

IB Foundations of Data Science

Damon Wischik, Computer Laboratory, Cambridge University

Michaelmas Term 2017

- There will be 12 lectures.
- There will be three example sheets, combining pen-and-paper work with practical work. The material covered in the practicals may be tested in the exam, but the practicals themselves won't be graded. Example sheets will be handed out in advance of the material they cover.
- There will be four practical help sessions. These are optional.

ACKNOWLEDGEMENTS

Thanks to Jakub Perlin, Richard Gibbens, Thomas Sauerwald, and A. Student for pointing out mistakes in the notes.

0. What is data science?

The Harvard Business Review called data science “the sexiest job of the 21st century”¹, and the Economist says “The world’s most valuable resource is no longer oil, but data”². So it is no surprise that *data science* is a label that many people have seized upon, to mean many different things. Here is my attempt at a definition:

Data science is any field of study where you can be surprised by data.

What do you think data science is? Note down your answer now, and again at the end of the course.

0.1. ‘Surprised by data’: reasoning about uncertainty

Here are some results from a survey of undergraduates, 15 female, 94 male, 4 other / didn’t specify.

		F	M	X
I am treated fairly in lectures	yes	14	86	.
	no	1	8	.
I’m comfortable asking questions	str.agree	2	36	.
	str.disagree	10	36	.

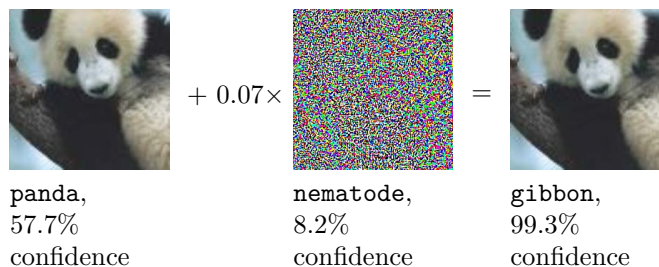
Simple percentages say that female students are more likely to say they’re treated fairly in lectures than male students, and yet they’re less comfortable asking questions, which seems surprising—but when we look at actual numbers rather than percentages we intuit that the numbers are so small we might expect some mixed signals. Is this intuition correct? In other words, what counts as surprising, and what counts as chance variation?

Data science is about fields of study where you can be surprised by data. To say what’s surprising, we need to be able to *reason about uncertainty*. Here is a quote from an astronomer, about possible detection of an exomoon, which illustrates that reasoning about uncertainty does not come naturally:

The work by Dr Kipping, his Columbia colleague Alex Teachey and citizen scientist Allan R Schmitt, assigns a confidence level of four sigma to the signal from the distant planetary system. The confidence level describes how unlikely it is that an experimental result is simply down to chance. If you express it in terms of tossing a coin, it’s equivalent to tossing 15 heads in row.

But Dr Kipping said this is not the best way to gauge the potential detection. He told BBC News: “We’re excited about it... statistically, formally, it’s a very high probability. But do we really trust the statistics? That’s something unquantifiable. Until we get the measurements from Hubble, it may as well be 50–50 in my mind.”³

Today’s neural networks for image classification also have trouble reasoning about uncertainty, as this adversarial panda illustrates:⁴



The main goal of this course is to learn how to express uncertainty in equations and computer programs, so that ‘what you know in your mind’ and ‘what your probability computations say’ support each other.

¹<https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century>

²<https://www.economist.com/news/leaders/21721656-data-economy-demands-new-approach-antitrust-rules-worlds-most-valuable-resource>

³<http://www.bbc.co.uk/news/science-environment-40741545>

⁴I. J. Goodfellow, J. Shlens, and C. Szegedy. “Explaining and Harnessing Adversarial Examples”. In: *ArXiv e-prints* (Dec. 2014). arXiv: 1412.6572 [stat.ML].

0.2. ‘Field of study’: scientific modeling

Given a crime and policing dataset⁵, here are some questions that spring to mind:

1. What is the typical crime rate in each neighbourhood?
Keywords: description, estimation.
2. How many police officers should we allocate to each neighbourhood this week?
Keywords: prediction accuracy, working system.
3. Do our policing strategies exacerbate racial tension?
Keywords: science, hypothesis, policy, counterfactual.

You’ll sometimes see *machine learning* used to mean “an operational engineering discipline, for designing algorithms that predict outputs when fed with inputs, and are evaluated according to the accuracy of their predictions”. This is the spirit, for example, of competitions at [kaggle.com](https://www.kaggle.com), and it has led to some remarkably clever algorithms. It also leads to offensive mistakes:

Google came under fire this week after its new Photos app categorized photos in one of the most racist ways possible. On June 28th [2015], computer programmer Jacky Alciné found that the feature kept tagging pictures of him and his girlfriend as “gorillas.” He tweeted at Google asking what kind of sample images the company had used that would allow such a terrible mistake to happen.

*Google’s chief social architect Yonatan Zunger responded quickly, apologizing for the feature.*⁶

A machine learning system isn’t value neutral, it’s a reflection of the choices that went into the training dataset. Data investigation is a required skill for responsible machine learning. Conversely, whenever you analyse data, the tools you use to test your hypotheses are estimation and prediction. So ‘data science’ and ‘machine learning’ are complementary and intertwined.

This course is about data *science*, i.e. about building models and hypotheses and understanding, using data. It’s *not* an introduction to the machine learning toolbox. The range of algorithms out there is exciting and growing rapidly. There are courses in Part II and III that will introduce you to some of them, and there are others that you will pick up in your own reading. The goal of this course is to give you practice at asking the right questions and finding the right tools, so that when you read about a new algorithm you can quickly pick it up and decide where it is and isn’t appropriate.

What is data science modeling?

All models are wrong but some are useful [...] there is no need to ask the question “Is the model true?”. If “truth” is to be the “whole truth” the answer must be “No”. The only question of interest is “Is the model illuminating and useful?”.⁷

Since no model is to be believed in, no optimization for a single model can offer more than distant guidance. What is needed, and is never more than approximately at hand, is guidance about what to do in a sequence of ever more realistic situations. The analyst of data is lucky if he has some insight into a few terms of this sequence, particularly those not yet mathematized. [...] The main tasks of pictures are then: to reveal the unexpected, to make the complex easier to perceive. Either may be effective for that which is important above all: suggesting the next step in analysis, or offering the next insight. In doing either of these there is much room for mathematics and novelty.⁸

⁵e.g. <https://data.police.uk/data/>

⁶<https://www.theverge.com/2015/7/1/8880363/google-apologizes-photos-app-tags-two-black-people-gorillas>

⁷G. E. P. Box. “Robustness in the Strategy of Scientific Model Building”. In: *Robustness in Statistics*. Vol. 1. May 1979, p. 40. URL: <http://www.dtic.mil/docs/citations/ADA070213>.

⁸John W Tukey. “Mathematics and the picturing of data”. In: *Proceedings of the international congress of mathematicians*. Vol. 2. 1975, pp. 523–531. URL: <http://www.mathunion.org/ICM/ICM1974.2/Main/icm1974.2.0523.0532.ocr.pdf>.

You've got to have models in your head. And you've got to array your experience—both vicarious and direct—on this latticework of models. You may have noticed students who just try to remember and pound back what is remembered. Well, they fail in school and in life. You've got to hang experience on a latticework of models in your head.

What are the models? Well, the first rule is that you've got to have multiple models—because if you just have one or two that you're using, the nature of human psychology is such that you'll torture reality so that it fits your models, or at least you'll think it does.⁹

0.3. The foundations

The foundations of data science are probability, computing, and statistics.

You can learn enough probability theory in a term, though it takes practice practice practice. You'll pick up the relevant computing skills over the course of your degree: you need to be able to think algorithmically and write fast code, and to understand databases and distributed systems for big data. You can pick up some statistics ideas in a term, but to really understand what it means to learn from data you will need a lifetime of experience, including either a stint in a startup or a degree in philosophy.

Warren Buffet's business partner says in colourful language how important it is to get practice at working with probability:

If you don't get this elementary, but mildly unnatural, mathematics of elementary probability into your repertoire, then you go through a long life like a one-legged man in an ass kicking contest. You're giving a huge advantage to everybody else. One of the advantages of a fellow like Buffett, whom I've worked with all these years, is that he automatically thinks in terms of decision trees and the elementary math of permutations and combinations.¹⁰

This course assumes you know the basic rules for manipulating probability, such as Bayes's rule. We'll use probability for modelling, and we'll ask what we can learn from data via our models. Here's a taster, the naïve Bayes classifier. Suppose we have the data

Emails labeled **spam**: “buy this viagra”, “cheap online pharma”, “cheap viagra today”.
 Emails labeled **genuine**: “will you buy the present or will I”, “I will buy it online today”.
 Test email 1: “buy viagra today”.
 Test email 2: “buy viagra as a present”.

Here's a simple model to start with: Each word in an email is chosen independently, with a probability that depends on the label of the email. In mathematical notation, let θ_w be the probability of word w in spam emails, and ϕ_w be the probability in genuine emails, and let

$$\mathbb{P}(\text{words } w_1 w_2 \dots w_n | \text{spam}) = \prod_i \theta_{w_i},$$

$$\mathbb{P}(\text{words } w_1 w_2 \dots w_n | \text{genuine}) = \prod_i \phi_{w_i}.$$

Based on the labeled data, the obvious parameter estimates are

w	buy	this	viagra	cheap	online	pharma	today	will	you	the	present	or	I	it
θ_w	1/9	1/9	2/9	2/9	1/9	1/9	1/9	0	0	0	0	0	0	0
ϕ_w	2/14	0	0	0	1/14	0	1/14	3/14	1/14	1/14	1/14	1/14	2/14	1/14

According to Bayes's rule,

$$\mathbb{P}(\text{spam} | \text{words}) = \frac{\mathbb{P}(\text{words} | \text{spam}) \mathbb{P}(\text{spam})}{\mathbb{P}(\text{words} | \text{spam}) \mathbb{P}(\text{spam}) + \mathbb{P}(\text{words} | \text{genuine}) \mathbb{P}(\text{genuine})}$$

⁹Charles Munger. *A lesson on elementary, worldly wisdom as it relates to investment management & business*. Speech given at USC Business School. 1994. URL: <http://www.safalniveshak.com/wp-content/uploads/2012/08/Lesson-on-Elementary-Worldly-Wisdom-Charlie-Munger.pdf>.

¹⁰Ibid.

where $\mathbb{P}(\text{spam})$ and $\mathbb{P}(\text{genuine})$ are prior beliefs about the label; prior beliefs might be chosen based on the anticipated statistics of the test document collection, e.g. 50% and 50%. When we evaluate the formula on the test documents,

$$\mathbb{P}(\text{spam}|\text{test email 1}) = 1$$

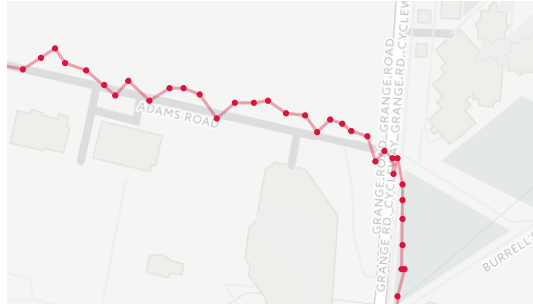
$$\mathbb{P}(\text{spam}|\text{test email 2}) = \text{divide by zero error.}$$

What does the divide by zero error mean? The simple naïve Bayes model might be inaccurate but it's not impossible, so something must have gone wrong with the way we applied it to the data. The statistics component of this course will give you practice at debugging this sort of 'inference bug'.

1. Probabilistic models

Goals. Refresh your memory of IA Maths for NST, where you were taught some basic probability, and practice on some harder questions. Learn about the four major types of probabilistic model, through examples.

Application. Suppose we wanted to write an app to detect if the user is cycling, running, or driving, and which records or assists as appropriate. The GPS readings might look something like this (showing one sample per second).



GPS readings are noisy, so the user's exact location is unknown, but it probably doesn't jerk about as wildly as the GPS readings do. The user is probably cycling given the speed, which suggests how smooth the trajectory is likely to be. If we had a large dataset of traces, we should be able to learn typical GPS noisiness, as well as typical statistics about speed and smoothness for different modes of transport.

We often face these generic issues in data science applications:

- observations are noisy;
- there is hidden state (true position and transport mode) that we'd like to reason about;
- it's a dynamical system we're observing, and if we know how it typically evolves then this tells us something about the hidden state;
- we want to learn system parameters from a large dataset.

In this section we'll look at four common types of model, which showcase these issues. The actual GPS smoothing problem is too involved for lectures, so we'll look at simpler idealized models.

1.1. Random samples

Application. I’ve developed a new load-balancing algorithm for my web server. I want to test my algorithm, by means of simulation. My simulator needs a random number generator (RNG) to generate file sizes, request times etc. The performance of my load balancer will surely depend on the random number generator I use. How should I program this random number generator?

