S1: Principles of Data Science

Lecture Notes

University of Cambridge

MPhil in Data Intensive Science

Matt Kenzie

mk652@cam.ac.uk

Rutherford 951

Michealmas Term 2023

“Data! Data! Data!.. I can’t make bricks without clay.”

– Sherlock Holmes, The Adventure of the Copper Beeches

Contents

Overview

Here are a few important links to help you find the relevant material for the course:

- The module webpage is `https://mphildis.bigdata.cam.ac.uk/principles-of-data-science/` [1].
- The course material, including lecture captures, lecture notes, jupyter notebooks, problem sheets *etc.* can be found on this moodle page.
- Course resources for the MPhil are distributed via our own `gitlab` group page which can be found at here. Please be aware that are some issues with access to the `gitlab` repo at the moment whilst we figure out permissions so in the meantime I will put all of the material on the moodle page anyway.
- The track listing and request form can be found here, the playlist on `YouTube` can be found here.

It's worth noting that whilst this course is nominally called *Principles of Data Science* it could be more accurately described as an "Introduction to Statistical Methods for Applications to Data Intensive Science". The course contains 24 lectures, which are all delivered in Michaelmas Term. The course will give an overview of probability and statistical theory and then show *classical* (also known as *frequentist*) methods for estimating parameters, distributions and hypothesis testing. We will also cover some of the core concepts at the heart of machine learning for which more in-depth discussion and specific applications are covered in the *Applied Data Science* (Michaelmas) and *Application of Machine Learning* (Lent) modules. More advanced statistical methods, predominantly Bayesian in their nature, are covered in the *Statistics for Data Science* (Lent) module.

Assessment

The course contains 24 lectures, which are all delivered over an 8 week period in Michaelmas Term. There will be a problem sheet every two weeks, four in total, which give some deeper insights into the course material and examples of the sort of questions you may end up facing in the coursework assessment and in the exam. You are expected to go through these and provide solutions which are discussed at fortnightly supervision meetings (small group discussion sessions). The problem sheets are not formally assessed. Problem sheets will be distributed in odd numbered weeks (weeks 1, 3, 5 and 7), the corresponding supervision

[1]Please be aware that for next year the module name has changed to "*Statistical Methods for Data Intensive Science.*"

should happen two weeks later (weeks 3, 5, 7 and 9). I will be one of the supervisors and will alternate which classes I see so that I should have at least one supervision with each of you by the end of the term.

The formal assessment for this module is based on two elements. One is a 3 hour written exam which takes place at the beginning of Lent Term and counts for 50% of the module score. The other is a piece of coursework which will require some code writing and a project write-up (max 3000 words). It will be set in Week 7 of Michaelmas Term (Mon 13th – Fri 17th Nov) and is due to be submitted by 15th December. The coursework counts for 50% of the module score and will be assessed on scientific accuracy, coding skills, presentation skills and written skills.

Please **be warned**, terms at Cambridge are short and intense. I advise you to try and keep on top of your work load and avoid getting to far behind, because it is hard to catch up. If you need help please come and talk to us.

Supervisions and Office Hours

I will have open office hours on Wednesday mornings from 10:00am - 12:00pm from Wednesday 11th October – Wednesday 22nd November (inclusive). If there is anything related to the course you would like to discuss you should be able to find me then. If I'm not in the office I should be close by but you can of course email me instead. My office is in the Cavendish Laboratory Rutherford Building Room 951.

Supervisions (small group teaching, going over the problem sheets) will happen fortnightly on Thursday afternoons. The dates for the supervisions are 19th October, 2nd November, 16th November and 30th November. There are three sessions at 2:00pm, 3:00pm and 4:00pm. You will be allocated to a session and a supervisor at the start of term.

Learning Outcomes

By the end of this course you should:

- Understand how to visualise datasets and how to produce, contrast and defend plots of statistical distributions
- Be able to perform basic sample estimates on datasets
- Be able to critique and analyse dataset selection, curation and processing
- Be able to discuss and appriase the two philosophies of statistics
- Understand and be able to exploit and manipulate random variables, and evaluate and contrast probability distributions of random variables
- Know the distributions and be able to interpret the properties of various common probability distributions
- Employ and justify the central limit theorem
- Be able to describe and perform frequentist evaluation of estimators, and compute the mean and variance of a distribution

- Understand and be able to produce different methods of estimation including least squares and maximum likelihood
- Understand the meaning, production, reproducibility and interpretation of frequentist confidence intervals
- Understand the purpose and benefits of resampling methods and be able to use them in practise
- Be able to explain and recognise appropriate deployment of more advanced methods including density estimation, expectation maximisation and Gaussian mixture models

Lecture Timetable

Lecture slots are for 1 hour. Please be on time. Lecture capture and webcast will be available (and should appear automatically on the panopto page) but I strongly encourage you to attend in person if you are able to. Requests for songs at the beginning and middle of lectures to mk652@cam.ac.uk or alternatively fill out a blank cell in the track listing spreadsheet. For convenience I have reproduced the module timetable below in Table 1 but please don't consider this the official version as I may have made a copy/paste error and may not update to the latest version.

Table 1: Lecture timetable and overview

	Title	Day	Date	Time	Location	Content
1.	Understanding data	Friday	06/10/2023	16:00	East 1	Course overview, datasets, visualisation, quantifying data.
2.	Learning from data	Monday	09/10/2023	12:00	East 1	Data structures, performance criteria and metrics, cross-validation
3.	Probability theory	Tuesday	10/10/2023	10:00	East 1	Definitions and properties of probability, random variables, probability density functions, changing variables
4.	Probability distributions	Friday	13/10/2023	11:00	East 1	c.d.f.s, joint, marginal and conditional p.d.f.s, Bayesian inference
5.	Distribution properties	Monday	16/10/2023	12:00	East 2	Expectation values, the characteristic function
6.	Common distributions	Tuesday	17/10/2023	10:00	East 1	The binomial, Poisson, normal and multi-variate normal distributions
7.	More distributions and generation	Friday	20/10/2023	11:00	East 1	The exponential, polynomial and chi-squared distributions, p.p.f. generation, accept-reject generation
8.	Limit theorems	Monday	23/10/2023	12:00	East 2	Convergence, the central limit theorem, propagation of errors
9.	Estimates	Tuesday	24/10/2023	10:00	East 1	Consistency, bias, efficiency, sample estimates of mean and variance
10.	Max. likelihood estimate	Friday	27/10/2023	11:00	East 1	The likelihood, minimum variance bound, maximum likelihood estimation
11.	More max. likelihood	Monday	30/10/2023	12:00	East 2	Many parameter likelihoods, profile likelihood, extended ML
12.	Binned ML and least-squares	Tuesday	31/10/2023	10:00	East 1	Binned ML, least-squares fits, Wilks' theorem, drawing contours and conversions between p-value and Z-score
13.	Method of moments	Friday	03/11/2023	11:00	East 1	Discrete profile method and method of moments
14.	Goodness of fit tests	Monday	06/11/2023	12:00	East 1	Chi-squared test and KS test
15.	Confidence intervals	Tuesday	07/11/2023	10:00	East 1	Confidence intervals, parameter estimation at boudaries, constraints, FC intervals
16.	Hypothesis testing	Friday	10/11/2023	11:00	East 1	Hyopthesis testing, the Neymann-Pearson lemma
17.	Limit setting	Monday	13/11/2023	12:00	East 1	Failure of ML, The CLs method, the FC method
18.	Resampling methods	Tuesday	14/11/2023	10:00	East 1	Bootstrapping, jackknifing, BCa
19.	Worked examples Part 1	Friday	17/11/2023	11:00	East 2	Work through some demonstrative examples
20.	Worked examples Part 2	Monday	20/11/2023	12:00	East 1	Work through some demonstrative examples
21.	Concepts for machine learning	Tuesday	21/11/2023	10:00	East 1	Forward modelling, optimisation and regularisation
22.	Density estimates	Friday	24/11/2023	11:00	East 2	Kernel density estimation
23.	Expectation maximisation and GMM	Monday	27/11/2023	12:00	East 1	Expectation maximisation and Gaussian mixture models
24.	Projecting joint p.d.f. components	Tuesday	28/11/2023	10:00	East 1	Sliced fits, sWeights, Custom Orthogonal Weight functions
Exam		**Monday**	**08/01/2024**	**09:30**	**East 2**	**Any of the above!**

Reading Material

There is almost limitless material available online, which helps to explain the concepts in this course. Most of what we cover are well established methods in statistics and machine learning so have excellent descriptions on Wikipedia and via the many and varied data science articles. There are some great tutorials available, nice coding examples and various demonstrations of the tools we will use. I encourage you to ask google (other search engines are available) for any parts of the course you are struggling with, you can often find really insightful discussions and examples. You should have received some guidance about the use of ChatGPT in your induction and in the course handbook. It is a great resource, that can aid your learning, but should be used with care and caution. Like with any resource you should critique and understand its methods to ensure you are not being misled and do not make mistakes.

