.12}$$

Since $|\boldsymbol{w}_1| = 1$, the only way we can achieve this is by ensuring that $\boldsymbol{w}_1$ and $\boldsymbol{a}$ are in the same direction, i.e.

$$\boldsymbol{a} = \lambda\boldsymbol{w}_1 \tag{9.13}$$

where λ is a scalar. Therefore

$$\boldsymbol{C}\boldsymbol{w}_1 = \lambda\boldsymbol{w}_1 \tag{9.14}$$

i.e. $\boldsymbol{w}_1$ is an eigenvector of $\boldsymbol{C}$, and for the largest result (i.e. the largest variance, which we are targeting) we should take the largest eigenvalue, which corresponds to λ. The variance can be calculated as

$$\begin{aligned} \boldsymbol{w}_1^t\boldsymbol{C}\boldsymbol{w}_1 &= \boldsymbol{w}_1^t(\lambda\boldsymbol{w}_1) && (9.15)\\ &= \lambda\boldsymbol{w}_1^t\boldsymbol{w}_1 && (9.16)\\ &= \lambda. && (9.17) \end{aligned}$$

which is just the eigenvalue itself. C9J

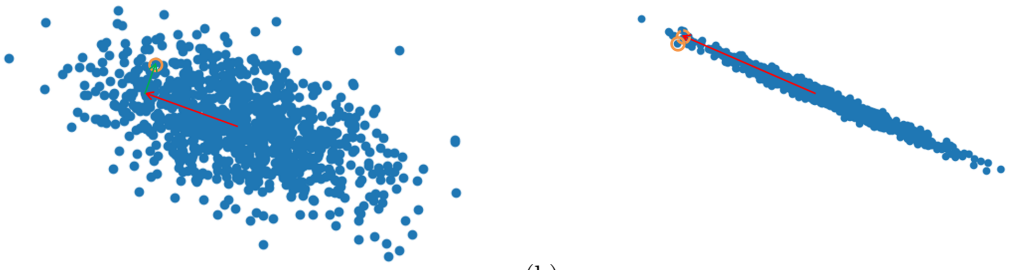
For the remaining terms, the other principal components can be shown to correspond to the other eigenvalues, however, we won't go through this. The overall approach for principal component analysis can be described as shown in Alg. 9.2

9.2.3 Use of principal component analysis

We have covered the maths behind PCA, without providing huge insight into what principal components actually mean. By capturing the directions with the most variation, principal components provide a description of the underlying data. The first principal component will indicate the direction of strongest correlation in parametric space, and the second the next highest, and so on. C9K

Figure 9.3(a) shows a simple 2D example. The two principal components are shown in red and green - red is the first and green is the second. Red is aligned to the distribution, while green is orthogonal to this. When we select a single data point, the largest weighting (on average) will be for the first principal component direction, and a smaller value for the second; the point highlighted illustrates this. Figure 9.3(b) gives an example where the second principal component (given by the eigenvalue, remember) is very small. Here the data is dominated by the first principal component, and in terms of describing the data we can neglect the very small second principal component. While this looks like a fairly uncommon scenario for a 2D dataset, in practice, the final eigenvalues are often negligibly small when we move to higher dimensional space. This has applications in data compression, since we can remove some redundant information, but importantly for machine learning it is reducing the dimensionality of the data, and also identifying the most significant features, which makes it very well suited as a pre-processing tool for other ML algorithms. C9L

[3]Note that if just finding directions then the scaling by m is unimportant. It only has relevance if needing to use the variance values directly.



(a) (b)

Figure 9.3: Examples of principal components in 2D. (a) a point in the dataset can be expressed as a combination of the two orthogonal principal components. (b) shows a scenario where the second principal component is very small.

At this point we can return to a comparison with supervised machine learning. Let us consider a problem with a single input parameter and a single output, both continuous; as discussed in Chapt. 3, we can perform a linear regression to identify a relationship between the input and output. By contrast we could put the input and output parameters together into a single two-parameter dataset $\boldsymbol{X}$, and have no output, which would be an unsupervised learning problem. In this case, we can perform PCA to identify a linear correlation between the two parameters. The first principal component will give a measure of the direction of largest variance, which is analogous to the gradient of the best fit linear regression line. In this case, the distinction between supervised and unsupervised learning is largely one of context - is the problem framed as us wanting to identify a specific output given an input (supervised) or are we simply trying to identify patterns in the data (unsupervised)?

9.2.4 Similarities to principal stress and strain

You will have studied principal stresses and strains earlier in your degree (in ME2 Stress Analysis), where a set of stresses are applied to an object and can be distilled down into a set of three normal stresses on three orthogonal planes. These stresses can be determined by an eigenvalue decomposition, in the same way that we have done here with principal component analysis. Solving eigenvalue problems is very common for finding principal directions in complex matrix equations.

9.3 Summary

This chapter has considered unsupervised learning - specifically, clustering and principal component analysis (PCA). Clustering involves identifying groups of similar data points. We studied the K-means algorithm, which iteratively allocates points to their nearest mean and then updates the mean until convergence, which is fast and lightweight. We also looked at PCA, to identify the links between different parameters in an unlabelled dataset. C9M

One of the big challenges of unsupervised learning is that there is no defined "right" answer. Whereas for supervised learning there is a clear cost metric that can be defined to identify how well the algorithm is performing, based on how well the output fits, for supervised learning this is not the case.

9.4 Questions

1. Express the point (1, 3) as a weighted sum of the non-orthogonal vectors (1,0), (0.2, -0.8). Now change the first vector such that it is orthogonal to the second and calculate the new weighting coefficients.

2. A colleague comes to you and tells you that they have identified all the principal components of their dataset as (0.5, 0.5, 0, 0.4) (0.2, 0.1, -0.2, 0.2) and (-0.3, 0, -0.5, -0.1). Give all the reasons why these cannot be correct, including mathematical calculations where necessary to justify these.

3. I have intermittent failures in a new design of engine and have recorded the age, temperature and RPM at failure, as well as mean temperature and RPM through the life. In each of the following cases indicate whether unsupervised or supervised learning is most suitable:

   (a) I wish to identify whether there are any patterns which could indicate the cause of the failure.