We use RNGs in situations like this because the real world is too complicated to model in a Newtonian cause-and-effect way. We use random numbers to say “There is variability, and I can quantify the degree of variability, but I’m not going to look in excruciating detail for causes for every little variation.” It’s up to the modeler to draw the line between ‘causes of variation that it’s worth including explicitly’ and ‘residual variation that we’ll label noise’.

Typically we take real-world measurements, we look at the data, and we pick a RNG that produces output consistent with the data. Many standard RNGs come with tuneable parameters. Typically we look at the data to estimate what values to use for the tuning parameters (and, in this application, to figure out how the parameters vary with time of day, request type, etc.)

Working definitions. A *random variable* is a function that can give different answers, e.g. a function that calls a random number routine. We say it *takes values in S* to mean that the return value of the function is in the set S . A *random sample* is a random variable that returns a list, in which the individual elements are chosen independently. A *dataset* is a collection of numbers.

Example. Here is a random variable:

```
1 def rgeom(p):
2     x = 1
3     while random.random() > p:
4         x = x + 1
5     return x
```

Let X be the output of `rgeom(1/3)`. To find the distribution of X , we can use simple probability calculation. To get $X = 1$ we need the `random.random() > p` test to fail on the first pass, which has probability p . To get $X = 2$ we need the test to succeed on the first pass then fail on the second, which has probability $(1-p)p$. Generalizing, $\mathbb{P}(X = k) = (1-p)^{k-1}p$.

Example. Let X be the set of birthdays of n people, assuming all days are equally likely, that people are independent, and ignoring leap years. Then X is a random variable, and so is $|X|$. As you saw in IA Maths for NST,

$$\mathbb{P}(\text{all } n \text{ have different birthdays}) = \mathbb{P}(|X| = n) = 1 \times \frac{364}{365} \times \cdots \times \frac{365 - n + 1}{365}.$$

Example. What’s the chance that two or more people present in the first lecture for this course share a birthday?

This is badly put, since it’s describing a dataset not a random variable. Either there is a shared birthday, or there isn’t, so the probability is either 1 or 0. So what on earth did the example on page 1 mean by “panda, 57.7% confidence”? This is a hard question, and we’ll come back to it in Section 3.

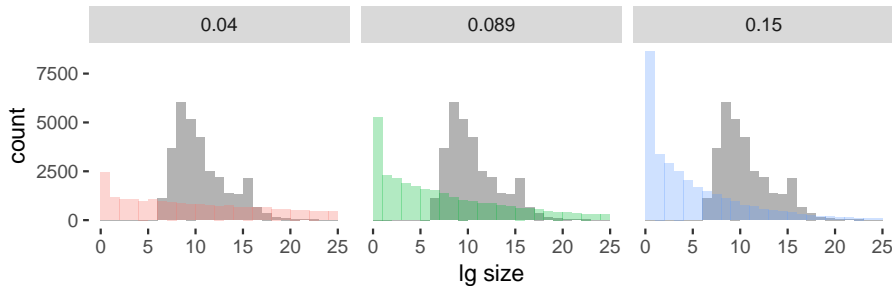
When we write for example “Let X be `rgeom(1/3)`”, remember that X doesn’t *have* any particular value. It’s a stand-in for all the possible values that the random variable might produce, weighted by their probabilities.

* * *

For a random variable that takes parameters, such as `rgeom(p)`, the obvious question is “How should I pick the value of the parameter so that the output of a random variable matches my dataset?”. A very simple (and perfectly good) answer is the eyeball method:

plot the histogram of your dataset, and superimpose the histogram of a random sample from the RNG, and tune the parameter until they look close.

Example. I collected 31,078 lines from the request log of a web server, and extracted the size in bytes of the contents. The sizes vary hugely, from 1 byte to 8,944,270 bytes, so to see them on a sensible scale I computed $\log_2(\text{size})$ and rounded down. I then generated 31,078 random variables using $\text{rgeom}(p)$ for different values of p . This plot shows the results, with one panel for each value of p I tried.



It looks like $\text{rgeom}(p)$ is a bad choice for this dataset, for the three values of p we tried. This prompts two questions: Is there a systematic way to pick the best p ? and Is there a systematic way to pick the best RNG? The answer to the former is Yes. The response to the latter is That’s a fundamentally wrong-headed question, as we’ll discover in Section 2.

Definitions. A random variable is *discrete* if it takes values in some countable space. The *density function* for a discrete random variable X is $f(x) = \mathbb{P}(X = x)$. If the density function depends on some parameter θ , then the *likelihood* of the parameter given a dataset x_1, \dots, x_n is

$$\text{lik}(\theta|x_1, \dots, x_n) = f(x_1)f(x_2) \cdots f(x_n).$$

A sensible way to estimate an unknown parameter, given a dataset, is to find the value of the parameter that maximizes the likelihood. This is called *maximum likelihood estimation*.

Example. Suppose I have a dataset x_1, \dots, x_n , all integers, and my model is the $\text{rgeom}(p)$ random variable. The likelihood is

$$\begin{aligned} \text{lik}(p|x_1, \dots, x_n) &= (1-p)^{x_1-1}p \times \cdots \times (1-p)^{x_n-1}p \\ &= (1-p)^{s-n}p^n \quad \text{where } s = x_1 + \cdots + x_n. \end{aligned}$$

It’s usually easier to maximize $\log \text{lik}()$ rather than $\text{lik}()$, because taking logs turns products into sums. Thus

$$\begin{aligned} \log \text{lik}(p|x_1, \dots, x_n) &= (s-n) \log(1-p) + n \log p, \\ \frac{d}{dp} \log \text{lik}(p|x_1, \dots, x_n) &= -\frac{s-n}{1-p} + \frac{n}{p}. \end{aligned}$$

To maximize this, we solve $d/dp = 0$, giving the maximum likelihood estimator $\hat{p} = n/s$. For the dataset of web server content sizes shown above, it evaluates to $\hat{p} = 0.089$.

* * *

Maximum likelihood estimation is nearly universal. It’s simple to explain. It’s easy to compute (or at least no harder than any other method). It’s got some nice properties:

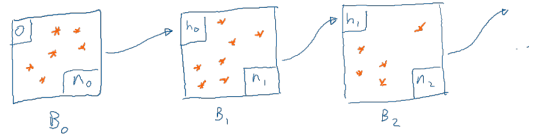
- It’s intuitively plausible. For simple problems like “Toss n biased coins, get x heads, estimate the the probability of heads” it gives the sensible answer x/n .
- If the RNG truly is the correct RNG for the dataset, then one can prove that as the size of the dataset increases, the maximum likelihood estimator is guaranteed to approach the true value of the parameter. We’ll learn more about the properties of big datasets in Section 2.

But there's nothing necessarily 'true' about maximum likelihood estimation. You'll learn from exercises on the example sheet that the maximum likelihood estimator can be biased, and that it performs poorly when there are lots of parameters to estimate.

Pay close attention to the style of reasoning behind the two bullet points. "Here's a method, M . How good is M ? Let's take an RNG, consider a random sample generated from that RNG, and see how close M gets to the truth." This is a good sanity check, and if the method didn't pass this test then we wouldn't want to use it. But, as the quotes on page 2 remind us, any model we use is wrong. What we need to ask is "How robust is this method, i.e. does it give a useful answer when my model is wrong?". This is much harder to answer. As you gain experience of data science, you will develop the skill to look at a method and quickly see what sorts of modeling errors will trick it, and then measure by how much.

1.2. Markov models

Application. Bitcoin¹¹ is a decentralized electronic cash system, introduced by Satoshi Nakamoto in 2008. It has been a wild success, arguably because of the ingenious way it balances incentives¹². An important part, and a focus of Nakamoto’s original paper, was how to solve the ‘double spend’ problem in a decentralized system. To understand what this problem is, and how Bitcoin solves it, let’s start with some background.

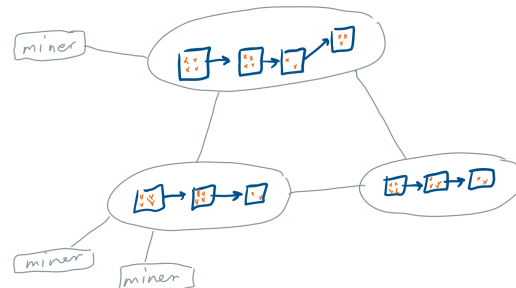


n_0 , a nonce that solves $\text{hash}(0, B_0.\text{records}, n_0) < \text{threshold}$,
 $h_0 = \text{hash}(0, B_0.\text{records})$,
 n_1 , a nonce that solves $\text{hash}(h_0, B_1.\text{records}, n_1) < \text{threshold}$,
 $h_1 = \text{hash}(h_0, B_1.\text{records})$,
 \vdots

Bitcoin is a system for storing and verify transaction records, which are depicted as stars in the diagram. A transaction record might be e.g. $Tx_1 = \text{“Alice transfers coin 314 to Bob”}$, cryptographically signed. Transaction records are assembled into blocks B_0, B_1, \dots , and each block additionally includes two values: the hash of the previous block, and a nonce which solves a computationally demanding inequality.

In the simplest world, there might be a central bank which publishes blocks, say one block every 10 minutes. If Alice wants to pay Bob, she asks the bank to record Tx_1 , the bank checks that previous blocks confirm that Alice owns the coin, Bob waits until the bank publishes a new block containing Tx_1 , and then he posts the widget to Alice. The nonces and hashes are unnecessary, in a centralized system where the bank’s blocks are fully public.

Bitcoin is a decentralized system with roughly 9750 nodes¹³, each of which keeps a copy of the entire blockchain. When Alice wants to record the transaction, she sends it to one of these nodes, which broadcasts it to the rest of the network; it takes 1.5 seconds to reach 50% of the nodes. Other machines work to mine blocks, i.e. to find a nonce with a suitable hash. The time between blocks depends on the threshold; Nakamoto specified an algorithm to dynamically adapt the threshold so that a new block is mined every 10 minutes on average, regardless of the number of miners. Roughly 3.9×10^{21} hashes must be tested to find a block, and each block contains around 2000 transactions. When a block has been mined, the nodes broadcast it to each other, and it takes roughly 5 seconds to reach 50% of the nodes.



¹¹Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2008. URL: <https://bitcoin.org/bitcoin.pdf>.

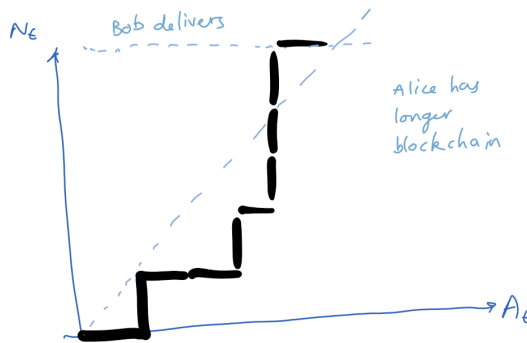
¹²Simon Barber et al. “Bitter to Better : How to Make Bitcoin a Better Currency”. In: *Financial Cryptography—FC 2012*. Vol. 7397. Lecture Notes in Computer Science. 2012, pp. 399–414. URL: <http://www.cs.stanford.edu/~xb/fc12/>.

¹³Bitcoin statistics are from September 2017. Current statistics can be found at bitnodes.21.co/nodes/live-map, bitcoinstats.com/network/propagation, data.bitcoinity.org/bitcoin/block_time, statoshi.info/dashboard/db/mining, www.bitcoinmining.com/bitcoin-mining-hardware, blockchain.info/charts/n-transactions-per-block.

What we've described so far allows money to be double-spent. Suppose Alice broadcasts T_{x_1} "Alice transfers coin 314 to Bob", Bob sees this, and sends Alice the widget. Suppose Alice also creates a new transaction T_{x_2} "Alice transfers coin 314 to Alicia" (her alter-ego), mines a block containing T_{x_2} , and broadcasts it. Now Alice has the widget, and if Alice's block gets spread widely then everyone accepts that Alicia owns the money.

The Bitcoin strategy to prevent double-spend attacks is for nodes to use the rule "If there are two possible chains, discard the shorter", and for Bob to use the rule "Wait for 6 blocks (1 containing T_{x_1} , then 5 more chained after it)" before sending Alice the widget. To double-spend, Alice would need to create an alternative history with T_{x_2} rather than T_{x_1} , and get it accepted by the rest of the nodes. The chance of a successful double-spend attack should be small, assuming Alice doesn't own too much of the worldwide block mining power. What is the chance of this? And why did Nakamoto come up with "wait for 6 blocks"?

Nakamoto's calculation was as follows. Assume that Alice controls a fraction p of the worldwide block mining power. Let A_t be the number of blocks that Alice has mined at time t after her attempted double-spend, and let N_t be the number of blocks mined by everyone else, so they start¹⁴ at $A_0 = N_0 = 0$. At any point in time, the probability that the next block comes from Alice is p , and the probability it comes from someone else is $1-p$. We want to calculate the probability that, at any time after reaching $N_t = 6$, we later hit $A_s > N_s$.