Below I list some of the text books and resources I have found useful in my career and have made extensive use of when preparing this course. There are many others available and if you come across a particular favourite of your own which is not listed here then *please* tell me about it. My background is predominantely in particle physics so I tend to favour the statistics books which are themselves written by particle physicists and as such tend to use examples that I understand. This is a rather shameful case of observation bias so I would be interested to hear your experiences and what literature you find useful.

- F. James, *Statistical Methods in Experimental Physics*, 2nd edition [1]. This guy is like the godfather of statistics in particle physics and this book is absolutely brilliant, although some find it rather mathematical and dry. Fred James is also one of the original authors of the `Minuit` optimiser which we will make extensive use of in this course. This is my go-to favourite stats book.

- R. Barlow, *Statistics: A Guide to the Use of Statistical Methods in the Physical Sciences* [2]. This has a much more narrative style and does not go into quite as much depth as James. Barlow (also a particle physicist) has an amusing and lighthearted writing style which make this quite enjoyable to follow. He is only one of the only authors willing to delve into the dark art of systematic uncertainties.

- S. Brandt, *Data Analysis: Statistical and Computational Methods for Scientists and Engineers*, 4th edition [3]. This is probably the most "text-book" like. It is a lot more descriptive and provides much more explanation and visualisation so I would recommend it if you find the others hard to follow. It's been through many editions and is very thorough in the areas it covers.

- A. Stuart and J. K. Ord, *Kendall's Advanced Theory of Statistics*, 5th edition [4, 5, 6]. This absolute monster of a tome (over three volumes) is perhaps the best reference book. It really is *the* book on statistics and has had several modern rewrites since its original version by Kendall many years ago. I find it a bit dull and lacking context but for quickly looking up a theorem or topic it is perfect.

- L. Lyons, *Statistics for Nuclear and Particle Physicists* [7]. Rather as it says on the tin. Another stats book written by a particle physicist, it is perhaps a bit specific to more traditional methods and their applications in particle physics only.

- G. Cowan, *Statistical Data Analysis* [8]. Another particle physicist I'm afraid but this is a really great book. It strikes the balance somewhere between the overview and context provided by Barlow and the mathematical rigour and detail provided by James. Cowan has also written the Particle Data Group review article on Statistics [9] which is itself an excellent short overview.

- K. Cranmer, *Statistics and Data Science* [10]. This is not a book but a set of online course notes which are really excellent and include many nice coding examples. They are presented in the form a `python jupyter notebook`.

Coding Resources

The topics we cover in this module, and the nature of this MPhil course itself, lend themselves to explanations via code examples. I *really* encourage you to try out some of the coding examples yourself including testing and tweaking them. For the demonstrations in this module I will exclusively code in `python`. That's predominantly because I know it, it's easy to understand and run, and because there are great libraries out there which do most of the heavy lifting for us already. The examples I give in this course **are not optimised for performance**. For your specific applications you will have to figure out what is best for you. Many of the little example code snippets I give will probably not scale nicely and are written for convenience and ease of understanding, not so that they run quickly. **You have been warned**.

The various code snippets for the course are kept in the module `gitlab` page. I have created a specfic `conda` environment file in that `gitlab` project, called `environment.yml`, which should setup all the packages and libraries you need to run any of the scripts and code used in this course. This requires access to the `conda` package manager which is very straight-forward to install on any system. I highly recommend using `conda` environments for each project you run. You can create the environment, in this case I name it "`mphil`", on your machine using

```
conda create -n mphil --file environment.yml
```

You can then make this environment the "active" environment with

```
conda activate mphil
```

Throughout these notes I create all example plots with `matplotlib` (sometimes created with the `seaborn` wrapper). I have a `matplotlib` style file (also in the module `gitlab` repository) that I use for all plots, which gets loaded at the top of a lot of my scripts. Throughout the lecture notes I will display various snippets of code which will have a light grey background, and yellowish-brown line numbers on the left-hand side. These code blocks will look like this (sometimes they will also have captions and be referred to as *e.g.* Code Block 1).

```python
# a snippet to test displaying of code blocks in LaTeX
import numpy as np
import matplotlib.pyplot as plt

x = np.random.normal(0,1,size=1000)
y = np.random.normal(4,2,size=1000)
```

```python

plt.scatter(x,y,s=2)
```

Code Block 1: Example of a code block in these lecture notes

At other points in the notes I will display the output from the terminal. This will be in a pink shaded box and look like this:

```
Hello world!
```

Chapter 1

Understanding Data

If we're going to be doing some data science then having a look at some data, and any properties we can infer from it, seems like a sensible place to start. I cannot stress enough how useful it is to actually visualise data that you have. One of the first things I ask PhD students to do is start taking a look at (*i.e.* making plots of) the data they will be analysing. It really helps you to gain insights and an intuition for what you are working with.

Perhaps a few points to remember:

- When we are dealing with some dataset or other, what we typically have is referred to as a *sample*, S, which has some number of independent *events* or *observations*, N, each of which is a measurement of one (or more) variable(s), x.

- This sample is normally a subset of a bigger *population*, P.

- The population, P, and the sample, S, are drawn from some underlying probability distribution which we often want to use the sample to infer the properties of.

- These type of data are often referred to as *independent and identically distributed* (i.i.d.) which means that different events (or datapoints) within the sample do not depend on each other and also follow the same underlying "*true*" distribution.

For Example: I may have a data sample, S, containing the heights of each student enrolled on this course, h_i. The events h_i are i.i.d.. The total sample size is N which is the number of students on the course, the subscript i labels each individual, where $i = 1, 2, ..., N$. The population P, in this case, may refer to all graduate students at Cambridge, or all people living in Cambridge, or all people in the world.

Real life data (and often our simulations of it) contains noise, which is usually, but not always, Gaussian distributed. Statistical analysis is what allows us to infer the properties of nature given that whenever we make a measurement it is always subject to some *randomness*. Sometimes this is referred to as *resolution* or *error* but I find this a bit misleading. The random nature of inferring properties from a finite data sample is somewhat irreducible (without increasing the data sample). For example when tossing a coin or rolling a dice there is no error or resolution in the measurement but there is a still a random outcome everytime you toss or roll.

1.1 Visualising data

There are some great tools out there for helping you to visualise data, you will learn more about these in the other modules on this course. In these lectures I make extensive use of `pandas` for conveniently storing and accessing datasets, and `matplotlib` for visualising data and distributions. There is also the `seaborn` package which has a wide variety of different plugin data visualisation tools that I would highly recommend. Of course lots of our statistical modelling will make extensive use of `numpy` and `scipy`, vital packages for any data analysist. Please be aware that in the examples below I generate my own data, typically from some probability distribution (usually a Gaussian distribution), but this is just for demonstrative purposes. In real life you will likely be presented with the data (or have collected it yourself) so will not need to generate it first.

1.1.1 Histograms

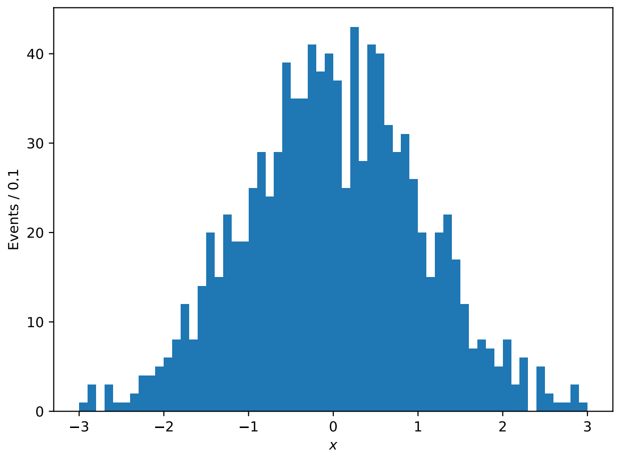
Probably the first point of call in visualising a distribution is to make a histogram. The boundaries of the histogram bins are predefined and the count in each bin is increased when an event falls between those boundaries. The standard convention is that x goes in bin i if $l_i \leq x < u_i$, where l_i and u_i are the lower and upper boundaries of bin i, respectively. The number of events in a given bin is proportional to the area of the bin (*i.e.* depends on the width of the bin as well as its height) which is not technically the same as a bar chart in which the number is proportional to the height. A common and convenient representation of a histogram is as a probability density, where the histogram y-values are scaled such that the area of each bar (the width multiplied by the height) gives the probability for an outcome in the range $l_i \leq x < u_i$. Consequently the sum of the areas of all bars will be unity.