   (b) I wish to estimate the age at failure based on the other parameters.

   (c) Taking a new engine and expected usage measures (temperature and RPM) I wish to predict whether or not it will fail within the warrantee period.

4. You are given the following points: (1, 1), (2, 2), (2, 3).

   (a) Treat this as a supervised learning problem and perform a linear regression on these points, outputting the equation of the best-fit line.

   (b) Now treating it as an unsupervised problem, perform Principal Component Analysis (PCA) and write out the equation of the line passing through the data in the direction of the first principal component.

   (c) Explain, using a diagram where appropriate, why the gradient of the line is different in the two cases. Also explain how this relates to regression being a supervised method and PCA being unsupervised.

5. I wish to undertake a k-means clustering approach. At one iteration I have means at (3, 5) and (7, 2).

   (a) Sketch out a single node neural network which would classify points as -1 or +1 depending on which respective mean they should be allocated to at this iteration, -1 indicating the point being allocated to (3, 5) and +1 for allocation to (7, 2). Calculate all the weighting terms in the network. Make sure you clearly state any activation functions used.

   (b) Explain why additional layers and nodes are not needed for this problem.

9.4.1 Solutions

1. Weightings will be 1.75 and -3.75 for the respective vectors - can find these through simultaneous equations. The second part has multiple answers depending on how you change the first vector, but note that you can do through orthogonal projection, but you do need to account for the vectors not being unit vectors.

2. (Q1 2020) Should be 4 components for 4 parameters.
   Not orthogonal: $0.5 \cdot 0.2 + 0.5 \cdot 0.1 + 0 \cdot (-0.2) + 0.2 \cdot 0.4 = 0.1 + 0.05 + 0.08 = 0.23 \neq 0$
   $0.5^2 + 0.5^2 + 0.4^2 = 0.25 + 0.25 + 0.16 = 0.66 \neq 1$, so not a unit vector.

3. (Q8 2020)
   a) Unsupervised
   b) Supervised
   c) Supervised

4. (Q2 2021)a) Linear regression – put into standard simultaneous equations.
   m = 3
   sum x = 5
   sum $x^2 = 9$
   sum y = 6
   sum $y \cdot x = 1 + 4 + 6 = 11$
   So equations are:
   $3\beta_1 + 5\beta_2 = 6 \quad (1) 5\beta_1 + 9\beta_2 = 11 \quad (2)$
   $\frac{(1)}{3} \cdot 5: \quad 5\beta_1 + \frac{25}{3}\beta_2 = 10$
   Now subtract (2): $\left(\frac{25}{3} - 9\right)\beta_2 = 1 \quad \Rightarrow \beta_2 = \frac{3}{2} = 1.5, \quad \beta_1 = 2 - \frac{5}{3}\beta_2 = -0.5$
   So equation is $y = 1.5x - 0.5$
   b) PCA:
   Remove mean initially:

$$\bar{x} = \frac{5}{3}$$
$$\bar{y} = 2$$
$$\begin{pmatrix} 2 & 3 \\ 2 & 2 \\ 1 & 1 \end{pmatrix} - \boldsymbol{\mu} = \begin{pmatrix} \frac{1}{3} & 1 \\ \frac{1}{3} & 0 \\ -\frac{2}{3} & -1 \end{pmatrix}$$

   Calculate covariance matrix:

$$C = \begin{pmatrix} \frac{2}{3} & 1 \\ 1 & 2 \end{pmatrix}$$

   Get eigenvectors – large one is:

$$v = \begin{pmatrix} -2 + \sqrt{13} \\ 3 \end{pmatrix}$$

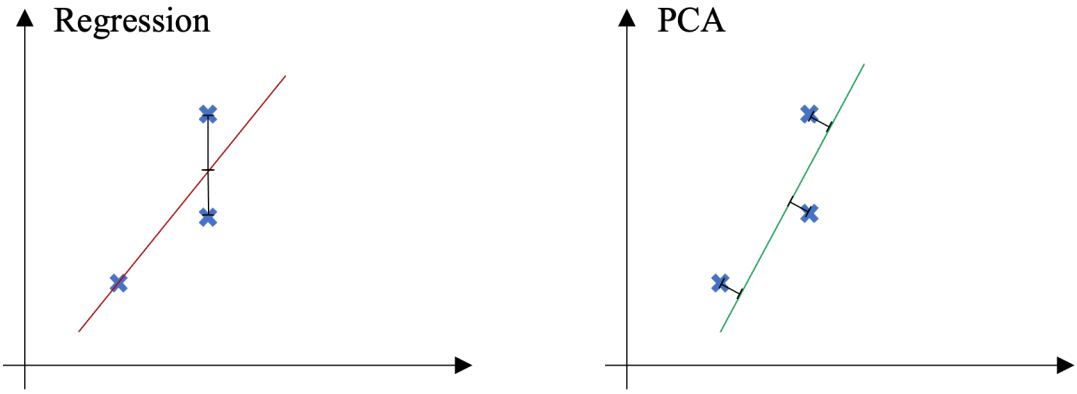
   Put into an equation – subtract off mean terms from x and y and make gradient between them along the eigenvector:

$$y - 2 = \left(x - \frac{5}{3}\right) \frac{3}{-2 + \sqrt{13}}$$

   So:

$$y = 1.87x - 1.12$$

   c)



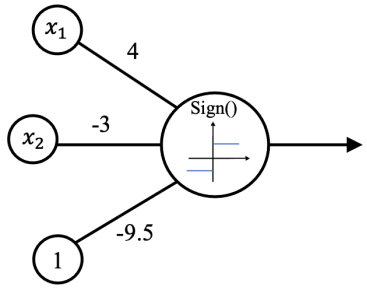
Regression minimises the square of the y distances. Principal component analysis instead minimises the square of the perpendicular distance between the line and the points. PCA makes no distinction between the two parameters, x and y – both are fitted equally - which fits with it being an unsupervised method. However, regression minimises the error in the y parameter, the output under a supervised method, which is to be fitted.