The starting point is to split $\mathbb{P}(\text{Alice double-spends})$ using conditional probability, conditioning on how it might have happened.

$$\mathbb{P}(\text{Alice double-spends}) = \sum_a \mathbb{P}(\text{Alice double-spends} \mid A_t = a \text{ when Bob delivers}) \mathbb{P}(A_t = a \text{ when Bob delivers}).$$

For the first term, with a flash of insight, we spot that all that matters is the gap $N_t - A_t$. At the instant Bob delivers, this is equal to $6 - a$, and thereafter it may go up or down, and if it ever hits -1 then Alice double-spends. Let

$$\pi_x = \mathbb{P}(\text{eventually hit } N_s - A_s = -1 \mid \text{currently at } N_t - A_t = x)$$

and split this using conditional probability, conditioning on who mines the next block:

$$\begin{aligned} \pi_x &= \sum_{m \in \{\text{others}, \text{Alice}\}} \mathbb{P}(\text{eventually hit } N_s - A_s = -1 \mid \text{currently at } N_t - A_t = x, \text{ and next block mined by } m) \\ &\quad \times \mathbb{P}(\text{next block mined by } m \mid \text{currently at } N_t - A_t = x) \\ &= \mathbb{P}(\text{eventually hit } N_s - A_s = -1 \mid \text{now at } N_t - A_t = x + 1)(1-p) \\ &\quad + \mathbb{P}(\text{eventually hit } N_s - A_s = -1 \mid \text{now at } N_t - A_t = x - 1)p \quad (1) \\ &= \pi_{x+1}(1-p) + \pi_{x-1}p \end{aligned}$$

¹⁴What if Alice prepares some malicious blocks in advance, and only launches her attack when she has enough blocks stored? This is a problem, called the selfish miner attack. See for example Yonatan Sompolinsky and Aviv Zohar. "Bitcoin's Security Model Revisited". In: *CoRR* (2016). URL: <http://arxiv.org/abs/1605.09193>.

except for $x \leq -1$ where $\pi_x = 1$. This is now a pure maths recurrence equation, and we can solve it and discover that, as long as $p < 1/2$,

$$\pi_x = \begin{cases} \left(\frac{p}{1-p}\right)^{x+1} & \text{if } x \geq 0, \\ 1 & \text{if } x \leq -1. \end{cases}$$

(You should check that this does indeed solve the recurrence equation. We'll see many more calculations along these lines in Section 4, and we'll discuss what happens when the recurrence equation has more than one solution.)

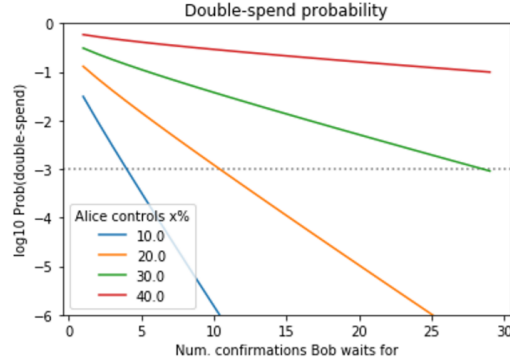
We still have to work out $\mathbb{P}(A_t = a \text{ when Bob delivers})$. Call it ξ_a . By simply counting up all the ways that we might reach $N_t = 6$, along the lines of Example 1, we find

$$\begin{aligned} \xi_0 &= (1-p)^6 \\ \xi_1 &= p(1-p)^2 + (1-p)p(1-p)^5 + \dots + (1-p)^5p(1-p) = \binom{6}{1}p(1-p)^6 \\ \xi_a &= \binom{a+5}{a}p^a(1-p)^6. \end{aligned}$$

Putting everything together,

$$\begin{aligned} \mathbb{P}\left(\begin{array}{l} \text{Alice} \\ \text{double-} \\ \text{spends} \end{array}\right) &= \sum_{a \in \{0, \dots, 6\}} \binom{a+5}{a} p^a (1-p)^6 \left(\frac{p}{1-p}\right)^{6-a+1} + \sum_{a \geq 7} \binom{a+5}{a} p^a (1-p)^6 \\ &= \sum_{a \in \{0, \dots, 6\}} \binom{a+5}{a} p^a (1-p)^6 \left(\frac{p}{1-p}\right)^{6-a+1} + \left(1 - \sum_{a \in \{0, \dots, 6\}} \binom{a+5}{a} p^a (1-p)^6\right). \end{aligned}$$

Here are some numbers, showing the probability that Alice successfully double-spends, depending on the proportion p that of block mining power that she controls, and the number of confirmations that Bob waits for. This is the data that Nakamoto used to choose the rule “wait for 6 confirmations”.



* * *

In this problem, we studied a dynamical process (N_t, A_t) , $t \geq 0$. There were several key steps in how we analysed it:

Conditioning. To find $\mathbb{P}(\text{event})$, we conditioned on how the event happened. We chose the conditioning strategically. We applied it in two ways: we conditioned on A_t at the instant that Bob delivers the widget, and we conditioned on who mines the next block. This conditioning is an application of the *law of total probability*, which states

$$\mathbb{P}(A) = \sum_i \mathbb{P}(A|B_i)\mathbb{P}(B_i)$$

where A is an event and B_1, \dots, B_n are mutually exclusive events that cover all possible ways that A can occur.

Conditioning might remind you of dynamic programming, the algorithmic strategy of decomposing a problem into smaller sub-problems. There is a substantial difference, though. We came up with the equation

$$\pi_x = \pi_{x+1}(1-p) + \pi_{x-1}p$$

which does not lend itself to a recursive algorithm because there is no base case. In dynamic programming, on the other hand, the goal is to turn the problem into a recursive algorithm. In Section 4 we'll look at computational methods for solving equations like this.

Memorylessness. The most important step in the calculation was (1). It expresses the idea “What happens in the future depends only on where you are now, not on how you got here.” Why is this true for bitcoin? The way the bitcoin hash calculation works, finding a nonce is like winning the lottery: if you haven't won one so far, it doesn't mean you're more likely to win next time. If we're at state $N - A = x + 1$, it's immaterial whether we reached there from $N - A = x$ or from $N - A = x + 2$, the future looks exactly the same either way. This is called *memorylessness*, or alternatively the *Markov property* after the Russian mathematician Andrey Markov, who invented the theory of memoryless processes, which we will study in much more detail in Section 4.

There are very many non-Markov processes, but they're often much harder to analyse. Even in the bitcoin problem, we can question whether it truly is memoryless. If for example the number of mining machines was slowly varying, then the fact “ $N - A$ used to be $x + 2$ before it became $x + 1$ ” gives a slight hint that Alice might have a slightly higher number of miners than she started with, which would impact our estimate of what happens in the future. To make the problem memoryless, we assumed that Alice controls precisely p of the worldwide mining power, and that p doesn't change.

Embedded chain. The process (N_t, A_t) evolves in continuous time, but all we chose to look at is the instants where it jumps. This is called *finding an embedded Markov chain*. The word *chain* here means “discrete sequence of events”.

1.3. Descriptive models

A group of four friends, A , B , C , and D , are deciding how to vote in the Brexit referendum. There are 16 possible outcomes. Based on survey statistics for similar groups, the estimated chance of each of the 16 possible outcomes for (A, B, C, D) is

(0, 0, 0, 0)	17.109%	(2)
(0, 0, 0, 1), (0, 0, 1, 0), (0, 1, 0, 0), (1, 0, 0, 0)	6.095%	
(0, 0, 1, 1), (0, 1, 1, 0), (1, 1, 0, 0), (1, 0, 0, 1)	7.821%	
(0, 1, 0, 1), (1, 0, 1, 0)	0.295%	
(1, 1, 1, 0), (1, 1, 0, 1), (1, 0, 1, 1), (0, 1, 1, 1)	3.069%	
(1, 1, 1, 1)	14.361%	

where 0 means remain and 1 means exit.

The voter model is called *multivariate*, meaning that the item of interest is a vector, in this case of length four. It's simple to simulate a random outcome in Python:

```
1 def rvote():
2     outcomes = [(0,0,0,0), (0,0,0,1), ...]
3     probs = [0.17109, 0.060905, ...]
4     return random.choices(outcomes, weights=probs)
```

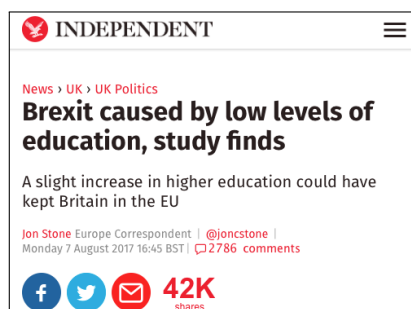
There are many other ways we could implement the simulator. For example, simply adding up the probabilities we get

$$\begin{aligned} \mathbb{P}(A = 0) &= 54.4\% \\ \mathbb{P}(B = 0 \mid A = 0) &= 68.2\%, \\ \mathbb{P}(B = 0 \mid A = 1) &= 37.8\% \\ \mathbb{P}(C = 0 \mid A = 0, B = 0) &= 62.5\%, \\ \mathbb{P}(C = 0 \mid A = 0, B = 1) &= 37.0\%, \dots \end{aligned}$$

which suggests the code

```
1 def rvote2():
2     pA = 0.544
3     A = random.choices([0,1], weights=[pA, 1-pA])
4     pB = {0: 0.682, 1: 0.378}[A]
5     B = random.choices([0,1], weights=[pB, 1-pB])
6     pC = {(0,0): 0.625, (0,1): 0.370, ...}[(A,B)]
7     C = random.choices([0,1], weights=[pC, 1-pC])
8     ...
9     return (A, B, C, D)
```

These two random simulators are equivalent—they generate exactly the same distribution of outcomes. If all we have is a descriptive model based on observed frequencies, it's impossible to tell how the components are generated. The 'outrage media' generates catchy headlines when it muddles descriptive models and causal mechanisms. Here's a headline¹⁵ describing a study that was purely observational:



In Section 1.4 we'll look harder at causal versus descriptive models.

¹⁵<http://www.independent.co.uk/news/uk/politics/brexit-education-higher-university-study-university-leave-eu-remain-voters-educated-a7881441.html>

How would you implement `random.choices()` if your language provides only simple uniform random numbers in $[0, 1]$? There is an obvious method, and a clever trick called the Alias Method.

DISCRIMINATIVE MODELS

When we're presented with a multivariate dataset, we typically start by investigating how one variable depends on the others, or on some summary of the others. This is called the *marginal distribution* of the variable we're investigating. For example, pick any one of the four friends, and call their vote Y , and let X be the total vote of the other three. Then

$$\mathbb{P}(Y = 0 \mid X = x) = \begin{cases} 73.7\%, & \text{if } x = 0 \\ 53.4\%, & \text{if } x = 1 \\ 63.4\%, & \text{if } x = 2 \\ 17.6\%, & \text{if } x = 3. \end{cases} \quad (3)$$

We might write the model this way if we want to predict the behaviour of an individual voter. The terms that we're conditioning on are called *features*, and it's an art to find useful features. We'll discuss this further in Section 5.

Definitions. A *descriptive* or *observational* model specifies the distribution of the data, without specifying the mechanism by which it was generated. A *discriminative* model is a type of descriptive model, written as the marginal distribution of some variable of interest conditional on *features*.

FACTOR DISTRIBUTIONS

A group of four friends are deciding how to vote in the Brexit referendum. For each of them, the probability of voting Remain given the number of friends n who vote Leave is 73.7% (if $n = 0$), or 53.4% (if $n = 1$), or 63.4% (if $n = 2$), or 17.6% (if $n = 3$).

Sometimes, we're only told marginal distributions, as in this second model. This invites questions:

- Is there a descriptive model that's consistent with these marginals, i.e. is it possible to define a random variable (A, B, C, D) that yields these marginals? In this case, yes, because the marginal probabilities are exactly what's specified in equation (3), which we know were derived from the full descriptive model in equation (2).
- Is there always a consistent descriptive model? No. Suppose we wrote down a contrarian model, $\mathbb{P}(A = a \mid B = b) = 1_{a=b}$ and $\mathbb{P}(B = b \mid A = a) = 1_{a \neq b}$. There's no distribution for (A, B) that has these two marginals.
- But isn't that contrarian case pathological? Yes, it is. We can always just define the probability of a particular outcome to be the *product* or *factor distribution*, i.e.

$$\begin{aligned} \mathbb{P}(A = a, B = b, C = c, D = d) \\ = \kappa \mathbb{P}(A = a \mid \dots) \mathbb{P}(B = b \mid \dots) \mathbb{P}(C = c \mid \dots) \mathbb{P}(D = d \mid \dots) \end{aligned}$$

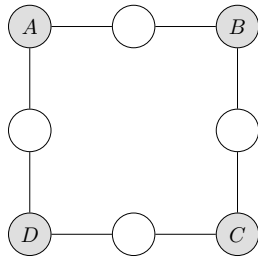
and pick κ so that the probabilities sum to 1. In the pathological case, all the probabilities are zero so it's impossible to pick κ . In all other cases, we can.

- Is there a unique descriptive model that's consistent with a given set of marginal distributions? No. In this case, the original descriptive model (2) turns out to be different to the product distribution.