Here's an example where we generate some data and then plot it using the `matplotlib` `hist` function.

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

# generate some random data
x = np.random.normal(size=1000)

# plot with matplot lib
plt.hist(x, range=(-3,3), bins=60 )
plt.show()
```

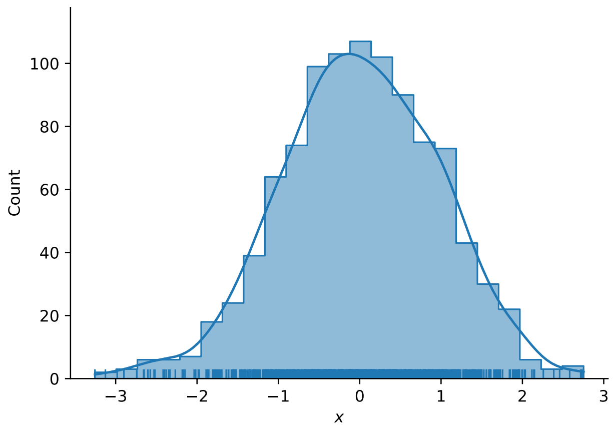


Here's another example where we use the power of the `seaborn` package which additionally plots us an estimate of the probability density (using a kernel density estimation - details of this are covered in Sec. 4.4) and gives us the "rug" plot showing how the density of events is distributed along the x-axis. `seaborn` is a really *awesome tool* for data visualisation.

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# generate some random data
x = np.random.normal(size=1000)

# plot with seaborn
sns.displot(x, element="step", kde=True, rug=True)
plt.show()
```



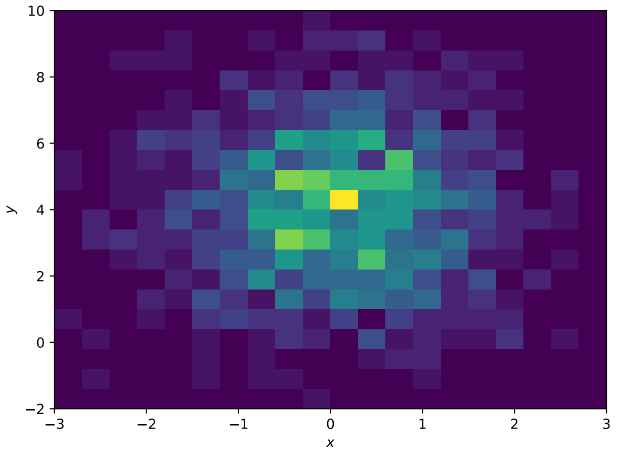
This helps us to see what our data (at least in the variable x) looks like. In this case we see that it looks kind of Gaussian distributed, which of course we know that it is because we generated it from a Gaussian distribution (with `np.random.normal`). We will return to the properties of Gaussian distributions later (see Sec. 2.4.3) and we will return to how we can generate events according to different distributions (see Sec. 2.4.9).

Histograms can be multi-dimensional and whilst we only have the ability to make a plot on a page in 2D it is in principle possible to plot any dimension of histogram just in 1D, because each bin is unique so you can just line them all up one one axis. Here is an example of a 2D histogram (actually shown in 2 dimensions). In this case the histogram is shown with $20 \times 20 = 400$ bins in 2D. We could equally show this same histogram in 1D with 400 bins.

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

# generate data
x = np.random.normal(size=1000)
y = np.random.normal(4,2,size=1000)

# plot data
plt.hist2d(x,y,range=((-3,3),(-2,10)), bins=20)
plt.show()
```



1.1.2 Scatter Plots

Another obvious visualisation tool is the scatter plot, which shows dependence (or lack of) between one variable in your dataset and another. This time I generate a slightly more complex dataset, just to demonstrate how good `seaborn` can be at visualsing data, where we have two variables, (x, y), which are correlated but in a different way for two classes of data. The two classes of data, here I imaginatively call them *Class A* and *Class B*, also have a relative shift in their central x and y positions.

First I will generate the data and then store it in a `pandas` `DataFrame`:

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# generate data for the two classes
xA = np.random.multivariate_normal( mean=[0,4], cov=[[1,1],[1,4]], size=1000 )
xB = np.random.multivariate_normal( mean=[2,3], cov=[[1,-1],[-1,4]], size=1000
    )

# put in a pandas dataframe
dfA = pd.DataFrame( xA, columns=['x','y'] )
```

```
dfB = pd.DataFrame( xB, columns=['x','y'] )
dfA['Class'] = "A"
dfB['Class'] = "B"
df = pd.concat( (dfA,dfB), ignore_index=True )
print(df)
```

which when I print it gives me:

```
              x         y  Class
0      0.478705  6.214592      A
1      1.136541  3.268415      A
2     -0.346498  0.875564      A
3      1.015408  3.419921      A
4      0.004856  6.122761      A
...         ...       ...    ...
1995   1.911136  5.090560      B
1996   3.894796  1.225123      B
1997   2.962565  4.105827      B
1998   2.744363  4.454348      B
1999   2.375640  4.934752      B

[2000 rows x 3 columns]
```

Now to demonstrate how great seaborn is for visualisation we can make a couple of plots of this data, one scatter and one histogram, in Fig. 1.1.

```
# seaborn scatter with kde in margins
sns.jointplot( df, x='x', y='y', hue='Class', s=10 )
plt.savefig('figs/data/scatter1.pdf')
plt.show()
```

```
# seaborn hist2d with 1d proj in margins
sns.jointplot( df, x='x', y='y', hue='Class', kind='hist', alpha=0.7 )
plt.savefig('figs/data/scatter2.pdf')
plt.show()
```

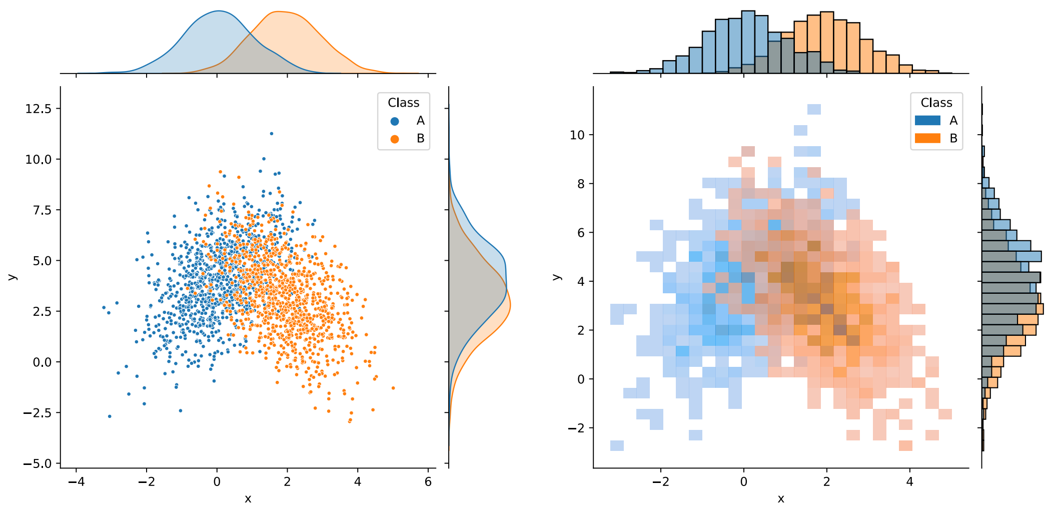


Figure 1.1: A demonstration of some cool seaborn style plots, scatter on the left and histogram on the right.

1.2 Measuring the moments

So what did we learn? Well, aside from the fact that `seaborn` seems like a pretty cool visualisation package, we didn't learn much. We certainly didn't learn anything quantitative about our data so let's move on to computing some actual numbers which help to describe the distributions in our data.

1.2.1 Averages

If you want to describe the data with one single number, the most popular (and arguably the most meaningful) is almost certainly the *arithmetic mean*, often simply referred to as the *mean*. This is often denoted with a "bar" over the quantity of interest.

So if we have a set of N data points in some variable x,

$$\{x_1, x_2, x_3, ..., x_N\},$$

then the arithmetic mean of x is

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i. \tag{1.1}$$

This gives the mean of the actual values but note that you can also compute the mean of any function, $f(x)$, with

$$\bar{f} = \frac{1}{N}\sum_{i=1}^{N} f(x_i). \tag{1.2}$$

If the data is *binned* (*i.e.* in a histogram) and you have n_j counts in bin j with central value x_j and total number of bins B then

$$\bar{x} = \frac{1}{N}\sum_{j=1}^{B} n_j x_j, \tag{1.3}$$

$$\bar{f} = \frac{1}{N}\sum_{j=1}^{B} n_j f(x_j), \tag{1.4}$$

but this will not be as precise as using the *unbinned* data, so should only be used if you don't have access to the individual data points.