5. (Q5 2021) a) Direction must be between the two points, so:

$$\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 7-3 \\ 2-5 \end{pmatrix} = \begin{pmatrix} 4 \\ -3 \end{pmatrix}$$

Bias value must be defined such that get zero out at the centre (5, 3.5):

$$\begin{pmatrix} 4 \\ -3 \end{pmatrix} \cdot \begin{pmatrix} 5 \\ 3.5 \end{pmatrix} + w_0 = 0$$
$$20 - 10.5 + w_0 = 0$$
$$w_0 = -9.5$$



b) The line equidistant from the two points under the L2 norm is straight, so a linear separation is sufficient. The addition of more nodes and/or layers is unnecessary since the single node can fully capture this linear problem.

Chapter 10

Machine learning in the world

Machine learning, artificial intelligence and related technologies achieve a very high profile within the media, which can largely be attributed to the apparently limitless opportunities which can arise from its application. This course has covered the techniques behind machine learning from a practical viewpoint; this chapter aims to provide some of the background context to machine learning. Some of these concepts are important when it comes to the applications of the techniques because they highlight fundamental limitations in capability, while others are important because they highlight, for example, the privacy issues which can arise from the use of machine learning. The chapter is split into three broad sections, considering dataset limitations, algorithmic issues and societal considerations around the use of machine learning.

10.1 Dataset limitations

C10A

It is an inherent limitation of machine learning algorithms that they can only classify or estimate values in regions where they have been given (sufficient) training data. The algorithms cannot predict outside the training dataset; they are not inherently "smart" - as an example, training on a current set of customers would not necessarily work on new customers (although we may be able to make some predictions). In a similar way, a ML routine may be able to work out momentum conservation by observing lots of fluid mechanics problems, but would not be able to apply this to solid mechanics.

The algorithms can also pick up biases within the data, for example, an algorithm which is trained on data which misses certain types of defects will be very likely to continue to miss these sorts of defects when applied. There are also societal examples of this, where certain demographics are not included in a training dataset, resulting in an trained algorithm which is inherently discriminatory against particular groups.

10.1.1 Outliers

C10B

Through the course we have usually associated outliers with being unrepresentative of the true behaviour and likely to cause overfitting, so we would generally put in place processes to avoid these. By definition, outliers are rare - they do not occur much during training, validation or actual use, so it can be argued that they are unimportant and can be ignored. The problem comes when such outliers have a disproportionally large impact associated with them.

The occurance of such outliers actually occurs much more frequently than is recognised, for a number of reasons. One is feedback mechanisms. Academics are more likely to cite a paper which is already highly cited, because this is a measure of how "good" the paper is[1]. Therefore such a paper will be cited more and more. This means that we get a disproportionate number of outliers compared to what may be predicted with the normal distribution. The problem is quite pervasive, arising across engineering. Other examples can be seen in the financial sector where

[1] There are a LOT of problems with this, but here we are just considering it as an example rather than discussing the merits.

extreme financial events – crashes – occur far more regularly (10-20 years) than the majority of financial models actually predict.

This manifests itself as "fat tails" at the extremities of the normal distribution curve. A solution is to utilise a more suitable distribution, such as STUDENT'S T-DISTRIBUTION. Above the normal distribution parameters of mean(μ) and standard deviation (σ), the distribution has a third parameter, ν, which defines the thickness of the tails, and provides an opportunity to fit these distributions; note that $\nu = \infty$ corresponds to the standard normal distribution. In theory, it would be possible to estimate this additional parameter from the training data, however, the issue is that this is completely dependent on the specific outliers which exist in that dataset, and because, by definition, they are rare, it is essentially impossible to produce any accurate distribution which captures outliers reliably. The general issue is that by fitting a distribution, extrapolation is effectively being performed from the highly sampled points (around the mean) out towards the very sparse outliers, and there is no guarantee that this will be correct. As highlighted, this is an issue when the outliers have high impact. The book "The Black Swan" by Nassim Nicholas Taleb discusses extensively the issue of outliers [1] and many of the points I have discussed here.

This can have big impact in many aspects of engineering where safety is critical. Almost by definition, failures are rare – we go to huge lengths to avoid these, which is generally successful. What we are saying here is that it is impossible to estimate exactly *how* rare because we just cannot get enough data in that region of the probability distribution. And you should challenge anyone who tells you otherwise!

10.1.2 Unknown unknowns

C10C

Related to this, many events are without precedent, and by definition, unlearnable. A good example within engineering is the de Havilland Comet, which was the worlds first commercial jet airliner. This was a pioneering plane, and an early example of having a pressurised cabin. Unfortunately, materials engineering was not at the same level, and several catastrophic accidents occurred, where fatigue cracks at the square windows joined up causing the entire aircraft to break up. This was previously completely unknown, and hence unpredictable[2]. You can find other examples of unprecedented scenarios throughout life. One that Taleb discusses in his book [1] is the concept of a turkey, which can reliably predict that every day it will go about its life as normal, based on its prior experience that this is what has happened throughout its life. Ultimately, however, the turkey will be surprised about its fate, while the butcher will not.

You should recognise that your model will only work around the mean, outliers will occur more often than you expect, and you should not feel that you can predict these. Ideally you should design your model such that it can identify when it is being asked to predict beyond its capabilities and report this.

10.1.3 Other data-related issues

C10D

In many scenarios, we have far more data in one class than the other, for example, we may be wishing to classify whether defects are present or not in an image. We have far more cases where defects are not present than when they are[3]. This is known as DATA IMBALANCE and generally reduces the performance of the models. Training is more difficult and less likely to ultimately produce accurate results.

As you are aware at this point in the course, having sufficient data is vital to achieving good results with machine learning. This raises the question of how much data is enough, and this is very problem dependent. Clearly validation methods can help to confirm when you have sufficient data, but it is very difficult to say beforehand how much is needed. In practice, it depends on the underlying complexity of the problem. A simple linearly separable problem will need very little data to be able to define a good separation boudary, but something with a very complex boundary will need many points to identify it, regardless of the model used. This relates to the CURSE OF DIMENSIONALITY too: to fully sample a space in N+1 dimensions, we need many more points

[2] Although inevitably it may look obvious to some people from a hindsight perspective.

[3] The vast majority of engineering companies try to avoid making components with defects in them, and most are successful at this.

than for an N-dimensional problem. It may not be necessary to have as dense sampling with more dimensions, but this rarely does much to mitigate the huge increase which would otherwise be required.