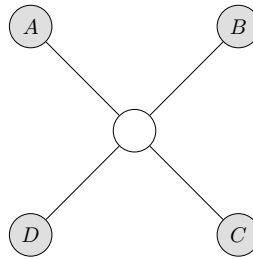
You often see factor distributions drawn as undirected graphs, with a node for each variable and edges to indicate which variables depend on which other variables. They are also called *Markov random fields*, where the word 'Markov' here means 'depends only on neighbours'. The original voter distribution in equation (2) was actually produced from a factor distribution, with some extra hidden variables corresponding to pairs of friends.

The notation 1_Q is shorthand for "1 if Q is true, 0 if Q is false".

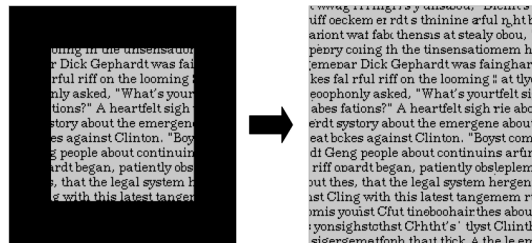
Graph behind the first voter model:



Graph of the second voter model:



Factor distributions are popular with physicists, e.g. the Ising model for magnetism. They can also be used in computer graphics, for synthesizing texture. Take a source image with a sample of the texture; pick a window size, e.g. 3×3 pixels; and find the marginal distribution of the middle pixel conditional on its eight neighbours, using the source image to learn the frequencies. Now consider the entire area we want to texturize to be a multivariate random variable, with the factor distribution, and generate a single sample of this random variable. (In Section 4 we'll learn how to generate samples from factor distributions, using Markov chains.) Here is an example¹⁶.



¹⁶Taken from lecture notes by Steven Seitz at the University of Washington, <https://courses.cs.washington.edu/courses/csep576/05wi/lectures/texture.pdf>

1.4. Causal models

The goal of data science is often to answer policy questions.

Should we cut tuition fees? What does the data tell us will happen?

This is a hypothetical question, about a situation that hasn't yet been observed. Such questions may be dressed up to look like descriptive questions

In places or times where tuition fees were cut, what happened?

but there's an implicit claim

If we were to cut tuition fees now, the same would surely happen.

Sometimes it's expressed as a counterfactual question:

If we had cut tuition fees, what state would we be in now?

Definition. A model is called *causal* or *generative* if the steps in the code represent mechanisms, such that if we intervene and alter some value or mechanism then the rest of the mechanism still works as before. It's convenient to draw causal models as directed acyclic graphs, where the edges show which variables are used to generate which other variables. A variable is called *latent* if it is not observed.

Descriptive models don't tell you the order in which variables are generated. In Section 1.3, for example, we saw two different mechanisms `rvote()` and `rvote2()` for simulating votes, which both correspond to exactly the same descriptive model. But if we intervene and force D to vote remain (by adding a line of code), then the two mechanisms will produce different outcome distributions. So, if all we know are descriptive summaries, it's generally impossible to answer causal questions.

A (fictional) drug is taken by some members of the population, and it leads to better survival outcomes. A zealous health minister wants to add the drug to drinking water. Just before this is approved, a heroic scientist runs a controlled trial, which show the drug actually leads to marginally worse outcomes. The health minister's plan is scrapped, saving a small number of lives and a large amount of money.

	population		trial		
	drug	clean	drug	clean	
$\mathbb{P}(\text{death})$.002	.028	.016	.014	(4)
$\mathbb{P}(\text{survival})$.998	.972	.984	.986	

The numbers in this table are taken from Tian and Pearl¹⁷, who put them instead in a much more interesting legal counterfactual context:

A lawsuit is filed against the manufacturer of drug x , charging that the drug is likely to have caused the death of Mr A, who took the drug to relieve symptom S associated with disease D .

- *The manufacturer claims that experimental data on patients with symptom S show conclusively that drug x may cause only minor increase in death rates.*
- *The plaintiff argues, however, that the experimental study is of little relevance to this case, because it represents the effect of the drug on all patients, not on patients like Mr A who actually died while using drug x . Moreover, argues the plaintiff, Mr A is unique in that he used the drug of his own volition, unlike subjects in the experimental study who took the drug to comply with experimental protocols. To support this argument, the plaintiff furnishes nonexperimental data indicating that most patients who chose drug x would have been alive if it were not for the drug.*

Manufacturer says the 'drug may cause only minor increase in death rates' i.e. from 1.4% to 1.6%, which is a tiny change, so it's very unlikely that a single death can be definitively blamed on the drug.

Plaintiff says 'most patients who chose drug x would have been alive' i.e. 97.2% of them, assuming the population is homogeneous.

¹⁷Jin Tian and Judea Pearl. "Probabilities of Causation: Bounds and Identification". In: *Proc. of the 16th Conference on Uncertainty in Artificial Intelligence*. 2000. URL: <https://arxiv.org/ftp/arxiv/papers/1301/1301.3898.pdf>.

- *The manufacturer counter-argues by stating that: (1) counterfactual speculations regarding whether patients would or would not have died are purely metaphysical and should be avoided, and (2) nonexperimental data should be dismissed a priori, on the ground that such data may be highly biased; for example, incurable terminal patients might be more inclined to use drug x if it provides them greater symptomatic relief.*

The court must now decide, based on both the experimental and non-experimental studies, what the probability is that drug x was in fact the cause of Mr A's death.

Let's invent a probabilistic model that generates outcomes like those in the table. We'll invent one specific model and choose parameters for it to match table 4, and calculate the probability of interest. The remarkable contribution of Tian and Pearl was to show that we would get the same answer for *any* probabilistic model with any parameters, as long as the model and parameters are consistent with table 4.

We'll imagine there are two types of patient, **regular** and **terminal**, as the manufacturer's lawyer suggests. Greek letters indicate parameters of the model.

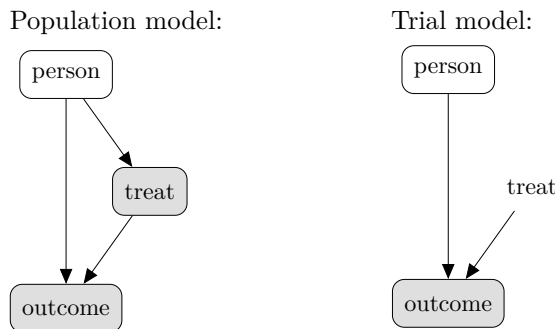
```

1 def population():
2     person = random.choices(['regular', 'terminal'], weights=[π, 1 - π])
3     p_drug = θ[person]
4     treat = random.choices(['drug', 'clean'], weights=[p_drug, 1-p_drug])
5     p_death = ξ[(person, treat)]
6     return random.choices(['die', 'survive'], weights=[p_death, 1-p_death])
7
8 def trial(treat):
9     person = random.choices(['regular', 'terminal'], weights=[π, 1 - π])
10    p_death = ξ[(person, treat)]
11    return random.choices(['die', 'survive'], weights=[p_death, 1-p_death])
12
13 π = 0.1183
14 θ = {'regular': 0, 'terminal': 0.125}
15 ξ = {'regular', 'drug': 0, ('terminal', 'drug'): 0.0181,
16      ('regular', 'clean'): 0.1184, ('terminal', 'clean'): 0}

```

Outline of their argument. In any causal model, all that matters are four types of patients: those who would like to take the drug and would die on it, those who wouldn't like to take it and who would die on it, and so on. First, extend the code to have four types of person, not just regular and terminal. Second, use numerical optimization to find the lowest and highest possible values for the probability of interest, over all parameter choices consistent with table 4.

Drawn as directed acyclic graphs, with white boxes for latent random variables, shaded boxes for observed random variables, and plain text for non-random variables,



If we simply count up the probability of each outcome, we get the formulae in the table below. In the population table all four probabilities sum to 1 (since the table describes the characteristics of a random person), whereas in the trial table each column sums to 1 (since the table describes the outcome for each type of patient in the trial). These equations are what I used to verify that the parameter choices in lines 13–16 agree with table 4.

population:	drug	clean
$\mathbb{P}(\text{death})$	$\pi\theta\xi_{r,d} + (1 - \pi)\phi\xi_{t,d}$	$\pi(1 - \theta)\xi_{r,c} + (1 - \pi)(1 - \phi)\xi_{t,c}$
$\mathbb{P}(\text{survival})$	$\pi\theta(1 - \xi_{r,d}) + (1 - \pi)\phi(1 - \xi_{t,d})$	$\pi(1 - \theta)(1 - \xi_{r,c}) + (1 - \pi)(1 - \phi)(1 - \xi_{t,c})$
trial:	drug	clean
$\mathbb{P}(\text{death})$	$\pi\xi_{r,d} + (1 - \pi)\xi_{r,d}$	$\pi\xi_{r,c} + (1 - \pi)\xi_{r,c}$
$\mathbb{P}(\text{survival})$	$1 - \pi\xi_{r,d} + (1 - \pi)\xi_{r,d}$	$1 - \pi\xi_{r,c} + (1 - \pi)\xi_{r,c}$

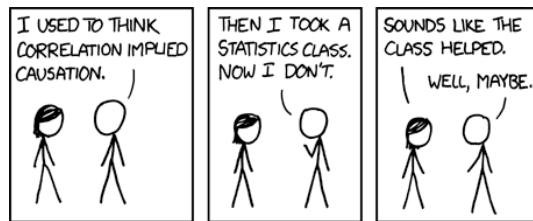
The legal counterfactual question is this: We know Mr A took the drug and died; given this, what's the probability he would have died if he hadn't taken the drug? Let's use Bayes's rule to first work out the probability that Mr A is regular or terminal given these facts.

$$\begin{aligned} & \mathbb{P}(\text{regular} \mid \text{drug, dead}) \\ &= \frac{\mathbb{P}(\text{drug, dead} \mid \text{regular})\mathbb{P}(\text{regular})}{\mathbb{P}(\text{drug, dead} \mid \text{regular})\mathbb{P}(\text{regular}) + \mathbb{P}(\text{drug, dead} \mid \text{terminal})\mathbb{P}(\text{terminal})} \end{aligned}$$

and this is 0 because **regular** people do not die on the drug, $\xi_{\text{regular,drug}} = 0$. Therefore Mr A is terminal. But $\xi_{\text{terminal,clean}} = 0$, i.e. **terminal** people do not die when they're not on the drug, therefore Mr A's death was caused by the drug.

* * *

You sometimes come across the glib remark 'correlation does not imply causation'. Causality theory goes far beyond this. It is a relatively new area, still barely developed, and still contentious. Machine learning algorithms are getting very good at descriptive and discriminative models, but I think it will be 15 years or more before AIs can manage general causal reasoning. Even today, according to Pearl¹⁸, many human statisticians still find it perplexing that we can draw a counterfactual conclusion about a single individual using observational data. Here's a last word from the inimitable Randall Munroe.¹⁹



* * *

What is modelling for?

- Sometimes we put forwards a model, even though we don't really believe it's accurate, because it lets us make useful inferences about hidden variables. Any reasonably sensible model for human motion should let us infer walking / cycling / driving from GPS traces, and probabilistic modeling is a convenient and interpretable tool that won't lead to crazy answers.
- Sometimes we have detailed observational data and we just want to find a reasonably good and simple approximation, either for storage or speed reasons. Here we're looking for a reasonably good fit for the distribution, and we want to find useful features and parameters.
- Sometimes we want to build a product that makes predictions, e.g. that classifies new images. Probabilistic models are one way to build predictors, and they're often useful when we want to assess how well our product might perform on data of a type it hasn't seen before.
- Sometimes we're interested in building a simulator of a system that we understand well. It's helpful to be familiar with a range of probability models, to be able to pull 'off the shelf' whatever is appropriate and has good library support.
- Sometimes we want to answer policy questions, or to build our scientific understanding. This, in my opinion, is the truest data science, and by far the most challenging.

¹⁸Judea Pearl. *Causality: Models, Reasoning and Inference*. 2nd ed. Cambridge University Press, 2009.

¹⁹<https://xkcd.com/552/>

1.5. Common random variables

It's useful to have a range of common random variables at our fingertips.

Definition. A random variable X is *discrete* if it takes values in some countable space, such as the integers. The *density* is $f(x) = \mathbb{P}(X = x)$. The density must be everywhere ≥ 0 and must sum to 1.

Definition. A random variable X is *continuous* if it takes values in a continuous set, such as the real numbers, or $[0, 1]$. A continuous random variable has a *density function* $f(x)$ such that

$$\mathbb{P}(X \in A) = \int_{x \in A} f(x) dx \quad \text{for all sets } A.$$

The density must be everywhere ≥ 0 and must integrate to 1.

VARIABLES ASSOCIATED WITH WAITING AND COUNTING

Geometric: If we're playing a lottery, and each week the chance of winning is p , then our first win happens on week $X \sim \text{Geom}(p)$. This random variable takes values in $\{1, 2, \dots, n\}$, and

$$\mathbb{P}(X = r) = (1 - p)^{r-1}p, \quad \mathbb{P}(X \geq r) = (1 - p)^{r-1}.$$

We came across it in Section 1.1. In Python, `numpy.random.geometric(p)`.