There are some other alternatives which are occasionally used instead of the arithmetic mean. The *mode* is the most popular value in a dataset (for a probability distribution it is the most probable value). The *median* is the middle value such that half the dataset has values below the median and half have values above the median.

1.2.2 Measuring the spread

The mean tells us something useful about our data, in particular it gives an indication of the region most of it is populated. However, if we want to know the extent of that region then what we need to quantify is the spread of our data about that mean. For this we use the variance, which quantifies the average squared distance from the mean. The reason we use the square is because otherwise the negative and postive deviations would cancel. *The proof of this is left for the first problem sheet.*

The variance is defined as

$$\begin{aligned} V(x) &= \frac{1}{N}\sum_i^N (x_i - \bar{x})^2 && (1.5) \\ &= \frac{1}{N}\sum_i^N \left(x_i^2 - 2\bar{x}x_i + \bar{x}^2\right) \\ &= \frac{1}{N}\left(\sum_i^N x_i^2 - 2\sum_i^N \bar{x}x_i + \sum_i^N \bar{x}^2\right) \\ &= \overline{x^2} - 2\bar{x}^2 + \bar{x}^2 \\ &= \overline{x^2} - \bar{x}^2. && (1.6) \end{aligned}$$

As for the mean, in (1.2), we can also define the variance for a function of x,

$$\begin{aligned} V(f) &= \frac{1}{N}\sum_i^N \left(f(x_i) - \bar{f}\right)^2 && (1.7) \\ &= \overline{f^2} - \bar{f}^2. && (1.8) \end{aligned}$$

From a statisticians point of view the variance is nice to work with. It has linear addition properties and is easier to manipulate. However from a scientists point of view it doesn't have the same dimensions (or units) as the mean which make it hard to interpret in the context of the spread. This is straightforwardly resolved by defining a quanity which does have the same dimensions as the mean, the *standard deviation*, which is simply the square root of the variance,

$$\begin{aligned} \sigma &= \sqrt{V(x)} && (1.9) \\ &= \sqrt{\overline{x^2} - \bar{x}^2} && (1.10) \\ &= \sqrt{\frac{1}{N}\sum_i (x_i - \bar{x})^2}. && (1.11) \end{aligned}$$

It is at this point where we have to introduce some words of warning about different definitions of the standard deviation. Barlow says the fact we call it "standard" must be some kind of sick joke [2]. At this point I want to show you what these alternative definitions are but it requires us to have some knowledge we have not yet covered in the course about probability distributions, expectation values and estimates. We will cover these properly once we get to Sec. 2.3 and Sec. 3.1.3. If you are struggling to follow the different descriptions below I suggest you wait until we have covered the later material in the course and then come back to it. For now lets simply bear in mind the following points

- The mean, $\bar{x}$, we have computed above in (1.1) is the *sample mean*.

- There is some underlying *true mean*, often labelled μ. This is the mean of the distribution our sample comes from[1]. If you know the underlying distribution then this

[1]We cover distributions in more detail later in Sec. 2.3.

true mean is the expectation value of that distribution[2],

$$\mu = \langle x \rangle. \tag{1.12}$$

- The common notation makes it easy to distinguish between the *sample mean*, labelled $\bar{x}$, and the *true mean*, labelled μ.
- The underlying distribution also has a *true variance* and *true standard deviation* but these are also typically labelled as $V(x)$ and σ which makes it harder to distinguish them from the *sample variance* and *sample standard deviation*. In terms of expectation values these are,

$$V(x) = \langle x^2 \rangle - \langle x \rangle^2, \tag{1.13}$$

$$\sigma = \sqrt{\langle x^2 \rangle - \langle x \rangle^2}. \tag{1.14}$$

Some people instead chose to define the standard deviation from the *true mean*, μ, instead of the *sample mean*, $\bar{x}$, such that

$$\sigma = \sqrt{\frac{1}{N}\sum_i (x_i - \mu)^2}. \tag{1.15}$$

In some sense this is a more fundamental quantity than the one we define in (1.11) but in practise it's useless if you do not know the *true mean*, μ. However, there is a consequence of using the determination in (1.11) because we use the sample to obtain an estimate of both the mean and the standard deviation and therefore have used the sample twice. This means the estimate of the standard deviation given in (1.11) is a *biased estimator*. An *unbiased estimate* of the variance, which when square rooted also gives an *unbiased estimate* of the standard deviation[3], is[4]

$$s = \sqrt{\frac{1}{N-1}\sum_i (x_i - \bar{x})^2}. \tag{1.16}$$

Notice now I use the symbol, s, in order to distinguish it from σ.

Later in the course we will discuss statistical inference, estimates, expectation values and probability distributions so we needn't worry too much about these different defintions at this point. For now let's just stick to our sample defintions of the mean and spread as given in (1.1) and (1.11), respectively.

1.2.3 Higher moments

So far we have seen the mean and the square of the deviation but we can in general define higher order moments of a sample. In general the r^{th} *moment*, sometimes also called the r^{th} *algebraic moment* of x, is

$$a^r = \frac{1}{N}\sum_i^N x_i^r. \tag{1.17}$$

[2] We cover expectation values of distributions in more detail later in Sec. 2.3.1.

[3] Infact the $1/(N-1)$ Bessel correction removes all of the bias in the estimate of the population variance and most of the bias in the estimate of the population standard deviation, it only removes all of it for the normal distribution.

[4] We will cover estimates in more detail later in Sec. 3.1.

The r^{th} *central moment* of x is defined as

$$c^r = \frac{1}{N}\sum_i^N (x_i - \bar{x})^r \,. \tag{1.18}$$

Consequently, we can see that the *arithmetic mean* is also the 1st moment, and the variance is the 2nd central moment.

There are specific names given to quantities related to the third and fourth central moments, which are the *skew* and the *curtosis*. The *skew* is defined as

$$\gamma = \frac{1}{N\sigma^3}\sum_i^N (x_i - \bar{x})^3 \,. \tag{1.19}$$

This is a measure of how asymmetric a distribution is. The σ^3 in the denominator ensures the *skew* is a dimensionless quantity. If it symmetrically distributed about the mean then it has a skew of zero. If more of the distribution lies on the high-side (right-hand-side) of the mean then it is *postively skew*, if more lies on the low-side it is *negatively skew*.

The *curtosis* is defined as

$$k = \frac{1}{N\sigma^4}\left[\sum_i^N (x_i - \bar{x})^4\right] - 3. \tag{1.20}$$

The σ^4 in the denominator ensure k is dimensionless and the shift of -3 means that the *curtosis* is zero for a Gaussian distribution, which is said to be *mesokurtic*. A postive *curtosis* implies a distribution with a narrower (taller) core (or peak) and wider tails, this is said to be *leptokurtic*. A negative *curtosis* implies a broad peak and short tails, this is said to be *platykurtic*.

Some illustrations of various distributions to visually demonstrate the *mean*, *standard deviation*, *skew* and *curtosis* are provided in Fig. 1.2. Note that absolutely no knowledge of the underlying distribution is required to make sample measurements of the *moments*. The code that produces these is:

```python
import numpy as np
from scipy.stats import norm, maxwell, laplace
from scipy.stats import skew, kurtosis
import matplotlib.pyplot as plt

# generate a normal distribution
# centered at 0 with width 1
x1 = norm.rvs(0,1,size=50000)

# now a distribution with positive skew
x2 = maxwell.rvs(-2.4,1.5, size=50000)

# now a distribution with positive curtosis
x3 = laplace.rvs(0,1, size=50000)

# draw them
fig, axes = plt.subplots(1,3, figsize=(12,3))
for x, ax in zip([x1,x2,x3],axes):
    ax.hist( x, bins=50, alpha=0.5, range=(-3.5,3.5) )
    ax.text( 0.75, 0.92, r'$\bar{x} = '+f'{np.mean(x): 4.2f}$', transform=ax.transAxes )
    ax.text( 0.75, 0.84, r'$\sigma = '+f'{np.std(x): 4.2f}$', transform=ax.transAxes )
    ax.text( 0.75, 0.76, r'$\gamma = '+f'{skew(x): 4.2f}$', transform=ax.transAxes )
    ax.text( 0.75, 0.68, r'$k = '+f'{kurtosis(x): 4.2f}$', transform=ax.transAxes )
    ax.set_xlabel('$x$')
axes[0].set_title('Zero skew and zero kurtosis')
axes[1].set_title('Positive skew')
axes[2].set_title('Postitive kurtosis')
fig.tight_layout()
plt.show()
```

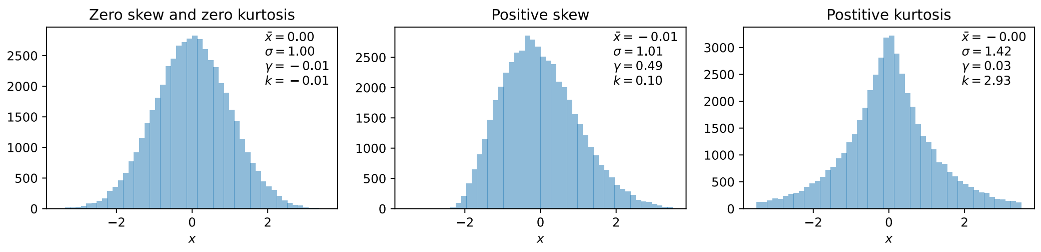


Figure 1.2: A visual demonstration of a few distributions with zero and non-zero skew and curtosis. Left: a normal (Gaussian) distributed sample with mean of zero, standard deviation of one, skew and curtosis of zero. Middle: a Maxwell distributed sample with postitive skew. Right: a Laplace distributed sample with postitive curtosis.