At the start of Chapter 1 I also highlighted one source of the more amusing machine learning issues, solving the wrong problem. ML algorithms are not really "smart", but instead will naively extract what they can find from the data available. This could be whether a tank is US or Russian, or it could be that sheep can only be detected when they are on a green background. On the whole it is relatively easy for a human to spot these errors, but traditional metrics like mean squared error may not!

10.2 Algorithm issues

10.2.1 Black boxes C10E

Many ML algorithms suffer from having limited transparency in how they work; this is particularly the case for neural networks. This is an inevitable issue for models which aim to capture ever-increasing complexity. It is difficult to provide full validation and justification for the use of these models, because interpreting them and providing reasoning is a huge challenge. There is a movement at the moment to provide more information about what the algorithms are doing beneath the surface, such as tools which highlight which pixels a ML algorithm may be sensitive to when making decisions in a computer vision scenario, for example.

10.2.2 Lack of physical appreciation C10F

Machine learning algorithms can be used to model the physical world. However, they blindly utilise data and hence have no appreciation for physical limitations. As engineers we must understand this and develop solutions which respect and exploit physical knowledge. The algorithms also provide limited insight. In an example we considered in Sect. 3.1.3, we discussed that shear stress in a fluid will be a function of shear strain rate as well as temperature, and ML can identify the trends. But it cannot recognise that there is a quantity of viscosity which is itself a function of temperature, and that this links the shear stress and strain rate.

10.2.3 No free lunch C10G

There is no algorithm which will outperform all other algorithms across all problems. Another way of describing a similar concept is that, given two algorithms, if you average their performance across all possible scenarios, you will get the same result.

So why do we have so many different algorithms? This is because the routines are tuned to specific cases which meet our needs. Inevitably this algorithm will be worse at other scenarios we've not considered. Related to this is the point that the selection of the algorithm is actually part of the training process itself - as you select a model which you know performs well for a particular task, you are already doing some fitting to the problem.

10.2.4 Computational resources C10H

For the largest problems, you will have heavy computational requirements, both when training, but sometimes also when being applied in practice. This is very expensive financially, with significant initial hardware and running costs, and is also expensive for the environment. It can also take significant time to train these algorithms. The types of calculations are typically very parallel (multi-layered neural nets are essentially a lot of matrix multiplications) so hardware well suited to this can be used; general purpose GPUs (graphics cards) are often utilised.

10.3 Societal considerations

Frequently, when machine learning appears in the news, the impact on society is what is being considered. Here we discuss two key areas – privacy and replacement of jobs by automation – as

well as considering how humans can work with machine learning algorithms.

10.3.1 Privacy C10I

ML algorithms have the ability to consume vast quantities of data and identify patterns and links. For example, if many people share their locations with a company[4], this company could correlate these locations and identify who their friends are, where they work, who they are having an affair with.

Other aspects involve facial recognition with ML, which is being increasingly used by various governments to either oppress citizens or maintain law and order, depending on your personal opinion. Additionally, ML opens up new opportunities for forgery, which may not have been possible otherwise. For example, by evaluating enough of someone's handwriting, it should be possible to artificially generate something which is indistinguishable from the real thing. This also relates to deepfakes, which you may have seen in the news, where machine learning is used to replace a person in an image or video with someone else's image, which is can be problematic given our reliance on photographs and videos as sources of truth. Automatically generated writing through ChatGPT and similar tools is also becoming increasingly common, which has implications across many fields.

10.3.2 Replacement of jobs by automation C10J

ML presents some huge opportunities for automation. There are many tasks which could be done by machines and technology in existence right now. But the uptake is heavily limited by political and societal issues. These considerations are typically far more significant than any technical limitations. ML will never replace all jobs. Inevitably, the rise of automation can be seen to fuel a rise in the division between rich and poor, and has done so through history as technology has advanced.

One interesting practical example of the political aspects involves the railways. We talk a lot about the potential for driverless cars, which have to deal with a huge range of unexpected occurrences out on the roads. This contrasts with the railway network, which is a closed, controlled system, where essentially we have to control just a single parameter, the train speed. This should be easy! However, there are relatively few driverless train systems.

There is one obvious exception which comes to mind, the DLR[5], which actually gives a clue about the politics. The DLR is driverless, and was initially opened in the 1980s before being expanded over the subsequent decades. It is the recent opening of the DLR which provides a clue about why the railway system is not automated: it is easy to build a completely new automated system, but much more difficult to switch an existing network to driverless, not because of technical aspects but because this would involve redundancies and fights with unions, in schemes, which because of their great expense, must be government-supported. We can imagine that these considerations will continue to limit the range of applications of ML in future.

10.3.3 Human augmentation C10K

One area to explore which may offer more opportunity is to evaluate how ML can augment human abilities, rather than replacing them. Computers have been doing this for years, actually – just think about calculators! Some examples of this from the non-destructive evaluation area include eliminating 90% of indications which are definitely not defects, while leaving the operator to focus on the remaining 10%. Also, some systems require two operators to independently perform the inspection, for safety critical applications. Research is being undertaken to utilise ML as the second operator in this scenario.

[4]Note here that this is not just explicitly sharing GPS with Google, but your network provider will be able to get a rough idea of where you are by triangulation, and a similar thing happens if you have wifi on in a shopping centre. Westfield are watching you.

[5]I assume most students know the DLR (Docklands Light Railway) and have been on it. If not, I suggest you head to east London and give it a go. It's weirdly fun. Canary Wharf is completely different from the rest of London, so worth visiting, and Greenwich is great with the Cutty Sark and the observatory. Worth reading the book "Longitudinal" by Dava Sobel before you go to the observatory. But I'm going to stop going on now...

10.3.4 Comparisons with intelligence

There are some philosophical discussions which can be had regarding whether machine learning can be considered intelligence[6]. Could it be the case that if a machine learning algorithm had been given access to all the information to which we have had access to throughout our entire lives, that it would be able to produce matching intelligence? What then does intelligence itself mean? Is a sufficiently good imitation of intelligence essentially the same as intelligence itself? These concepts have been discussed quite a lot - you can look up "Turing tests" and the "Chinese room argument", if you are interested. It is beyond the scope of this course to go any further on this topic here[7].