Exponential: The Exponential random variable is a continuous-time version of the Geometric. It's used to model the time until an event, for many natural processes: for example the time until a lump of radioactive matter emits its next particle, or the time until a lightbulb blows, or the time until the next web request arrives. If $X \sim \text{Exp}(\lambda)$ then it takes values in $[0, \infty)$, and

$$f(x) = \lambda e^{-\lambda x}, \quad \mathbb{P}(X \geq x) = e^{-\lambda x}, \quad \mathbb{E}X = \frac{1}{\lambda}.$$

The parameter λ is called the *rate*. The chance of an event in a short interval of time $[t, t + \delta]$ is

$$\mathbb{P}(X \leq t + \delta \mid X \geq t) = \frac{\mathbb{P}(X \in [t, t + \delta])}{\mathbb{P}(X \geq t)} = \frac{\int_t^{t+\delta} \lambda e^{-\lambda x} dx}{e^{-\lambda t}} \approx \delta \lambda.$$

In Python, `numpy.random.exponential(scale=1/\lambda)`.

Binomial: If we toss a biased coin n times, and each coin has chance p of heads, the total number of heads is $X \sim \text{Bin}(n, p)$. This random variable takes values in $\{0, 1, \dots, n\}$, and

$$\mathbb{P}(X = r) = \binom{n}{r} p^r (1 - p)^{n-r}.$$

In Python, `numpy.random.binomial(n, p)`. When $n = 1$, i.e. a single coin toss, it's called a Bernoulli random variable. There is a related random variable called the negative binomial, which arose in Section 1.2 when we calculated $\mathbb{P}(A_t = a \text{ when Bob delivers})$.

Multinomial: If we have n individuals each of whom falls into one of k categories, and the probability of falling into category i is p_i , then the total number in each category is a multivariate random variable $X \sim \text{Multinom}(n, p)$. We used it for counting outcomes in the drug model in Section 1.4. It takes values in $\{0, 1, \dots, n\}^k$, and

$$\mathbb{P}(X = x) = \frac{n!}{x_1! x_2! \dots x_k!} p_1^{x_1} p_2^{x_2} \dots p_k^{x_k}.$$

In Python, `numpy.random.multinomial(n, p)`. (The binomial distribution is the special case when $k = 2$.)

Poisson: The random variable $X \sim \text{Poisson}(\lambda)$ takes values in $\{0, 1, \dots\}$, and

$$\mathbb{P}(X = r) = \frac{\lambda^r e^{-\lambda}}{r!}.$$

In Python, `numpy.random.poisson(λ)`. Suppose we're counting the number of events in a fixed interval of time, for example the number of buses passing a spot on the street, or the number of web requests, or the number of particles emitted by a lump of radioactive matter. If the time between events is $\text{Exp}(\lambda)$, then the total number of events in time t is $X \sim \text{Poisson}(\lambda t)$.

VARIABLES ASSOCIATED WITH SIZES

Normal/Gaussian: This distribution is a very popular choice for data analysis because it's often a good model for things that are the aggregate of many small pieces, for example height which is the aggregate of many influences from genetics and the environment. It's also easy to do probability calculations with it. If $X \sim \text{Normal}(\mu, \sigma^2)$, also written $X \sim N(\mu, \sigma^2)$, then X is a continuous random variable taking values in the entire real line, and

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad \mathbb{E}X = \mu, \quad \text{Var } X = \sigma^2.$$

In Python, `numpy.random.normal(loc= μ , scale= σ)` (and watch out for σ versus σ^2). If $X \sim N(\mu, \sigma^2)$ and $Y \sim N(\nu, \rho^2)$ and they are independent, then

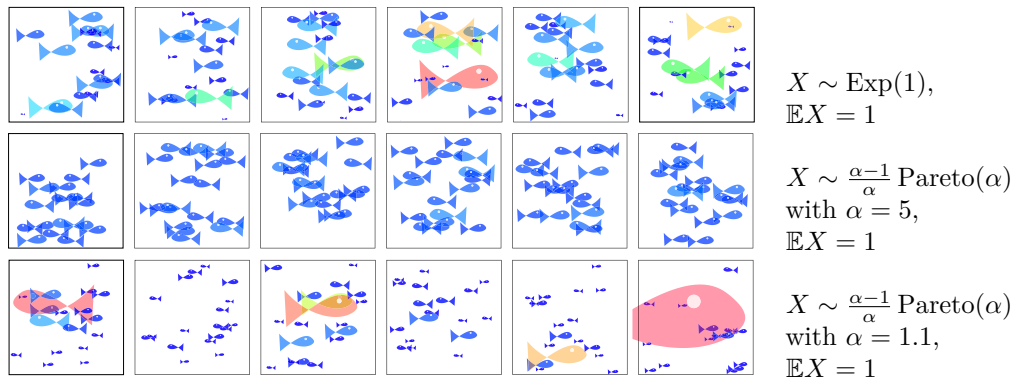
- $aX + b \sim N(a\mu + b, a^2\sigma^2)$
- $(X - \mu)/\sigma \sim N(0, 1)$
- $X + Y \sim N(\mu + \nu, \sigma^2 + \rho^2)$.

There is also a multivariate version, called the multivariate normal.

Pareto and lognormal: Some natural phenomena, like sizes of forest fires, or insurance claims, or Internet traffic volumes, or stock market crashes, have the characteristic that there are events of wildly different sizes. This tends to cause problems for simulations and forecasting, since the entire outcome can hinge on one 'black swan' event²⁰. A common random variable with this characteristic is the Pareto distribution, $X \sim \text{Pareto}(\alpha)$, named after the Italian economist Vilfredo Pareto who studied extreme wealth inequality. It is a continuous random variable taking values in $[1, \infty)$, and

$$f(x) = \alpha x^{-(\alpha+1)}, \quad \mathbb{P}(X \geq x) = x^{-\alpha}, \quad \mathbb{E}X = \begin{cases} \infty & \text{if } \alpha \leq 1 \\ \alpha/(\alpha - 1) & \text{otherwise.} \end{cases}$$

For $\alpha < 2$ it tends to produce many small values ('mice') and very occasional huge values ('elephants'). To illustrate, here are some samples drawn from three different distributions, all with mean value 1.



The lognormal distribution $X \sim e^{N(\mu, \sigma^2)}$ has similar characteristics to the Pareto but is not quite as extreme. It was invented by the Cambridge senior wrangler and medic Donald MacAlister.

²⁰Nassim Nicholas Taleb. *The Black Swan: The Impact of the Highly Improbable*. 2nd ed. Random House, 2010.

Zipf: The random variable $X \sim \text{Zipf}(n, s)$ takes values in $\{1, 2, \dots, n\}$ and

$$\mathbb{P}(X = r) = \frac{r^{-s}}{1 + 2^{-s} + \dots + n^{-s}}.$$

It is named after the American linguist Goerge Zipf, who used it to describe frequencies of words in texts²¹. Take a large piece of text, and count the number of occurrences of each word, and rank the words from most common to least common. Say that the most common word has rank 1, the next most common has rank 2, and so on. Zipf observed that the number of occurrences of the r th ranked word is roughly $\text{const} \times r^{-s}$ where $s \approx 1$ in English texts. Another way of putting this: if we pick a word at random from the entire body of text, then the rank of that word is $\text{Zipf}(n, s)$, where n is the size of the vocabulary. The same phenomenon happens with cities: if we take a person at random from the entire population, and look at which city they come from, and rank cities by size, then the rank of that person's city is $\text{Zipf}(n, s)$ where n is the number of cities and s is roughly 1.07.

There is a direct link between the $\text{Pareto}(\alpha)$ and $\text{Zipf}(n, 1/\alpha)$ distributions. First, create a 'pseudo-random' sample of n city sizes, to match the $\text{Pareto}(\alpha)$ distribution. Make the largest city have size $x_{(1)}$ such that $x_{(1)}^{-\alpha} = 1/N$, make the second-largest city have size $x_{(2)}$ such that $x_{(2)}^{-\alpha} = 2/N$, etc. This is a deterministic equivalent of the $\text{Pareto}(\alpha)$ distribution, in which $\mathbb{P}(X \geq x) = x^{-\alpha}$. Then, the city of rank r has size $\text{const} \times r^{-1/\alpha}$, which fits with $\text{Zipf}(n, 1/\alpha)$.

²¹See the IA course *Machine Learning and Real-World Data*

1.6. Independence and joint distributions

The concept of independent random variables is fundamental in modeling. Informally it means “knowing the value of one of them gives no information about the other.” We’ve used the word several times so far, but we haven’t defined it.

Definition. Two random variables X and Y are *independent* if

$$\mathbb{P}(X \in A, Y \in B) = \mathbb{P}(X \in A) \mathbb{P}(Y \in B) \quad \text{for all } A \text{ and } B.$$

For discrete random variables it’s sufficient to check

$$\mathbb{P}(X = x, Y = y) = \mathbb{P}(X = x) \mathbb{P}(Y = y) \quad \text{for all } x \text{ and } y,$$

and for continuous random variables with joint density function $f_{X,Y}(x, y)$, it’s sufficient to check

$$f_{X,Y}(x, y) = f_X(x) f_Y(y) \quad \text{for all } x \text{ and } y.$$

We can also write the definition in terms of conditional probability:

$$\mathbb{P}(X \in A | Y \in B) = \mathbb{P}(X \in A) \quad \text{for all } A \text{ and } B \text{ such that } \mathbb{P}(Y \in B) > 0.$$

A collection of independent random variables drawn from the same distribution, such as we investigated in Section 1.1, are said to be *independent and identically distributed*, abbreviated IID.

Example. I throw a fair die. Let Z be the result. Let $X = Z \bmod 2$ and let $Y = Z \text{ div } 3$, so for example $Z = 3$ gives $X = 1$ and $Y = 1$. Are X and Y independent? The definition gives a condition that has to be satisfied for all x and y . Let’s try some:

- Try $x = 0, y = 0$. For these, $\mathbb{P}(X = 0, Y = 0) = \mathbb{P}(Z = 4) = 1/6$, and $\mathbb{P}(X = 0) = 1/2$ and $\mathbb{P}(Y = 0) = 1/3$. So this pair passes the test.
- Try $x = 0, y = 1$. For these, $\mathbb{P}(X = 0, Y = 1) = \mathbb{P}(Z = 4) = 1/6$, and $\mathbb{P}(X = 0) = 1/2$ and $\mathbb{P}(Y = 1) = 1/2$. So the test fails.

Thus X and Y are not independent.

Exercise 1.1. I throw a fair die. Let Z be the result. Let $X = (z - 1) \bmod 2$ and $Y = (Z - 1) \text{ div } 2$. Show that X and Y are independent.

Example. In the voting example in Section 1.3, equation (2), are A and $N = B + C + D$ independent? No: when we added up the probabilities we saw $\mathbb{P}(A = 0 | N = 0) = 0.737$ and $\mathbb{P}(A = 0 | N = 1) = 0.534$, so they can’t be independent.

Example. Let X and Y be independent $\text{Bin}(1, p)$ random variables, so

$$\mathbb{P}(X = x, Y = y) = p^x (1 - p)^{1-x} p^y (1 - p)^{1-y},$$

and suppose p is fixed but unknown. Obviously, learning the value of X tells us something about p (exercise: show that the maximum likelihood estimator for p given X is $\hat{p} = X$). That doesn’t prevent X and Y from being independent: the joint probability still factorizes into an x -part and a y -part, so the definition of independence is satisfied.

Whenever you hear “independent random variables”, it’s a good idea to whisper to yourself the coda “given their parameters”, so you don’t confuse ‘unrelated’ and ‘independent’.

Exercise 1.2. In this code snippet,

```

1 def PXY():
2     P = random.random() # generates a random number in [0,1]
3     X = numpy.random.binomial(1, P)
4     Y = numpy.random.binomial(1, P)
5     return (P,X,Y)
```


show that X and Y are not independent. Note however that

$$\mathbb{P}(X = x, Y = y \mid P = p) = p^x(1-p)^{1-x} p^y(1-p)^{1-y}$$

which we describe as “ X and Y are conditionally independent given P ”.

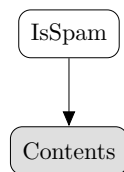
* * *

If we ever try to compute a probability or expectation and we end up with a random variable on the right hand side, we’ve made a mistake. Probabilities are numbers in $[0, 1]$, and random variables are functions, and we should be hyper-vigilant about which is which. In machine learning we want to write things like

$$\mathbb{P}(\text{email is spam}) = \text{some function of email contents.}$$

It’s usually intuitively clear what is meant, but when we come across such statements deep in the middle of a problem with 15 other moving parts it’s sometimes befuddling. Are the email contents random? If so, what are they doing on the right hand side of a probability equation? If not, how can the spam-nature be a random variable yet the email’s contents be non-random?