.

1.3 Covariance and correlation

More often than not our dataset contains information on more than one variable. The example given in Fig. 1.1 shows a comparison between two. A very useful quantity to ascertain is the covariance (or correlation) which tells us about the dependence or relationship between different variables. Imagine a dataset that contains just two variables, x and y. The dataset, of size N, will consist of pairs of numbers,

$$\{(x_1, y_1), (x_2, y_2), (x_3, y_3), ..., (x_N, y_N)\}.$$

We can of course find the means, $\bar{x}$ and $\bar{y}$, using (1.1), the variances, $V(x)$ and $V(y)$, using (1.6), and the standard deviations, σ_x and σ_y, using (1.11). The covariance between x and y is defined as

$$\begin{aligned}
\mathrm{cov}(x, y) &= \frac{1}{N}\sum_i^N (x_i - \bar{x})(y_i - \bar{y}) && (1.21)\\
&= \frac{1}{N}\sum_i^N (x_i y_i - y_i\bar{x} - x_i\bar{y} + \bar{x}\bar{y}) && (1.22)\\
&= \overline{xy} - \bar{x}\bar{y}. && (1.23)
\end{aligned}$$

Therefore if values of x above the average, $\bar{x}$, also occur when values of y are above the average, $\bar{y}$, then the *covariance* is positive. If the opposite is true then the *covariance* is negative. If above average values of x occur just as often for above or below average values of y then the *covariance* is zero, or close to zero.

It is worth noticing that the covariance is simply just a generalisation of the variance as $\text{cov}(x, x) = V(x)$ and $\text{cov}(y, y) = V(y)$. We should also notice that for a dataset that contains M different variables the covariance can be written as a symmetric matrix, with dimensions $M \times M$, with a covariance between each pair of variables. The diagonal elements of this matrix will simply be the variances. Thus, we can write the covariance in matrix notation as

$$V_{xy} = \text{cov}(x, y) = \overline{xy} - \bar{x}\bar{y} \tag{1.24}$$

$$= \frac{1}{N}\sum_{i}^{N}(x_i - \bar{x})(y_i - \bar{y}). \tag{1.25}$$

Please be aware that the letters x and y in the notation above index the dimension (it is not restricted to just two dimensions x and y but infiniety many) whereas the letter i indexes the event or element in the sample.

The *covariance* has dimensions and so consqeuently can be a bit diffiult to inuitively intepret. A much easier to understand and related quantity, which is dimensionless, and can only have values between $\{-1, 1\}$, is the *correlation*, normally labelled, ρ. It it simply the covariance normalised by the standard deviation product so that,

$$\rho(x, y) = \frac{\text{cov}(x, y)}{\sigma_x \sigma_y} = \frac{\overline{xy} - \bar{x}\bar{y}}{\sigma_x \sigma_y}. \tag{1.26}$$

Similarly for a multi-dimensional space the *correlation matrix* is

$$\rho_{xy} = \frac{V_{xy}}{\sigma_x \sigma_y}, \tag{1.27}$$

and thus the covariance can be conveniently written as

$$V_{xy} = \rho_{xy}\sigma_x\sigma_y. \tag{1.28}$$

The correlation matrix elements must lie between -1 and $+1$ and the diagonal elements are all $+1$. Understanding correlations in our data is a very important aspect of *machine learning* as we will now discuss in the next section. Notice, for example, that the blue points in Fig. 1.1 generally show a *postitive correlation* between x and y, whilst the orange points show a *negative correlation*. A *machine learning* algorithm can exploit the difference in these correlations when attempting to build a classifer which distinguishes between the blue and orange.

1.4 Learning from data

We want to use our sample to learn and potentially predict properties of the population. Further details about how we can do this are given later in this module and in other modules of this course. In this section we will simply discuss a few thoughts on how we can qualitatively assess our sample dataset and how we can best exploit it for our desired purpose. This section covers a few "food-for-thought" topics related to learning from data. It is meant to simply give you an overview regarding datasets, the details you will learn throughout the year.

Modern day computing power allows us to deploy machine learning techniques to fit datasets, estimate their probability distributions, classify datasets and make predictions about trends in our datasets (regression). We often make use of empirical techniques for these which means we can avoid assumptions about the underlying "true" distribution, we simply *learn* it from the data.

1.4.1 Typical structure of data

Datasets typically contain *variables* or *features* on one axis, and *events* or *rows* on the other axis. A widely-used and convenient representation of this is in a `pandas` `DataFrame`. The columns of the dataframe represent the *features* and the rows represent the *events* which are normally i.i.d..

When using python you will most often manipulate data using `numpy` arrays and `pandas` dataframes so I suggest you get to grips with them if you haven't already. You are of course free to use whichever packages you wish and do not have to use these. In the lectures I show an example Higgs boson dataset.

1.4.2 Exploiting correlations in data

The vast quanity of data available in most areas of science, and the ever advancing ability of our computers to process this data, means we are able to exploit more *features* of our data. In machine learning (ML) a *feature* relates to some variable in our dataset. Typically ML algorithms attempt to fit some empirical function based on a set of features in the data. Let's pick out a set of features in our sample, $\vec{x}$, I use the vector notation to demonstrate that $\vec{x}$ is multi-dimensional (*i.e.* could be the heights, weights, ages, income, *etc.* of some people) and it contains many, say N, i.i.d. events:

$$\vec{x}_i = \{\vec{x}_1, \vec{x}_2, \vec{x}_3, \ldots, \vec{x}_N\}. \tag{1.29}$$

We then want to train the ML algorithm so that it learns the form of some function, f, in order to produce an output y:

$$y_i = f(\vec{x}_i). \tag{1.30}$$

The flexibility of ML, aided by the power of modern computing, means that we can trial and exploit various different feature sets and do not need to necessarily make assumptions about their distributions.

Note, the convention in python when coding the feature set and output, is to label the feature array with a capital "X" (which will be a 2-dimensional array with shape `(nevents, nfeatures)`) and the output with a lower-case "y" (which will normally be a 1-dimensional array with shape `(nevents,)`).

Let's think about this in the context of a classification example. For ease of demonstration I will use just two *features* (or *variables*) in this example but it applies equally to any number and the potential increase in performance scales rapidly (of course so does the computation power required). I will generate a dataset with two features from two different categories (imaginatively called "signal" and "background"). Our goal as data analysts is to build some algorithm which classifies, or distinguishes between, them. The two classes are closely overlapping but contain *very different correlations.*

First up let's generate our dataset and inspect it:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

Ns = 5000
s_mu = [1,1]
s_cov = [[4,1],[1,4]]