10.4 Summary

Machine learning is powerful, but has limitations which need to be recognised for successful implementation.

[6] I appreciate that I have used the word "smart" at various points through the notes to indicate the limitations of algorithms...

[7] You may wish to do a philosophy degree if you wish to learn more. A PhD is almost as good though... we are always recruiting – contact p.huthwaite@imperial.ac.uk if you are interested in having a chat about this!

Chapter 11

General concepts

In this machine learning course we cover several concepts and tools which are common to many (if not all) the methods. This chapter is intended as a reference for these methods.

11.1 Bias and variance

C11A

Fitting data will result in two main forms of error: bias and variance. Let us start by considering where data comes from. Conceptually we can have a true function[1] which describes exactly how the data would behave in a completely error-free scenario. Ultimately it is our desire to perfectly estimate this function from the training data set. Bias and variance describe, respectively, how well this function fits the training dataset and how well it generalises to other datasets from the same underlying function.

Figure 11.1 illustrates bias. In this case, we get high bias when the model is very simple (a linear function fitting the curve is shown); the discrepancy will be large between the sample data and the modelled version. As we increase the complexity of the model it is able to better capture the behaviour of the underlying function, and the lowest bias is when it matches the data in the training set best (although not necessarily the underlying function).

Variance contrasts to this, and is a measure of how much the error changes when the model is compared with other datasets from the same underlying function. As shown in Fig. 11.2, a very complex model fits the training data well (overfits this), but the error with another dataset is large. This change in error means there is a large variance. The variance is then reduced by reducing the model complexity to a smoother function. This fits the training data less well, but the change between different datasets is then reduced. Finally, for low variance, the discrepancy in error between datasets is very small.

To minimise error overall, we wish to minimise both bias and variance. There is a trade-off here, however, since bias is large with simple models, and variance is large with complex models; in order to establish the right level of complexity, we must perform validation.

[1] Note we will use regression terms to describe these terms for convenience, but the principles apply just as much to classification

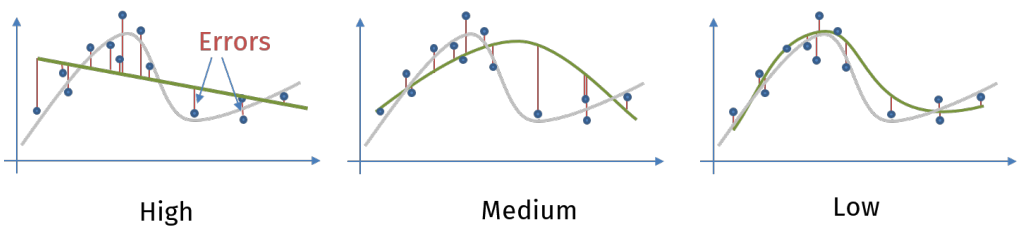


Figure 11.1: Illustration of how bias can change with model complexity.

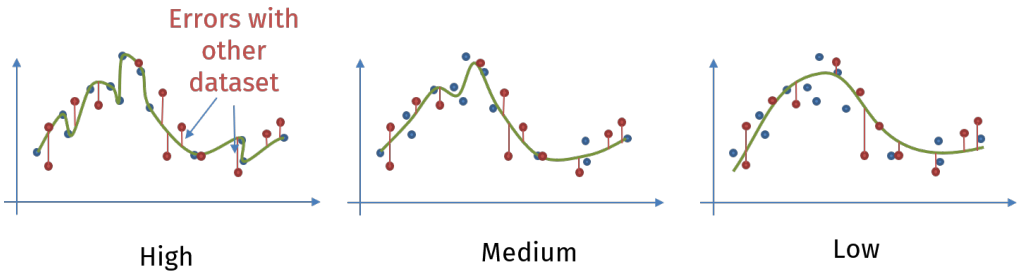


Figure 11.2: This shows how variance is influenced by the complexity of the model.

11.2 Validation

C11B

It is important to validate any model, i.e. check that it produces the correct results with some data which was not used for training. This provides a measure of how generalisable the method is, and whether overfitting has been avoided. This helps us to judge the trade-off between bias and variance. It is critical that this data is "unseen" by the trained model, because we would expect the algorithm to already perform well with any data it has been fitted to.

One simple approach is to keep a subset of the available data back when training, then use this on the trained data to confirm performance. It should be noted that the algorithm will then be considered to be fitted, to an extent, to this validation dataset too (although not as strongly as the training set) through this confirmation process. It should be noted that this process will always perform better when more data is available to both train and test with, because of the random nature of the noise present. You should also recognise that the model can be considered to have learnt from the validation data (although not to the same extent as the training data) – if the model did not match the validation data then it would be rejected, which will imply some dependence.

11.2.1 K-fold cross-validation

C11C

K-fold cross-validation aims to average across several subdivisions to obtain a more accurate estimate of performance. The algorithm is as follows:

1. Split all training data into K equally-sized random parts
2. Use K-1 of these parts for training
3. Use the remaining part for checking performance
4. Discard previous trained model. Repeat with another set of K-1 for training and 1 for testing until all done.

A typical library (such as scikit-learn) will often have a tool available to do this splitting, but it is generally straightforward to do this yourself if necessary.

A question sometimes arises about how the output from K-fold can be used – there are K different models, so assuming performance is acceptable, which one should be used? Actually, what we are trying to do is use K-fold to confirm that the model design is suitable (avoids overfitting etc.) i.e. that parameters like the number of layers/nodes in a neural network, or the amount of regularisation used, is correct.

Note that no specific trained model from the K sets is necessarily used. The routine is intended to establish performance for a model and improve accuracy of the performance values by averaging. In practice you could use the trained version from any case if you are happy with performance.