It often helps to draw out a belief graph, of the sort we drew in Section 1.4, and to label all random variables with capital letters, and values with lower case letters. Here we really mean



and the probability we want is

$$\mathbb{P}(\text{IsSpam} = \text{true} \mid \text{Contents} = c) = \text{function}(c).$$

As a shorthand for this, we write

$$\mathbb{P}(\text{IsSpam} = \text{true} \mid \text{Contents}) = \text{function}(\text{Contents}).$$

FOR MATHEMATICIANS ONLY

At the beginning of Section 1 we used the working definition “a random variable is a function that can return different answers”. Now let’s give a better account, which lets us talk about the joint distribution of two random variables X and Y . This will help clear up some difficulties with continuous random variables. It’s usually intuitively clear what is needed, so you should treat this section as background reading, for interest only.

A random variable is a function that can return different answers, in the following sense:

```

1  ω = simulate_experiment(parameters)
2  def X():
3      return some function of ω
4  def Y():
5      return some other function of ω
  
```

Even for Markov chains, we should think of ω as a complete trace of the entire process, run forever. (Mathematicians don’t worry about simple things like finite memory or keeping variables out of the global scope.) A joint distribution like $\mathbb{P}(X = x, Y = y)$ really means $\mathbb{P}(\{\omega : X(\omega) = x, Y(\omega) = y\})$. Most of the time we don’t have to worry about this level of detail, but it’s useful in some tricky cases.

Definitions. A pair of continuous random variables X and Y has a *joint density function* $f_{X,Y}(x,y)$ such that

$$\mathbb{P}((X, Y) \in A) = \int_{(x,y) \in A} f_{X,Y}(x,y) dx dy$$

for all sets A in the real number line squared. The density must be everywhere ≥ 0 and must integrate to 1. The marginal density of one of them is

$$f_Y(y) = \int_x f_{X,Y}(x,y) dy$$

and the conditional density is

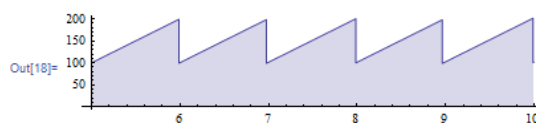
$$f_{Y|X}(y | X = x) = \frac{f_{X,Y}(x,y)}{f_X(x)}, \quad \text{assuming } f_X(x) > 0.$$

2. Distributions of random variables

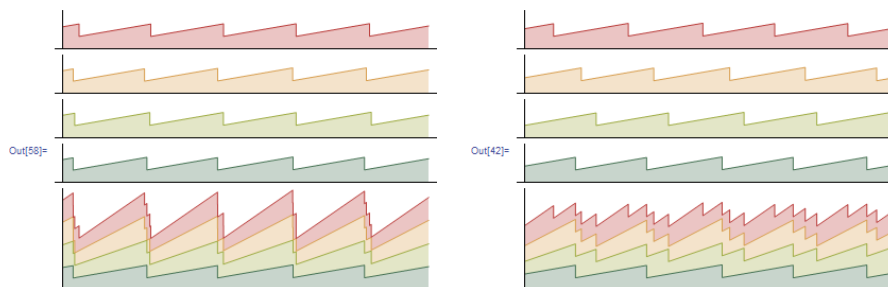
Goals. Get practice at generating and reasoning about random variables. Understand what the empirical distribution is, and what it is for. See how limit theorems are used, in the context of Monte Carlo estimation. Form an intuitive understanding of how random samples behave.

2.1. Working with random variables

Application. For most traffic flows on the Internet, the rate at which the server sends data is controlled by the TCP algorithm. It aims to detect Internet congestion, and it adjusts the data rate to strike a balance between ‘use all available capacity’ and ‘don’t cause overload’. It does this by steadily increasing the sending rate (increasing it by 1 packet per round trip time, every round trip time) until a packet is dropped, which signifies congestion, whereupon it cuts the sending rate in half. This produces the characteristic “TCP sawtooth”.



Suppose a network operator wants to build in enough capacity to support 1000 users each running at 30 kB/sec. How much capacity is needed? In the worst case the sawteeth might all be aligned, giving a peak rate of 40 MB/sec. (To find this, let x_{peak} be the peak rate, note that the trough is $x_{\text{trough}} = x_{\text{peak}}/2$ because of TCP’s congestion rule. The average is $(x_{\text{trough}} + x_{\text{peak}})/2$ and this is 30 kB/sec. Solving for x_{peak} gives 40 kB/sec.) Intuitively we might guess that perfect alignment is unlikely, and that the troughs on one sawtooth are likely to cancel out the troughs on another. This is called *statistical multiplexing*. How much statistical multiplexing should we expect?



RULES FOR EXPECTATION AND VARIANCE

This section of the course is all about numerical random variables. What makes them particularly useful is that they can be summed and averaged, which lets us define their *expectation*. They’re so useful that we often write ‘random variable’ to mean ‘numerical random variable’, and use other wording when it’s not numerical.

$$\mathbb{E}X = \sum_x x\mathbb{P}(X = x) \quad \text{for a discrete random variable,}$$

$$\mathbb{E}X = \int_x xf(x) dx \quad \text{for a continuous random variable with density } f.$$

For a function of a random variable $Y = f(X)$, there are two equivalent ways to compute the expectation:

$$\mathbb{E}f(X) = \sum_y y\mathbb{P}(f(X) = y) = \sum_x f(x)\mathbb{P}(X = x)$$

and similarly for continuous random variables. Now, some handy results about expectation. For any random variable X , the *variance* and *standard deviation* are

$$\text{Var } X = \mathbb{E}((X - \mathbb{E}X)^2), \quad \text{std. dev}(X) = \sqrt{\text{Var } X}.$$

This equality is called the *law of the unconscious statistician*, because it’s easy to interchange them without even realizing one is doing so.

For all constants a and b ,

$$\begin{aligned}\mathbb{E}(aX + b) &= a(\mathbb{E}X) + b \\ \text{Var}(aX + b) &= a^2 \text{Var} X \\ \text{std. dev}(aX + b) &= a \text{std. dev}(X).\end{aligned}$$

For any two random variables X and Y ,

$$\mathbb{E}(X + Y) = (\mathbb{E}X) + (\mathbb{E}Y).$$

For any two independent random variables X and Y ,

$$\begin{aligned}\mathbb{E}(XY) &= (\mathbb{E}X)(\mathbb{E}Y) \\ \text{Var}(X + Y) &= \text{Var} X + \text{Var} Y \\ \text{std. dev}(X + Y) &= \sqrt{\text{std. dev}(X)^2 + \text{std. dev}(Y)^2}.\end{aligned}$$

The *covariance* of two random variables X and Y is

$$\text{Cov}(X, Y) = \mathbb{E}((X - \mathbb{E}X)(Y - \mathbb{E}Y)).$$

The *conditional expectation* $\mathbb{E}(X | Y = y)$, for a discrete random variable Y , is

$$\mathbb{E}(X | Y = y) = \sum_x x \mathbb{P}(X = x | Y = y) = \sum_x x \frac{\mathbb{P}(X = x, Y = y)}{\mathbb{P}(Y = y)}.$$

We write $\mathbb{E}(X | Y)$ to mean “Define $f(y) = \mathbb{E}(X | Y = y)$, and return $f(Y)$ ”. This is a random variable (because it’s a function of Y which is itself a random variable). When Y is a continuous random variable, $\mathbb{P}(Y = y) = 0$ so the conditioning doesn’t make sense—it’s a divide-by-zero error—so we just replace probabilities by densities and sums by integrals.

CONFIDENCE INTERVALS

Here are two facts about the normal distribution: if X has a normal distribution then so does $aX + b$ for any constants $a \neq 0$ and b ; and the interval $[-1.96, 1.96]$ is a 95% confidence interval for $\text{Normal}(0, 1)$ i.e.

$$\mathbb{P}(-1.96 \leq \text{Normal}(0, 1) \leq 1.96) \approx 0.95.$$

Example. What is a 95% confidence interval for $\text{Normal}(\mu, \sigma^2)$, the normal distribution with mean μ and variance σ^2 ?

Let $X \sim \text{Normal}(\mu, \sigma^2)$. Using the rules for expectation and variance, $\mathbb{E}(X - \mu) = (\mathbb{E}X) - \mu = 0$, and $\text{Var}(X - \mu) = \text{Var} X = \sigma^2$, thus $\text{Var}((X - \mu)/\sigma) = 1$. So $(X - \mu)/\sigma \sim N(0, 1)$, thus

$$\mathbb{P}(-1.96 \leq \frac{X - \mu}{\sigma} \leq 1.96) \approx 0.95.$$

Rearranging the expression inside the brackets,

$$\mathbb{P}(\mu - 1.96\sigma \leq \text{Normal}(\mu, \sigma^2) \leq \mu + 1.96\sigma) \approx 0.95. \quad (5)$$

* * *

Here’s a general purpose rule of thumb:

A random variable X can be approximated by $\text{Normal}(\mathbb{E}X, \text{Var} X)$.

This is so useful and simple that it can’t possibly be true—but what’s surprising is that it’s often nearly true. We’ll see circumstantial evidence for why the approximation is so good in Section 2.3, and in the example sheet you’ll investigate cases where it doesn’t work.

Example. I throw a die 100 times and compute the total score. What range of values should I expect?

Let X be the outcome of a single throw. We can explicitly calculate its mean and variance:

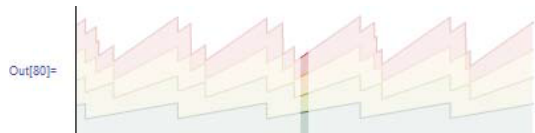
$$\begin{aligned}\mathbb{E}X &= 1 \times 1/6 + 2 \times 1/6 + \cdots + 6 \times 1/6 = 7/2, \\ \text{Var } X &= (1 - 7/2)^2 \times 1/6 + \cdots + (6 - 7/2)^2 \times 1/6 = 35/12.\end{aligned}$$

Let Y be the sum of 100 independent copies of X . By the rules for mean and variance of sums of independent random variables,

$$\mathbb{E}Y = 100 \times 7/2, \quad \text{Var } Y = 100 \times 35/12.$$

Using the normal approximation, $Y \approx \text{Normal}(700/2, 3500/12)$. Applying the approximation(5), we are 95% confident that Y lies in the range [316, 384].

Example (statistical multiplexing). Returning to the TCP example, consider an arbitrary instant in time. Let X_1, \dots, X_n be the sending rate of each of $n = 1000$ flows at this time, and let $Y = X_1 + \cdots + X_n$ be the total.



We might have caught a flow at any point in its sawtooth, so each X_i might take any value between the trough and the peak i.e. in the range $[2x/3, 4x/3]$ where $x = 30$ kB/sec is the average rate we want to support. Furthermore, because of the shape of the sawtooth, each value in this range is equally likely. The appropriate distribution is thus

$$X_i \sim \text{Uniform}(2x/3, 4x/3)$$

After looking up the formulae for mean and variance on Wikipedia,

$$\begin{aligned}\mathbb{E}X_i &= \frac{2x/3 + 4x/3}{2} = x \\ \text{Var } X_i &= \frac{(4x/3 - 2x/3)^2}{12} = \frac{x^2}{27}.\end{aligned}$$

Using the rule for expectation of sums,

$$\mathbb{E}Y = \mathbb{E}X_1 + \cdots + \mathbb{E}X_n = nx$$

and assuming the X_i are all independent then

$$\begin{aligned}\text{Var } Y &= \text{Var } X_1 + \cdots + \text{Var } X_n = \frac{nx^2}{27} \\ \text{std. dev}(Y) &= \sqrt{\text{Var } Y} = \sqrt{n} \frac{x}{\sqrt{27}}.\end{aligned}$$

With probability 95%, Y will lie in the range

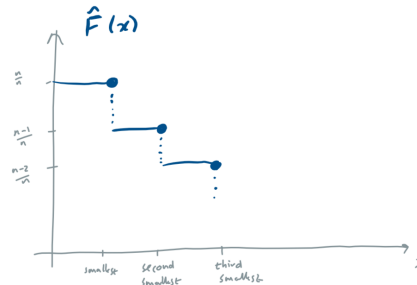
$$[\mathbb{E}Y - 1.96 \text{std. dev}(Y), \mathbb{E}Y + 1.96 \text{std. dev}(Y)]$$

which evaluates to [29.8, 30.2] MB/sec. This is much less than the worst-case value 40 MB/sec.