Nb = 5000
b_mu = [-1,-1]
b_cov = [[4,-1],[-1,4]]

sx, sy = np.random.multivariate_normal(s_mu, s_cov, size=Ns).T
bx, by = np.random.multivariate_normal(b_mu, b_cov, size=Nb).T

df = pd.DataFrame({
    'x': np.concatenate((sx,bx)),
    'y': np.concatenate((sy,by)),
    'C': ['signal']*Ns + ['background']*Nb
})

splot = sns.jointplot( df, x='x', y='y', hue='C', s=2, alpha=0.8, space=0,
    zorder=20 )
```

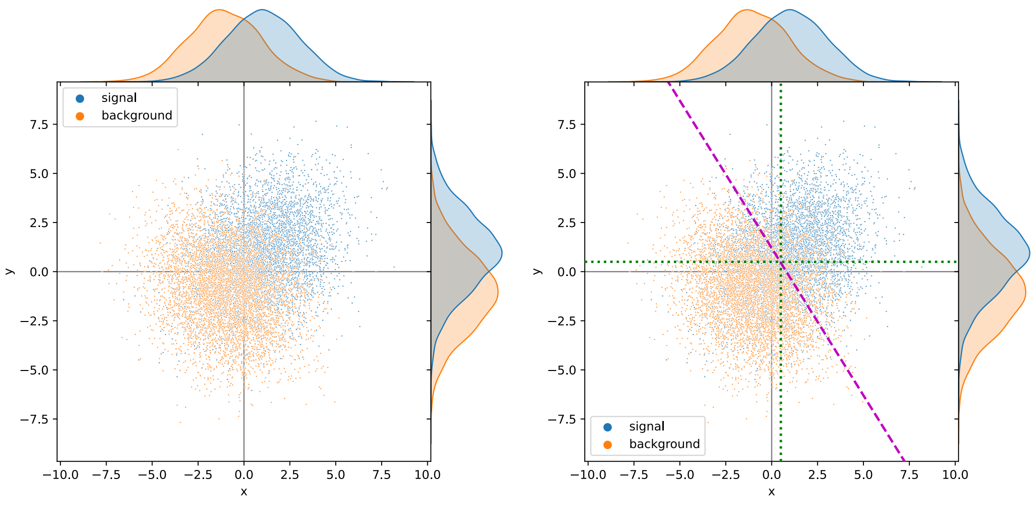


Figure 1.3: A visualisation of our dummy data (left) with possible *rectangular* and *variable* cut boundaries overlaid (right).

If we wanted to seperate the blue and orange we could maybe decide to define some rectangular selection or cuts, depicted by the green dotted lines on the right of Fig. 1.3. We would then classify signal as anything that is to the right of the vertical dotted line and also above the horizontal dotted line. These kind of selections are known as *rectangular* cuts. On inspection of this particular dataset we may decide that a better cut would be a diagonal one (following a $y = -mx + c$ line) depicted by the magenta dashed line. We classify signal as anything to the top-right of the dashed line and the rest as background. This is sometimes known as a *variable* cut. It is rather trivial in the two-dimensional case but imagine how complex this becomes when the feature set is large. To extend this further we could define a more optimal cut boundary which does not follow a straight line but something more of higher-order, this becomes more and more complex as more features are added and does not work at all if their are disjoint regions in our classification problem.

ML algorithms can entirely avoid these difficulties by learning a non-parametric form of the optimal classification boundary. They are particularly good at exploiting correlations between features in datasets and many are also good at ignoring features that are unimportant. Below (Fig. 1.4) I show a decision boundary found by training a simple random forest, using the `scikit-learn` `RandomForestClassifier`. The boundary is not entirely smooth, although with suitable tuning of the parameters I probably could have made it so. The beauty is this works in any number of dimensions (of either inputs or outputs) and can pick out disjoint regions as well.

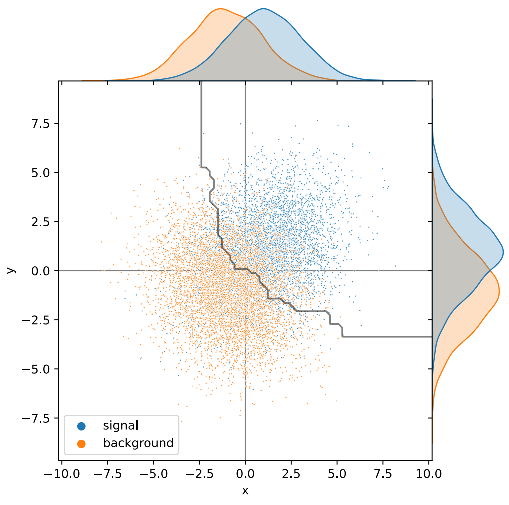


Figure 1.4: The decision boundary found by a simple ML algorithm

The remaining decision still left to us is *where* to place the boundary, not just what it's shape should be. Given the basic shape of the boundary in the Fig. 1.4, I could move further to the top right and reduce my efficiency to classify signal but at the same time increase the rate at which I reject background. Conversely, I might prefer to move the boundary to the bottom left and increase by signal hit rate but suffer a higher background rate. Consequently, we need to define various different criteria to quantify these decisions, which is the topic of the next short section.

1.4.3 Performance criteria and metrics

In order to quantify the performance of ML algorithms (or indeed any classification or regression algorithm) we need to define some criteria and metrics by which we can assess them. There are various different choices of metric we can use and we then have freedom in how we choose which is the most optimal "cut" or "classification" point for our means. A few of the common ML evaluation metrics are described below:

- **True positive (TP)**. Correct prediction of positive or signal outcome.
- **False positive (FP)**. Incorrect prediction of positive or signal outcome.
- **True negative (TN)**. Correct prediction of negative or background outcome.
- **False negative (FN)**. Incorrect prediction of negative or background outcome.
- All positive or signal events are given by $P = TP + FN$.
- All negative or background events are given by $N = TN + FP$.
- All events classified as positive or signal-like are given by $C_P = TP + FP$.
- All events classified as negative or signal-like are given by $C_N = TN + FN$.
- **Recall: true positive rate**. Sometimes also called *signal efficiency* or *sensitivity.* The fraction of all positive or signal events that are correctly classified:
$$TPR = \frac{TP}{TP+FN}. \tag{1.31}$$
- **Specificity: true negative rate**. Sometimes also called *background efficiency.* The fraction of all negative or background events that are correctly classified:
$$TNR = \frac{TN}{TN+FP}. \tag{1.32}$$
- **False positive rate**. The fraction of all negative or background events that incorrectly classified:
$$FPR = \frac{FP}{FP+TN}. \tag{1.33}$$
- **False negative rate**. The fraction of all positive or signal events that incorrectly classified:
$$FNR = \frac{FN}{FN+TP}. \tag{1.34}$$
- Note the relationships between TPR and FNR, as well as TNR and FPR:
$$TPR + FNR = \frac{TP}{TP+FN} + \frac{FN}{FN+TP} = \frac{TP+FN}{TP+FN} = 1, \tag{1.35}$$
$$TNR + FPR = \frac{TN}{TN+FP} + \frac{FN}{FP+TN} = \frac{TN+FP}{TN+FP} = 1, \tag{1.36}$$
so that $FPR = 1 - TNR$, $FNR = 1 - TPR$.
- **Precision**. The fraction of all positively classified events that are correctly classified:
$$p = \frac{TP}{FP+TP}. \tag{1.37}$$

- **Accuracy**. The fraction of all events that correctly classified:

$$\alpha = \frac{TP + TN}{P + N}. \tag{1.38}$$

- **Error rate**. The fraction of all events that are incorrectly classified:

$$\varepsilon = \frac{FP + FN}{P + N}. \tag{1.39}$$

- **Purity**. The fraction of all events classified positively that are correctly classified:

$$\rho_P = \frac{TP}{TP + FP} \quad \text{and} \quad \rho_N = \frac{TN}{TN + FN}. \tag{1.40}$$

- **Significance**. For a counting experiment quantifies the statistical significance:

$$\sigma = \frac{TP}{\sqrt{TP + FP}}. \tag{1.41}$$

- **Signal-to-noise**, sometimes also called *signal-to-background ratio*:

$$SNR = \frac{TP}{FP}. \tag{1.42}$$

- **F-score**, sometimes also called *F-measure*:

$$F = 2\frac{\text{precision} \times \text{recall}}{\text{precisoin} + \text{recall}} = \frac{2TPR}{2TPR + FPR + FNR}. \tag{1.43}$$

Different choices of metric are better suited to different problems. Remember we have a choice in where we place the classification criteria. It is straight-forward to maximise the TPR but the down-side is that you will also have a very high FPR. This is also problem specific. For example when developing Covid-19 tests you may decide it is preferable to have a very low FNR - you do not want people who actually have the disease testing negative and circulating in society. The price to pay is that you will end up with a higher FPR - but it seems more acceptable to tell people who actually don't have the disease that they do. We will come back to discuss these so called Type-I and Type-II errors more when we talk about hypothesis testing in Sec. 3.8.

A common choice for choosing the optimum point in classification problems is to use the so called reciever-operating-chacteristic (ROC). This is a function of the FPR *vs.* the TPR, for which an example is shown below in Fig. 1.5. A pure guess will give you a straigh-line in the ROC curve. For better results you want to push the curve to the top-left. Consequently a good metric to optimise is the so-called area-under-the-ROC (AUC).

1.4.4 Data challeneges

In many fields of data analysis the data is abundant. Quite often so large we have difficulty processing it, or figuring out what parts of it are relevant for our analysis (which you could argue amounts to the same thing). This is why this broad spectrum of different fields are colloquially referred to as *big data*.

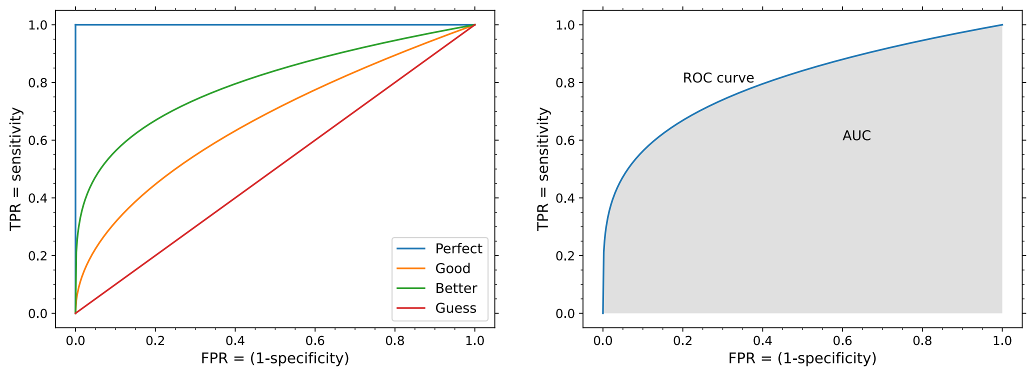


Figure 1.5: Some example ROC curves.

Quantity

Quantity can refer to both the size of the sample itself (*i.e.* the number of i.i.d. events) and the number of features or variables within the dataset. In many use cases the individual i.i.d. events can also be arrays themselves, of fixed or variable length. For example when processing images,

- each image is an *event* or a *row* in the dataframe,
- but each image contains many pixels, so each pixel has to be stored in some array (probably a two-dimensional grid or matrix). These are not i.i.d. because the context of one pixel will be correlated with the content of adjacent pixels. Providing the images in the dataset are all of the same size then the length of these arrays is fixed.
- the *features* in this case could be the {r,g,b} values in each pixture.

As another example when processing language, our input events may be sentences which each contain a variable number of words.

A larger number of i.i.d. events is typically a good thing. If we do not have the processing power or storage capacity to handle them all we can simply take a sub-sample from the sample we have, although doing this in an optimal way can be a big challenge. For example at the Large Hadron Collider beauty experiment, there are approximately 30 million collisions per second (30 MHz) and the raw size of each collision event is $\sim$ 125 kB. This gives a readout rate of 5 TB/s, which is simply impractical to write to disk. Instead we have to develop a “trigger” system which reduces this rate to $\sim$ 10 GB/s whilst trying to keep as many of the events of interest as possible.

A huge number of variables or features is often simply inconvenient as many of them are not relevant, but of course we often don’t know *a priori* which ones are and which ones aren’t. We have to curate feature sets to keep sample sizes and algorithm complexity manageable. Often we can also find or implement *derived* features which encapsulate the most relevant information, exploit symmetries in our data and reduce the number of features. An example of this could be that to classify different particle decays I do not necessarily need to have (x, y, z) coordinates for the particle flight path but simply need the total distance, $r = \sqrt{x^2 + y^2 + z^2}$.

Quality