11.3 Scaling

C11D

Consider a scenario where we have input parameters of both stress and temperature. Stress can be of the order of millions of pascals, i.e. 10^6, while temperature is typically in tens to hundreds

of degrees for most engineering problems. This is a significant discrepancy in scale. We wish to train a machine learning algorithm on this data, and perform a gradient descent approach to fit the data. When we do this, if we assume that a change of $x\%$ in either parameter will cause a similar change in the underlying cost function, the gradient with respect to each parameter will be dominated by the smaller parameter, since a change in stress of 1 megapascal will result in a relatively small change in the underlying cost function compared to the change in temperature of 1°C. The scaling issue also manifests itself in other areas, particularly when we come to consider nonlinear functions where (for example) $1000f(x) \neq f(1000x)$. Other algorithms where scaling is important include ones where some form of distance metric (see Sect. 11.4) is employed, such as K-nearest neighbours and K-Means. Similarly, the components in PCA will be dominated by the largest values, so scaling should be used there, if large differences are expected.

This problem can be minimised by scaling data prior to learning from it. There are a number of ways of achieving this, but one of the most common is to subtract the mean then divide by the standard deviation

$$a_{sc} = \frac{a - \bar{a}}{\sigma_a}. \tag{11.1}$$

Providing this is applied to all variables, they will all then have similar scales and be centred around zero. It should be noted that the standard deviation and mean values from the training data will need to be stored so that the same transform can subsequently be applied when the model is used in practice. It is also possible to scale the full range of each variable in a similar way, based on minimum and maximum values of each dataset, but this achieves a similar goal so will not be discussed further. The scikit-learn library has tools to do the scaling for you.

11.4 Metrics

C11E

Throughout machine learning, the concept of a METRIC appears regularly. A metric is a measure of distance between two points; a small metric would indicate a significant similarity between the two points being considered, and a large metric would indicate dissimilarity. Exactly what this looks like depends on the context and how the problem is parameterised.

A metric D must obey the following rules

- Non-negativity: $D(\boldsymbol{a}, \boldsymbol{b}) \geq 0$
  - The distance metric cannot be less than zero.
- Reflexivity: $D(\boldsymbol{a}, \boldsymbol{b}) = 0$ if and only if $\boldsymbol{a} = \boldsymbol{b}$
  - The metric is only zero when the two points being compared match each other.
- Symmetry: $D(\boldsymbol{a}, \boldsymbol{b}) = D(\boldsymbol{b}, \boldsymbol{a})$
  - Comparing point $\boldsymbol{a}$ to point $\boldsymbol{b}$ is the same as comparing point $\boldsymbol{b}$ to point $\boldsymbol{a}$.
- Triangle inequality: $D(\boldsymbol{a}, \boldsymbol{b}) + D(\boldsymbol{b}, \boldsymbol{c}) \geq D(\boldsymbol{a}, \boldsymbol{c})$
  - The distance from $\boldsymbol{a}$ to $\boldsymbol{c}$ is never more than that from $\boldsymbol{a}$ to $\boldsymbol{b}$ to $\boldsymbol{c}$.

These rules are generally clear when considering the standard Euclidian distance metric

$$D(\boldsymbol{a}, \boldsymbol{b}) = \left(\sum_{k=1}^{n} (a_k - b_k)^2 \right)^{1/2}.$$

The general l_p norm, discussed in Sect. 3.1.1:

$$D(\boldsymbol{a}, \boldsymbol{b}) = \left(\sum_{k=1}^{n} (a_k - b_k)^p \right)^{1/p} \tag{11.2}$$

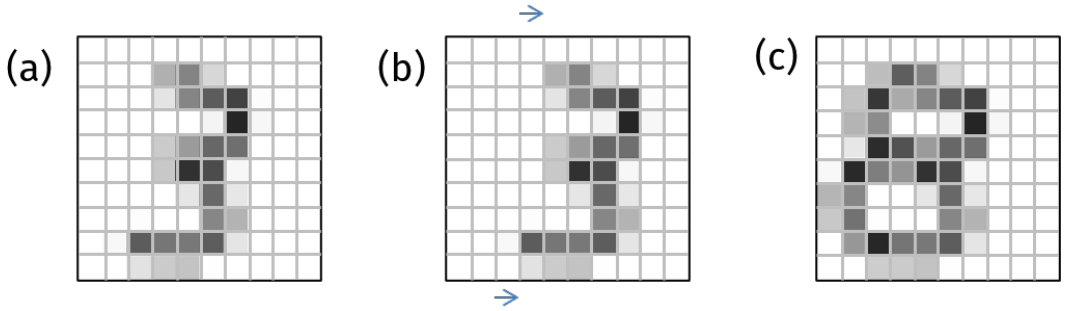


Figure 11.3: Comparing images - what is a good metric? (a) is the number 3, (b) is (a) shifted one pixel right, and (c) is the number 8.

can also be shown to conform to each of these. C11F

In many machine learning approaches, we rely on the concept of a metric. By assessing similarity between points, we can classify a nearest neighbour, or we can minimise a metric to establish the best fitting of a set of learnt parameters. In the majority of simple cases, it is easy to conceptualise how a standard metric like the l_2 norm is a good measure of distance. However, there are some scenarios where this is not possible.

Figure 11.3 illustrates a challenging problem for a metric to deal with. Figure 11.3(b) is Fig. 11.3(a) shifted one pixel to the right, while (c) is a completely different number. However, if we were to just consider summing (somehow) the differences at each pixel, the difference will be much larger for (b) than for (c). However, we would really want (a) and (b), since they are the same number, to have the smallest metric. Solutions to this problem can involve attempting to remove any offset or rotation prior to calculating the metric, extracting features from the data which will be insensitive to these issues, or the application of more complex fitting routines such as deep learning. We will not go into these now, however, it is important to be aware of these issues.

11.5 Data preprocessing and feature extraction

C11G

This section will focus on what happens to the data before a machine learning algorithm operates on it - where it comes from, how it is processed etc. Unless indicated, it is assumed that the same processes are applied to any validation, testing and training datasets, although clearly some tuning may be required from the training dataset. Preprocessing is a vital part of machine learning, and can be where a designer makes a real difference to the performance of the algorithm. Actual components of interest may be hidden, so preprocessing techniques can be vital to help make these more exposed to any learning algorithm.

11.5.1 Data sampling

C11H

The first aspect of using data is acquiring it! Data can come from a huge variety of sources, and can be a single value, a set of values of different types, many samples of the same type, taken in time and/or space, or any combination of these. Within engineering, data is often made from measurements of physical quantities, and hence limited by the actual measurement system; this is an enormous field in itself which will not be discussed fully here.