2.2. Custom distributions

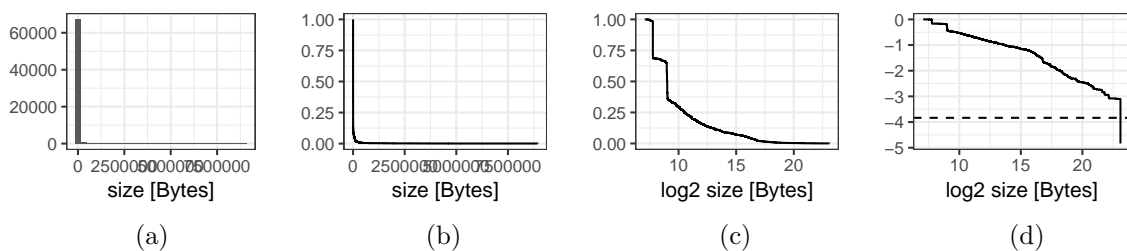
Application. I've collected logs from my web server, $n = 68,506$ records, and I want to program a random number generator that mimics the file sizes I see in these logs. I start by plotting a histogram of file size, shown as (a) below. This is useless, because nearly all sizes are tiny and a handful are gigantic, and the binning of the histogram hides all the detail. A good way to see more detail is to plot what is known as the *empirical distribution*,

$$\hat{F}(x) = \frac{1}{n} (\text{how many items there are } \geq x).$$



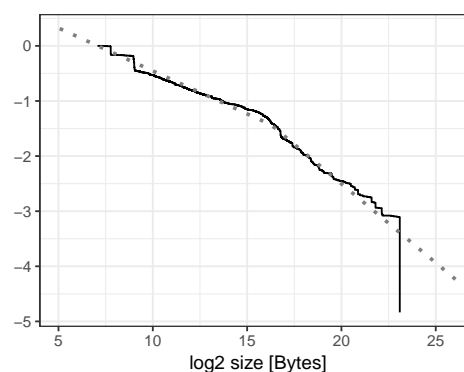
Warning: there is also the empirical cumulative distribution function, which counts the number of datapoints that are $\leq x_i$. When you read 'distribution function', you need to work out from the context whether the writer means a cumulative distribution function or a tail distribution function.

which is easy to plot by sorting the data and putting it on the x axis. The empirical distribution of web server file sizes is shown in (b). It's still not showing very much detail because of the scale, so I'll apply the golden rule of engineering: "if you don't like what you see, take logs". In (c) I've taken logs of the x axis and in (d) I've also taken logs of the y axis, and that makes my data look nice and regular. The dotted reference line is at $\log_{10}(10/n)$ — this way I can see that the precipitous drop at the right hand size of (d) isn't just a single outlier, but it's not much more than 10 datapoints.



How do I know which standard random variable to use, to match this dataset? Or, even better, can I construct a custom random number generator to match? It looks like the data is trying to tell me there are two straight lines (plus a handful of huge files, which I'll ignore for now), i.e. that for some parameters α , β , γ and θ which I can fit from the data,

$$\log \hat{F}(x) \approx \alpha - \beta \log x - \gamma \max(\log x - \theta, 0).$$



THE DISTRIBUTION FUNCTION AND THE INVERSION METHOD

The *distribution function* of a random variable X is

$$F(x) = \mathbb{P}(X \geq x).$$

Sometimes it's easier to work with the distribution function rather than the density:

Example. What is the density function of the continuous random variable X generated by this code?

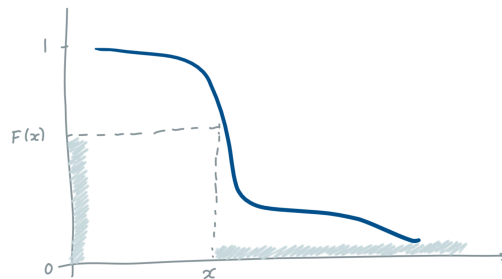
```
1 def rx():
2     u = random.random()
3     return u*u
```

Let's work out its distribution function first.

$$\mathbb{P}(X \geq x) = \mathbb{P}(U^2 \geq x) = \mathbb{P}(U \geq \sqrt{x}).$$

Since U is a simple uniform random variable on $[0, 1]$, $\mathbb{P}(U < u) = u$, and so $\mathbb{P}(X \geq x) = 1 - \sqrt{x}$. For continuous random variables, the distribution function and the density function are related to each other by integration, $\mathbb{P}(X \geq x) = \int_x^\infty f(y) dy$, so $f(x) = -F'(x)$. In this case, we end up with $f(x) = 1/(2\sqrt{x})$.

There is a universal way to generate a random variable given its distribution function, called the *inversion method*. (1) Generate a simple random variable $U \sim \text{Uniform}[0, 1]$. (2) Solve $F(X) = U$. (3) That's it, X has distribution function F . This plot shows why the method works:



it ensures that for every x the event $\{X \geq x\}$ is precisely the event $\{U \leq F(x)\}$, which has probability $F(x)$. Intuitively, in regions where the density f is high then the distribution F will be steep, and so X is more likely to hit those regions.

The inversion method requires us to solve $F(X) = U$, which is easy to do algebraically for simple continuous functions like the two-straight-line fit we found earlier. The method is also correct for discrete random variables, whose step functions are staircases, but here we usually want an algorithmic method for solving $F(X) = U$ rather than algebra. Section 2.5 looks at one special case. For a discrete distribution over a finite set of n outcomes, there is an obvious brute force algorithm that preprocesses the list of outcomes and then takes $O(\log n)$ to generate a random value; there is also an elegant algorithm called the *alias method* which takes $O(1)$ to generate a random value.

* * *

It's worth mentioning some terminology for describing distribution functions. The

<i>first quartile</i>	is a number x such that	$\mathbb{P}(X \leq x) = 25\%$
<i>median</i>	...	$\mathbb{P}(X \leq x) = 50\%$
<i>third quartile</i>	...	$\mathbb{P}(X \leq x) = 75\%$
<i>p-percentile</i>	...	$\mathbb{P}(X \leq x) = p\%$

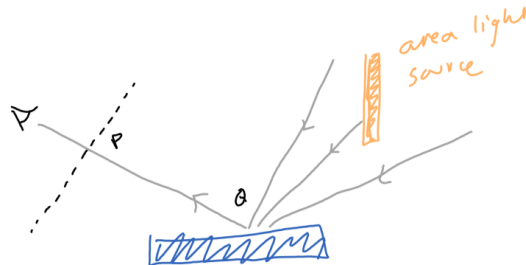
For discrete random variables it may not be possible to get exact percentiles, and there is no convention about rounding.

The range $[x_1, x_2]$ is called a *95% confidence interval* if $\mathbb{P}(x_1 \leq X \leq x_2) = 95\%$. Often we choose a two-sided confidence interval with $\mathbb{P}(X < x_1) = \mathbb{P}(X > x_2) = 2.5\%$. In some contexts it may be more useful to report a one-sided confidence interval, either $[x_1, \infty)$ or $(-\infty, x_2]$.

The *cumulative distribution function* is $\text{CDF}(x) = \mathbb{P}(X \leq x)$. What we've been calling the distribution is also called the *tail distribution function*, to disambiguate the two.

2.3. Limit theorems

Application. In computer graphics rendering and shading, we can compute the colour of a pixel on the screen by reasoning about light rays. First figure out the surface point Q that is to be shown at pixel P , by casting a ray from the camera through P and finding what surface it hits. Then figure out the colour and shading of Q by adding up all the light rays that might be illuminating it and reflecting out through P .



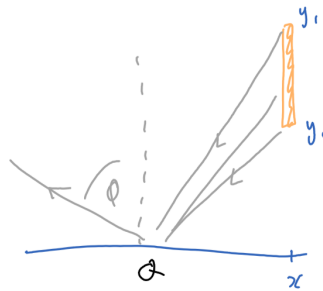
The surface might be perfectly reflective, or perfectly diffuse, or more generally we can model it with a specular lobe function $\text{BRDF}(\theta, \phi)$, which measures how much light is emitted at angle ϕ when it comes in at angle θ .



When we take into account the intensity of light glancing the surface as a function of angle θ , the total light reflected at angle ϕ , from a point light source of intensity I , is

$$I \cos(\theta) \text{BRDF}(\theta, \phi).$$

If illumination comes from an area light source, then we treat it as though the total intensity I is smeared across a set of point light sources:



$$\int_{y=y_0}^{y_1} \frac{I}{y_1 - y_0} \cos(\tan^{-1}(x/y)) \text{BRDF}(\tan^{-1}(x/y), \phi) dy.$$

* * *

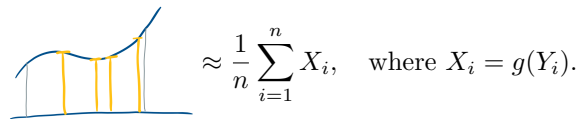
In a more abstract setting, suppose we want to compute

$$\int_{y=a}^b g(y) \frac{1}{b-a} dy.$$

The obvious method is to split the y range into n equally sized pieces, and approximate the function by a series of rectangles, e.g. taking the height of the rectangle to be the value of g at the midpoint.

$$\approx \frac{1}{n} \sum_{i=1}^n x_i, \quad \text{where } x_i = g\left(a + \frac{b-a}{n}(i-1/2)\right).$$

But there's actually nothing special about sampling g at grid points. Why not just pick the sampling points at random? In other words, pick n independent $\text{Uniform}[a, b]$ random variables Y_1, \dots, Y_n , and approximate



$$\approx \frac{1}{n} \sum_{i=1}^n X_i, \quad \text{where } X_i = g(Y_i).$$

This is called *Monte Carlo* integration. Is it any good, and if so why?

EXPECTATION

Let's calculate the expected value of the Monte Carlo approximation.

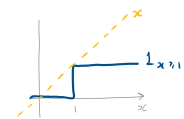
$$\begin{aligned} \mathbb{E}\left(\frac{1}{n} \sum_{i=1}^n X_i\right) &= \frac{1}{n} \sum_{i=1}^n \mathbb{E}X_i \quad \text{by linearity of expectation} \\ &= \frac{1}{n} n \mathbb{E}X_1 \quad \text{since they are IID} \\ &= \mathbb{E}g(Y_1) = \int_{y=a}^b \frac{1}{b-a} g(y) dy \quad \text{by definition of expectation.} \end{aligned}$$

So, the expected value is exactly the integral we want to compute. But this is just punting the question. What does the expected value have to do with anything?

WEAK LAW OF LARGE NUMBERS

Let's calculate the probability of error. Let $\mu = \mathbb{E}X_1$ be the integral we want to compute, and let $\varepsilon > 0$. Then

$$\begin{aligned} \mathbb{P}\left(\left|\frac{1}{n} \sum_{i=1}^n X_i - \mu\right| > \varepsilon\right) &= \mathbb{P}\left(\frac{(n^{-1} \sum X_i - \mu)^2}{\varepsilon^2} > 1\right) \quad \text{by simple algebra} \tag{6} \\ &= \mathbb{E}\left(\mathbb{1}\left[\frac{(n^{-1} \sum X_i - \mu)^2}{\varepsilon^2} > 1\right]\right) \quad \text{since } \mathbb{E}\mathbb{1}_A = \mathbb{P}(A) \\ &\leq \mathbb{E}\left(\frac{(n^{-1} \sum X_i - \mu)^2}{\varepsilon^2}\right) \quad \text{since } \mathbb{1}_{x \geq 1} \leq x \\ &= \frac{1}{\varepsilon^2} \text{Var}\left(\frac{1}{n} \sum X_i\right) \quad \text{by linearity of } \mathbb{E} \text{ and definition of Var} \\ &= \frac{1}{n^2 \varepsilon^2} n \text{Var} X_1 \quad \text{by linearity of Var, and independence} \\ &= \frac{1}{n} \frac{\text{Var} X_1}{\varepsilon^2} \\ &\rightarrow 0 \quad \text{as } n \rightarrow \infty. \end{aligned}$$



So, the more samples we use the better accuracy we get. This result is known as the *weak law of large numbers*. Another way to state it is in terms of the limit behaviour of the distribution function: for all $x \neq \mu$,

$$\mathbb{P}\left(\frac{1}{n} \sum_{i=1}^n X_i \geq x\right) \rightarrow \begin{cases} 1 & \text{if } x < \mu \\ 0 & \text{if } x > \mu \end{cases} \quad \text{as } n \rightarrow \infty.$$

Why is it called weak? The weak law, equation (6), says $\mathbb{P}(\text{err}_n > \varepsilon) \rightarrow 0$. There is also a strong law, $\mathbb{P}(\text{err}_n \rightarrow 0) = 1$. The strong law implies the weak law, but is harder to prove.

It's useful to know not only that there is convergence, but also how fast convergence happens. We just derived an upper bound on the probability of error, and it suggests that the accuracy depends on the variance of an individual sample. Is the bound tight?

THE CENTRAL LIMIT THEOREM

In Section 2.1 we saw a general purpose approximation:

$$X_1 \approx \text{Normal}(\mu, \sigma^2) \quad \text{where } \mu = \mathbb{E}X_1 \text{ and } \sigma^2 = \text{Var} X_1.$$

If we apply this approximation to all the X_i ,

$$\begin{aligned}\sum_{i=1}^n X_i &\approx \text{Normal}(n\mu, n\sigma^2) \\ \frac{1}{n} \sum_{i=1}^n X_i &\approx \text{Normal}\left(\mu, \frac{\sigma^2}{n}\right) \\ \frac{1}{n} \sum_{i=1}^n X_i - \mu &\approx \text{Normal}\left(0, \frac{\sigma^2}{n}\right) \\ \frac{n^{-1} \sum_{i=1}^n X_i - \mu}{\sqrt{\sigma^2/n}} &\approx \text{Normal}(0, 1).\end{aligned}$$

Another result, the Berry-Esseen theorem, gives a bound on the error in this limit statement.