There are certain considerations regarding data *quality* as well. All dataset will contain some level of noise, due to the random nature of how data is collected (or sampled). This is part and parcel of dealing with statistical models. However, we can often improve our algorithms if we understand the distribution and in some cases the source of the noise.

Below I list various aspects and examples that can degrade the quality of our data. Many of these can be overcome if we *understand* their nature. Some of them are unavoidable.

- Data collected with different apparatus. In this case the *resolution* for each event is not the same. This is incredibly common in practical scientific studies. For example I may have a large dataset of MRI images of some organ or other but these may be taken with a wide variety of different MRI scanners. Each will all have slightly different design and specifications, and different performance.

- Missing data. Sometimes we have events which are not *complete*. For example it could be a set of blank or masked pixels in an individual image (which is different for each image in the dataset), or it could be that one response on a survey is left blank, or for what-ever reason some one feature of the data could be no recorded for one event.

- Missing or hidden variables. We will see later in the course (Sec. 4.5) how in some cases these can be specifically dealt with. These are cases where we are testing for or expect some dependence on a variable that we can imagine but can not necessarily directly measure. An example could be attempting to measure "happiness" in a population. Of course you could ask them but that requires actually doing so, them telling you the truth in a unbiased and calibrated way and does not allow you to predict happiness in others. Instead you could build a dataset including various wealth, health, education, geography, environment *etc.* metrics and then attempt to learn the dependence of happiness on these.

- Biases in the data or data collection procedure. Some datasets include biased features. This could originate from detection equipment that has not been properly calibrated or for example from survey particpants that are not telling the truth.

Improving the quality of data is difficult to do after it has been collected. You can attempt to calibrate it *a posteriori* but that requires you where fore-sighted enough to collect a calibration sample. You can also attempt to cherry pick data of the best quality but this itself can lead to substantial biases dependent on the cherry picking procedure.

Another way the quality of our data sample can be degraded is if it does not sufficiently *represent* the population. That is the topic of the next section.

Representativeness

Representativeness is a rather vague concept with no real formal definition. Clearly we desire that our data sample is representative of the population. Quite often this is not the case so we require some procedure to create a new sample that is representative. This technique is known as *stratified sampling*.

Let's say we have a large sample of student records at this university and we want to use if to understand and predict certain trends. Whenever we want to train an algorithm we typically split our sample up into different subsamples (see Sec. 1.4.6). We can do this in a totally random way, which is known as *random sampling*, but when we do so we inherit any biases or unrepresentativeness in our original sample. One of the problems we might encounter is that some of the trends we want to learn depends on factors like age, race, gender, family income *etc.* and we don't want to bias our learning of any trends based on the fractions of each characteristic in our training sample. The solution is to split up our sample into mutually exclusive sub-samples, called *strata*, and then randomly sample from the *strata* so that our final sample contains an appropriate representation. This is graphically depicted in Fig. 1.6.

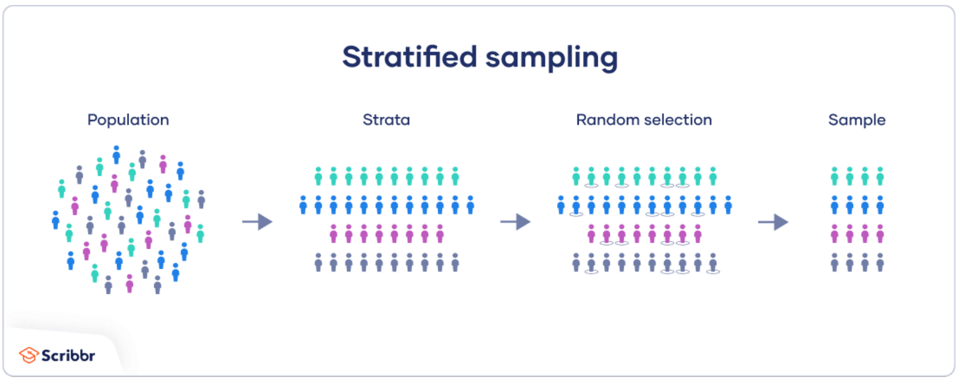


Figure 1.6: A depication of the stratified sampling method. Image from Ref. [11].

For another illuminating example I will use my own field once again. In collision events at the Large Hadron Collider beauty experiment the data taking conditions vary considerably over time. The conditions of the beam, position of the detector and various other factors are particularly different in each data taking year. Data has been collected in the years, {2011, 2012, 2015, 2016, 2017, 2018}, each in various different amounts (mostly dependent on the luminosity of the beam and the run time). We produce simulations of these conditions for each different year, however the simulation samples are typically of a fixed size, *e.g.* 1M events in 2011, 2M events in 2012 *etc.*. Consequently if we want to produce a simulation sample which is *representative* of the entire data taking period, we sample the appropriate fraction from each year of simulation such that it matches the amount collected in data.

Over / under-fitting

Recall from (1.30) that we want to learn the form of some function f which depends on our feature set $\vec{x}$ and produces some output y. The function that we learn is typically non-parametric, *i.e.* it is not some smooth function like $y = \cos(x)$ but depends on a series of `if` / `else` statements at the nodes of our network or branches of our decision tree. This can be parametrised by a very large series of smooth terms with different weights. At each of these nodes there will be some free parameters, typically a weight w and a bias b. Thus our entire trained model is a function of these weights and biases as well as the feature set $\vec{x}$. The process of training the model is to find the "best" values for w and b, *i.e.* those that minimise our loss function and produce the best function f we can find. Once we have trained the model these parameters are fixed. Thus a perhaps more accurate representation of our ML model would be

$$y_i = f(\vec{x}_i; \vec{w}, \vec{b}), \tag{1.44}$$

where $\vec{w}$ and $\vec{b}$ are the weights and biases respectively, and are fixed once the model is trained.

When training a model we have to be wary of either *over-fitting* or *under-fitting* our model. If we give the model too much freedom then it will *fit* to statistical fluctuations in our training sample. For example if there is as many weights and bias parameters as there are events in the sample then the model will be "perfect" for that sample. If there is only a single weight and single bias then it won't be much better than a guess.

Both underfitting and overfitting will result in poor performance on a sample that has not been seen before. Consqeuently we test for overfitting by splitting our original sample up into a sample for training and a sample for testing, more on that below (Sec. 1.4.6). Some demonstrations are given below in Fig. 1.7.

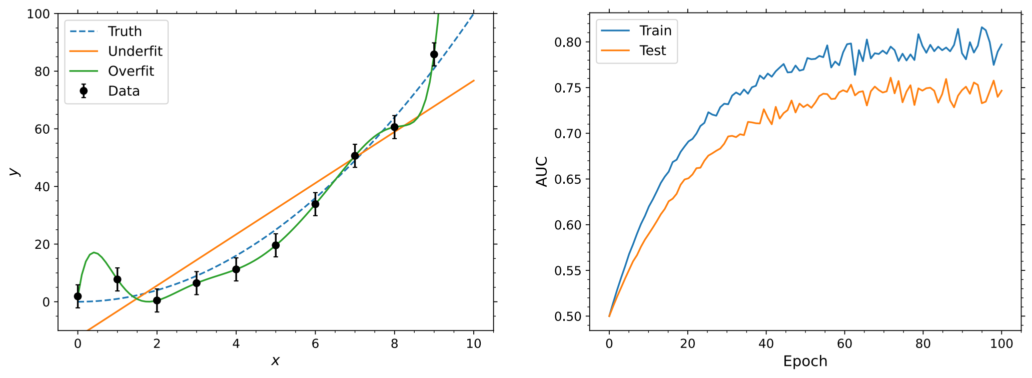


Figure 1.7: Left: an example of a simple polynomail regression showing an underfit and an overfit. Right: an example you might see in real life of an overtrained network. The plot shows the AUC score as a function of traniing epochs. You can see the performance is much better on the training sample than the test sample, suggesting the model is overfitted to the training sample.

1.4.5 Hyperparameter tuning

When training our machine learning model we pick optimal values for the weight and bias parameters. It will become clearer later in the course what optimal really means, but for now it is just the parameter values which maximises or minimises some function. You can think of this as maximising the AUC score from above. However, ML algorithms also depend on various other parameters, which are collectively known as the *hyper-parameters* of the model. For example, if training a decision tree, we have freedom to choose how many branches are in the tree and what the depth of the tree is. When training a neural network, we can choose how many nodes there are, how many layers there are, how many events the network sees at any one time, how the weights and biases are propagated through the network *etc.*. The optimal choice of hyperparamaters very much depends on the model being learnt and the size and complexity of the dataset it is being trained on. There is no sure-fire way to find the best hyperparameters other than a brute force grid-search. Depending on the problem, and on the model, you can get a huge boost in performance by finding better hyperparameters, so it is well worth looking. In order to avoid biases from this procedure we typically split our original sample into a training sample and a validation sample, more below (Sec. 1.4.6), where the hyperparameters are optimised by assessing the performance on the validation sample. An example of the sort of plot you might see comparing hyperparameter tuning is the ROC curve, *e.g.* Fig. 1.8.

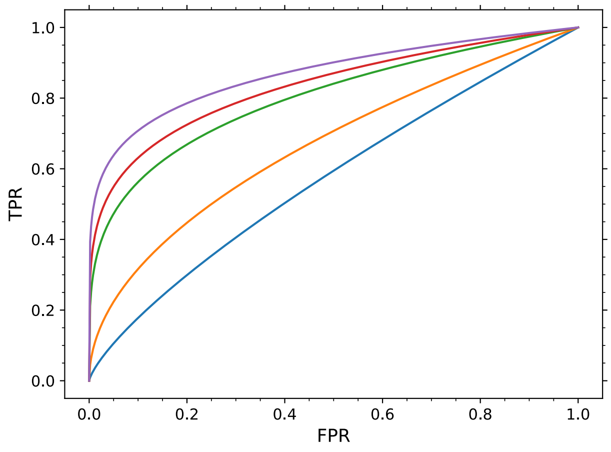


Figure 1.8: An example ROC curve showing potential performance of the same network with different hyperparameters.

1.4.6 Training, testing and validation samples

In order to avoid biases from *overfitting* and *hyperparameter tuning* the normal procedure in machine learning is to split the sample into subset for training, validation and testing. This split does not have to be uniform (you often want to keep a larger sample for training) but should of course follow the same sampling procedure, *i.e.* if stratified sampling is used it should be the same in each.

- **The training sample** is the one used to train or fit the model.
- **The validation sample** is the one use to optimise the hyperparameters.
- **The test sample** is the one used to evaluate the final performance of the model.

Most of the ML libraries include built-in methods for splitting samples up and it is convenient to use the same splitting each time. An example is shown below using `scikit-learn`.