One aspect to consider is errors in the data, which may manifest themselves as noise. Depending on how these errors are produced we may be able to process to minimise these; for example, filtering can help if we know the frequency spectrum of the signal of interest or the noise itself. The noise will contribute to the statistical distributions which can lead to bias and variance, and may be CORRELATED or UNCORRELATED. Uncorrelated noise is common random noise, which can be reduced by taking the average of many cases, whereas correlated noise may be caused by some repeating error, appearing the same (in some way) across all measurements, and cannot be removed by averaging.

Quantisation and sampling are other aspects you should already know about from the rest of

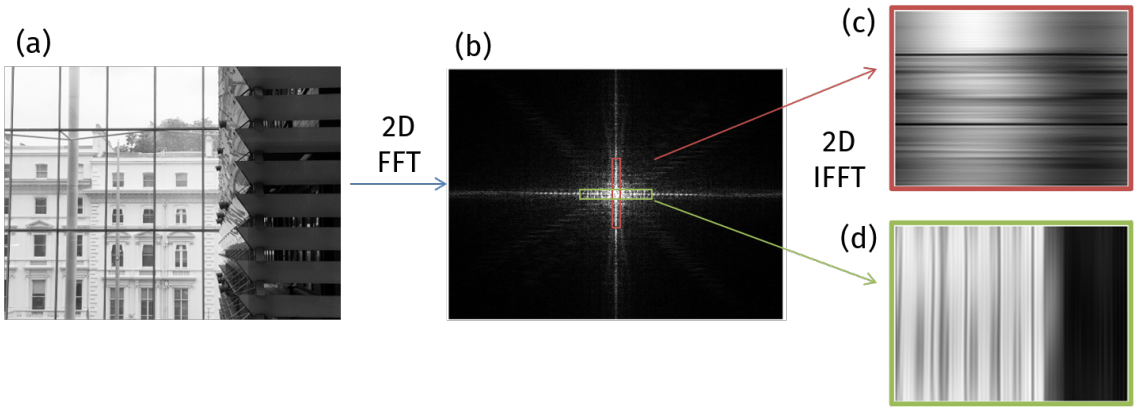


Figure 11.4: Fourier transform of an image. (a) shows the original image, (b) is then the amplitude of the Fourier transform, with the low frequency components at the centre. (c) and (d) are inverse Fourier transformed images of different components, from within the two rectangles marked in (b).

your degree (ME2 Mechatronics covers this). Quantisation reflects the precision to which data is recorded; for example, 8 bits enables $2^8 = 256$ different values, and the sampling rate will dictate how regularly (usually in time, but also in space or any other domain) we sample the data. Increasing either or both of these will generally increase the cost of the system and the amount of memory required to store the data.

11.5.2 Preprocessing

C11I

Having established what forms the data may take, we can now consider how this data can be preprocessed - what approaches and techniques can be applied to the data to help any subsequent algorithm to extract the necessary components.

Fourier transforms appear across many areas of engineering. A Fourier transform will transform a signal (commonly time) into its frequency spectrum. This has a number of applications. Potentially, this frequency spectrum may be adjusted, filtered, to remove noise or other unwanted components, prior to inverse transforming the data back. Or a single frequency can be extracted and fed into the ML algorithm. While we commonly consider Fourier transforms for time-frequency transformations, they can also be used in other spaces, such as spatial Fourier transforms; it is also possible for these to be applied in multiple-dimensions. This is particularly relevant to image processing.

Figure 11.4 shows how a 2D Fourier transform can identify particular components of an image. Figure 11.4(a) shows an original image and Fig. 11.4(b) shows the 2D Fourier transform (generated via the Fast Fourier Transform, FFT). It is clear that there are strong components in the vertical and horizontal directions, which correspond to horizontal and vertical features respectively in the original image. Fig. 11.4(c) shows what happens if we filter to keep just the components along the y axis, producing a series of horizontal features. Figure 11.4(d) shows the vertical features by extracting components along the other axis. From these two processed images we could anticipate that a machine learning model may be able to preferentially learn from the features contained within one or other of these images.

Another processing technique which is often applied to images in particular is thresholding. Figure 11.5 illustrates thresholding the image in Fig. 11.4(a) at two different levels, 207 and 65 out of 256 colours in the original. Through doing this it is possible to minimise much unwanted background information in certain areas, which may not be of interest. Thresholding may be applied to a filtered image which has had a broad (low frequency) variation removed from it first, to account for different lighting (for example) in different parts of the image. One issue with thresholding is that a suitable level must be selected, and the method is also quite sensitive to noise and errors in the data.

In many cases we might want to differentiate a function as part of preprocessing. This can be

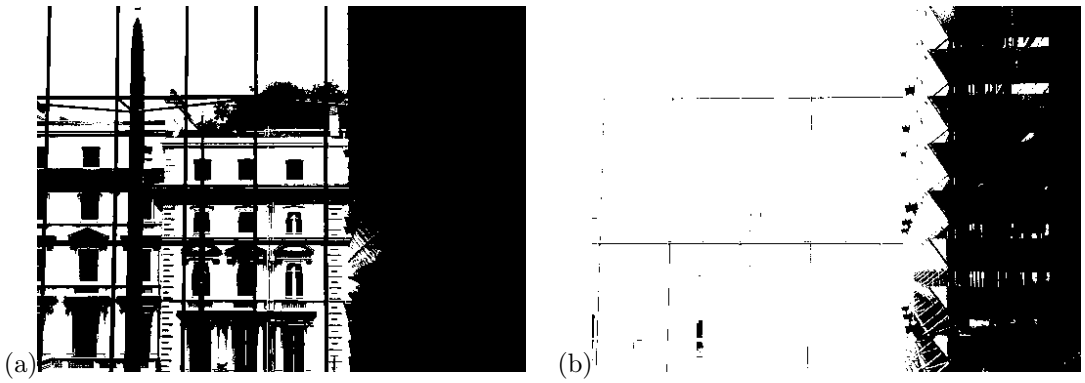


Figure 11.5: Thresholding image Fig. 11.4(a), consisting of 256 colours, at two different levels: 207 in (a) and 65 in (b).