```python
from sklearn.model_selection import train_test_split

# entire sample has 10000 events
# want to train on 4 features

feature_set = ['Length','Weight','Fin length', 'Fin span']

# X shape is (10000,4)
X = df[feature_set].to_numpy()

# target output, y, shape is (10000,)
```

```python
y = df['Is a dolphin'].to_numpy()

# split into training, test and validation samples
train_frac = 0.75
test_frac  = 0.10
val_frac   = 0.15

# get training sample - 75% of original
X_train, X_temp, y_train, y_temp = train_test_split( X, y, test_size = 1 -
    train_frac )

# split temp into test - 10% and val - 15%
X_val, X_test, y_val, y_test = train_test_split( X_temp, y_temp, test_size =
    test_frac/(test_frac + val_frac) )
```

1.4.7 Cross validation

A very nice, and commonly used method, for overcoming both issues with overfitting and the fact we have to reduce our training sample to accomodate testing and validation is known as *cross-validation*. Under this method we split our sample up into k equal "folds". This then gives us k testing samples and we can train k algorithms on the remaining sample not in k. For example let's say $k = 4$ and we randomly split our sample into 4 folds, called A, B, C and D. We would then train 4 different algorithms, with all hyperparameters kept the same, using in the first fold, A as the test / validation sample and all of B, C, D as the training sample. We then repeat with B as the test / validation sample and A, C and D as the training sample, and so on. This means that we can increase our training sample size, it means every single event in the sample is seen by the majority of the algorithms and it means that no event is evaluated (tested or validated) with an algorithm on which it was used to train. It gives us independent samples with which to check for overtraining. A pictoral example of k-fold cross validation is shown in Fig. 1.9.

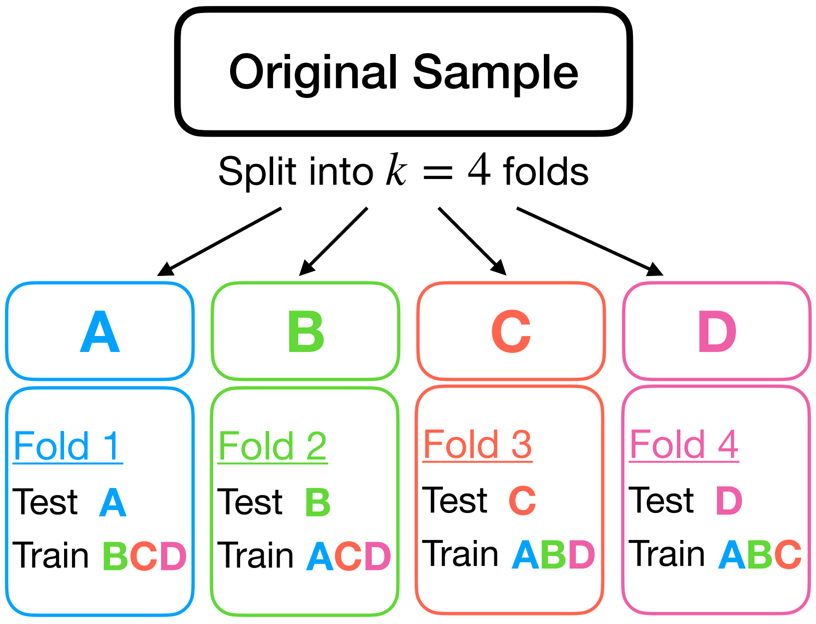


Figure 1.9: A representation of k-fold cross validation.