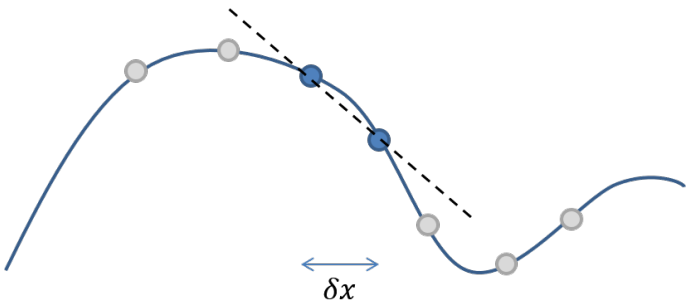


Figure 11.6: Derivative estimation. It is possible to approximate the derivative of the function from its samples and the spacing by assuming it is linear between two points, i.e. we calculate the gradient of the dashed line.

achieved by utilising the first term of the Taylor series expansion between two points, giving

$$\frac{\partial f}{\partial x} \approx \frac{f_{t+1} - f_t}{\delta x} \tag{11.3}$$

where f_t, f_{t+1} are samples of the underlying function f, and δx is the step size between samples. This is a forward approximation, and f_t, f_{t+1} could be replaced with f_{t-1}, f_t instead. There are a variety of higher-order methods which can be used too, but these are beyond the scope of this course. Note that the δx term is constant throughout, so can be dropped if it is only the relative derivative which is important. In higher dimensions the derivative can be calculated in the same way in each respective dimension, and if necessary, different terms can be combined with

$$f_{deriv2D} = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}. \tag{11.4}$$

11.5.3 Feature extraction

C11J

So far we have considered processing all the data available to us, which can be computationally slow and resource intensive, as well as potentially sensitive to redundant components of the data, such as noise and transforms/rotations. Feature extraction involves processing the data to extract new parameters. As with much of this section, these approaches are common in image processing. They are a form of dimensional reduction, where the number of parameters in a problem is reduced. This can also be viewed the other way, where dimensional reduction tools, like PCA (Sect. 9.2), can act as feature extractors.

Edge detection is one common feature extraction task – this becomes step detection in 1D signals or change detection over time. It can also form a basis for further feature extraction, where if we know where the boundaries are we can start to extract the shapes and sizes of various features. Edge detection can also help to eliminate many of the issues of processing directly on the raw image, such as lighting variation etc.

While there are a huge number of different approaches to detecting edges in images, we will briefly give an overview of one, the use of derivatives. When the first-order derivative is large, this indicates a rapid change in the underlying image, which indicates a boundary. Note that an image would typically be smoothed first, via a filter, to minimise noise or spurious features. The difference in each direction can be calculated with eq. (11.3) and combined with eq. (11.4). The RMS of this can be found, and the peak values can be identified as the desired boundaries.

In some cases it is possible to perform shape fitting, where a known shape is sought from an image. Techniques such as those explored in [2] involve fitting a shape, described by a set of points around the boundary, to an image by iteratively shifting the points until a best fit is achieved. These approaches can avoid issues with rotation and translation highlighted at the end of Sect. 11.4.

C11K

Finally, we should mention that the concept of deep learning, where a neural network of many layers is utilised, is considered to automatically perform feature extraction. The early layers perform the extraction, then the later layers use these to process the outputs - as an example, early layers may detect edges, but later layers may detect digits. These networks are designed to operate directly on the raw unprocessed dataset.

11.6 Questions

1. Sketch a graph plotting typical variance, bias and total error as the number of trainable parameters within an ML model increase.

2. What will the mean and standard deviation of the scaled variable from eq. (11.1) be?

3. I have points A and B at (0, 0) and (3, 2) respectively.

   (a) Find the coordinates (Cx, Cy) of the point C which is equidistant from both A and B, as measured under both the L1 and L∞ norm, subject to the limits $0 < Cx < 2$ and $2 < Cy < 4$.

(b) Show that the L2 distances are also equal at this point.

11.6.1 Solutions

1. This is the plot given towards the end of lecture 3, with number of trainable parameters on the x axis (effectively complexity).

2. 0 mean and 1 standard deviation.

3. (Q4 2021)
   a) Define $X = 3, Y = 2$, as horizontal and vertical distances respectively between the two given points. Take $x_1 = cx, y_1 = cy$ as the horizontal and vertical distances from point A and the equidistant point, and x_2, y_2 as the same from point B. $x_1 + x_2 = X, y_1 + y_2 = Y$.
   L1 dist:
   Sum of absolute components
   $|x_1| + |y_1| = |x_2| + |y_2|$
   $|cx| + |cy| = |X - cx| + |Y - cy|$
   $cx + cy = X – cx – Y + cy \quad$ (since $Y < cy$)
   $cx = \frac{(X–Y)}{2} = 0.5$

   Linf dist:
   Maximum of absolute components
   $Max(|x_1|, |y_1|) = Max(|x_2|, |y_2|)$
   $Max(|cx|, |cy|) = cy$ since $cx = 0.5$ from above and cy must be > 2 from question
   $Max(|x_2|, |y_2|) = Max(|X - cx|, |cy - Y|) = Max(|2.5|, |cy - 2|)$
   $cy = cy - 2$ is incompatible (and also unlikely that $cy - 2 > 2.5$) so take $cy = 2.5$
   Point C is at $(0.5, 2.5)$

   b) L2 dist is sum of the squares (rooted)
   L2 dist (squared) from $(0, 0) : 0.5^2 + 2.5^2 = 6.5$
   $x_2 = 3–0.5 = 2.5$
   $y_2 = 2–2.5 = -0.5$
   L2 dist (squared) from $(3, 2) : 2.5^2 + (-0.5)^2 = 6.5$
   Same dist.

Bibliography

[1] N. N. Taleb, *The Black Swan.* Allen Lane, 2007.

[2] T. Cootes, C. Taylor, D. Cooper, and J. Graham, "Active Shape Models-Their Training and Application," *Computer Vision and Image Understanding*, vol. 61, pp. 38–59, jan 1995.