It is a mathematical theorem that this approximation becomes increasingly accurate (assuming the X_i are independent and that μ and σ^2 are finite). The *central limit theorem* makes this precise, as a statement about convergence of the distribution function:

$$\mathbb{P}\left(\sqrt{n}\left(\frac{n^{-1} \sum_{i=1}^n X_i - \mu}{\sigma}\right) \geq x\right) \rightarrow \mathbb{P}(\text{Normal}(0, 1) \geq x) \quad \text{as } n \rightarrow \infty, \text{ for all } x.$$

I find it more helpful to remember the central limit theorem when it's written as an approximation,

$$\frac{1}{n} \sum_{i=1}^n X_i \approx \text{Normal}\left(\mu, \frac{\sigma^2}{n}\right).$$

We could use this to estimate how many samples we need for the Monte Carlo integration to reach a target level of accuracy, if we knew σ . It tells us that, with probability 95%, $n^{-1} \sum X_i$ is within $\pm 1.96\sigma$ of μ . We don't know the true σ , but we can just estimate it with Monte Carlo approximation!

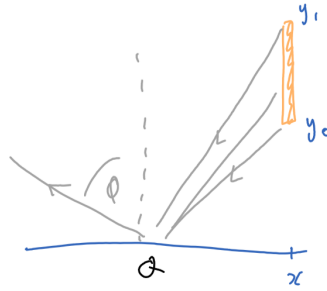
$$\sigma^2 = \mathbb{E}(X_1 - \mu)^2 \approx \frac{1}{n} \sum_{i=1}^n (X_i - \mu)^2$$

and also plug in the Monte Carlo approximation for μ . Our computer graphics code could keep generating more and more samples until it decides the error is small enough as to be imperceptible.

This should make us uneasy! How come it's OK to say "Plug in the Monte Carlo approximations to get a point estimate for σ ", and at the same time "Don't just use the plain Monte Carlo approximation for μ , use the central limit theorem to work out how accurate it is"? We will revisit this question in Section 3.

2.4. Importance sampling

In the computer graphics problem of Section 2.3, we studied how to use Monte Carlo integration to calculate the light intensity at a pixel.



The answer we want is

$$\int_{y=y_0}^{y_1} \frac{1}{y_1 - y_0} g(y) dy \quad (7)$$

where $g(y)$ is some complicated formula involving the specular characteristics of the surface and the angle of illumination. We approximated (7) by

$$\frac{1}{n} \sum_{i=1}^n g(Y_i)$$

where Y_1, \dots, Y_n are independent random variables drawn uniformly from the range $[y_0, y_1]$. In words,

$$\begin{array}{l} \text{reflected light} \\ \text{at angle } \phi \end{array} \approx \frac{1}{n} \sum_{i=1}^n \begin{array}{l} \text{light due to a simulated ray coming from} \\ \text{a random point } Y_i \text{ on the light source.} \end{array}$$

We found that the approximation error is of the order of σ/\sqrt{n} , where σ is the standard deviation of the answer from a single light ray, $\sigma = \text{std.dev}(g(Y_1))$. This suggests two questions. First, is there a way to change the simulation to reduce σ ? Second, if it's a highly reflective surface, then there isn't any need to simulate the entire light source, since only a few pieces of it will end up reflected at angle ϕ : how can we change the simulation to achieve this? It turns out that these two questions are asking exactly the same thing.



Let's try tweaking the simulated light rays. Physics tells us that the BRDF function is symmetrical. Consider picking a random point Y on the light source, but not uniformly at random: pick it instead so that angles closer to the center of the specular lobe are more likely. Let's also define $X = g(Y)f_0/f(Y)$, where $f(\cdot)$ is the density function for Y and $f_0 = 1/(y_1 - y_0)$ is the density function for Uniform $[y_0, y_1]$. This has expected value exactly equal to the integral (7) that we're after:

$$\mathbb{E}X = \int_{y=y_0}^{y_1} \left(\frac{g(y)f_0}{f(y)} \right) f(y) dy = \int_{y=y_0}^{y_1} \frac{1}{y_1 - y_0} g(y) dy$$

Where did the magic formula $X = g(Y)f_0/f(Y)$ come from? Previously we had used $g(Y)$. When Y is not uniformly distributed, $\mathbb{E}g(Y)$ isn't equal to (7)—it has an extra $f(y)$ term. We picked $X = g(Y)f_0/f(Y)$ simply to cancel out that extra $f(y)$.

We have designed a random variable X with $\mathbb{E}X$ equal to (7). So we can use the Monte Carlo method, i.e. approximate (7) by the average of a random sample from distribution X . In words,

$$\begin{array}{l} \text{reflected light} \\ \text{at angle } \phi \end{array} \approx \frac{1}{n} \sum_{i=1}^n \left(\begin{array}{l} \text{light due to a simulated ray coming from} \\ \text{a random point } Y_i \text{ drawn from density } f \end{array} / (f(Y_i)/f_0) \right).$$

See Section 2.2 for how to generate a random variable from an arbitrary distribution.

This is called *importance sampling*. Whatever distribution we choose for the Y_i , Monte Carlo integration will give us the right answer, and we have a whole design space of sampling distributions to choose from. Our aim is to choose a density function f to reduce $\sigma^2 = f_0^2 \text{Var}(g(Y)/f(Y))$.

* * *

It's surprisingly easy to design the optimal sampling distribution. From the definition of variance,

$$\text{Var} \frac{g(Y)}{f(Y)} = \mathbb{E} \left[\left(\frac{g(Y)}{f(Y)} - \mu \right)^2 \right] = \mathbb{E} \left[\left(\frac{g(Y)}{f(Y)} \right)^2 \right] - \mu^2 \quad \text{where} \quad \mu = \mathbb{E} \frac{g(Y)}{f(Y)}.$$

The whole point of importance sampling is that μ doesn't depend on how we choose f . The only term we have left to minimize is

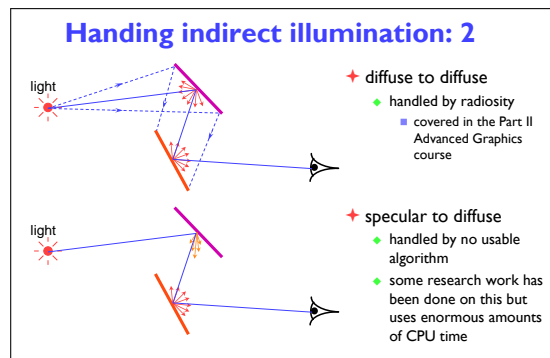
$$\mathbb{E} \left[\left(\frac{g(Y)}{f(Y)} \right)^2 \right] = \int \left(\frac{g(y)}{f(y)} \right)^2 f(y) dy = \int \frac{g(y)^2}{f(y)} dy$$

and it turns out (using some standard tools from optimization theory) that to minimize this we should pick a density function f such that $f(y)$ is proportional to $g(y)$.

Unfortunately, if we try to sample a random variable from $\text{const} \times g(y)$ using the inversion method, we have to integrate $g(y)$, which defeats the whole purpose of Monte Carlo integration!

Importance sampling is nonetheless useful as a heuristic. The full distributed ray tracing algorithm, taking account of indirect illumination, is "To work out the shading at a point Q , sample light rays by following them backwards; pick a random incoming angle at each bounce, and heuristically try to pick an angle in proportion to how much light is expected from that angle." No matter how bad the heuristic, importance sampling will give the right answer for large enough n . If the heuristic is good, it will give the right answer for small n .

This is why specular-to-diffuse lighting is tricky: the question "which angle is likely to give most illumination?" can't be answered with only local knowledge at the diffuse surface.²²



²²Slide from IA Introduction to Graphics.

2.5. The empirical distribution

The empirical distribution function of a dataset x_1, \dots, x_n is

$$\hat{F}(x) = \frac{1}{n} (\text{how many items there are } \geq x).$$

In Section 2.2 we looked at fitting: we started with a family of distribution functions $F_\theta(x)$ with some parameter or list of parameters θ , we picked specific parameter values $\hat{\theta}$ so that $\hat{F}(x) \approx F_{\hat{\theta}}(x)$, and then we used the inversion method to generate random variables from this fitted distribution. We were implicitly assuming that the empirical distribution function of a random sample should be close to the true distribution function from which the random sample was generated. Now, having learnt about limit theorems in Section 2.3, we can investigate this assumption.

Consider a random sample X_1, \dots, X_n , independent random variables with common distribution function $F(x) = \mathbb{P}(X_i \geq x)$. Let $\hat{F}_n(x)$ be the empirical distribution function. Then

$$\mathbb{E}\hat{F}_n(x) = \mathbb{E}\left(\frac{\sum_{i=1}^n 1_{X_i \geq x}}{n}\right) = \mathbb{E}1_{X_1 \geq x} = \mathbb{P}(X_1 \geq x) = F(x).$$

By the weak law of large numbers,

$$\mathbb{P}(|\hat{F}_n(x) - F(x)| > \varepsilon) \rightarrow 0 \quad \text{as } n \rightarrow \infty$$

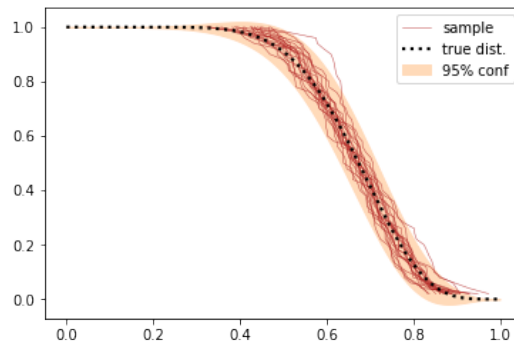
for any $\varepsilon > 0$. By the central limit theorem,

$$\hat{F}_n(x) \approx \text{Normal}\left(F(x), \frac{\sigma^2}{n}\right) \quad \text{where } \sigma^2 = \text{Var} 1_{X_1 \geq x} = F(x)(1 - F(x))$$

and so a 95% confidence interval for $\hat{F}_n(x)$ is

$$\mathbb{P}\left(\hat{F}_n(x) \text{ is in the range } F(x) \pm 1.96\sqrt{\frac{F(x)(1 - F(x))}{n}}\right) \approx 0.95.$$

Here is an illustration, 20 random samples of size 50 drawn from the Beta(10, 5) distribution.

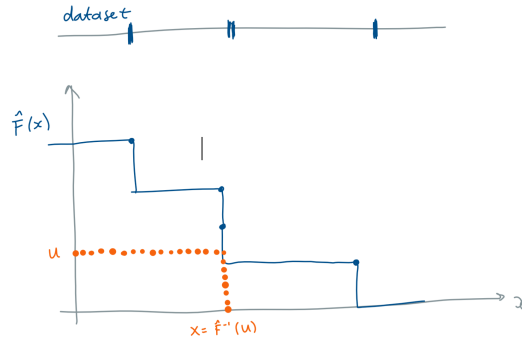


The Beta distribution arises in Bayesian inference about binary outcomes. It takes values in $[0, 1]$. For the density etc., look it up on Wikipedia.

RESAMPLING

When all we have is a dataset, how do we choose which family of distributions $F_\theta(x)$ to fit? Sometimes there are sound scientific reasons for choosing a particular family, and our goal is to integrate this background scientific knowledge with the dataset at hand. If there is no background science, then it's daft to use the data to fit a parameterized distribution function as we did in Section 2.2 when a *perfect* fit is staring us in the face, namely the empirical distribution itself! This is a perfectly valid distribution function: it starts at 1, and ends at 0, and is decreasing.

We can sample from the empirical distribution function using the inversion method—and a moment's thought tells us that this is exactly the same as picking a value at random from the dataset, each item in the dataset equally likely. The overall concept, that the best-fitting distribution for a dataset is nothing other than its empirical distribution, is called *resampling*. Conventionally, a superscript $*$ denotes a random variable drawn in this way from the empirical distribution.



Example 2.1 (Exact resampling). Given a dataset x_1, \dots, x_n , what are $\mathbb{E}X^*$ and $\text{Var} X^*$? From the definition of expectation,

$$\mathbb{E}X^* = \sum_{x \in \{x_1, \dots, x_n\}} x \mathbb{P}(X^* = x) = \sum_i x_i \frac{1}{n}$$

i.e. the sample mean, often written \bar{x} . From the definition of variance,

$$\text{Var} X^* = \sum_{x \in \{x_1, \dots, x_n\}} (x - \bar{x})^2 \mathbb{P}(X^* = x) = \sum_i (x_i - \bar{x})^2 \frac{1}{n}$$

i.e. the sample variance.

Example 2.2 (Monte Carlo resampling). I collected a dataset x_1, \dots, x_n and I found the sample mean \bar{x} . If I repeat the exercise and collect further datasets, how much variability should I expect to see in the sample mean?

The best-fitting distribution is the dataset itself, and the best we can do is assume that subsequent datasets will be drawn from the same distribution. In other words, the next time I collect data, I'll expect to see something like \bar{X}_n^* , the sample mean of n random values drawn from the empirical distribution. This is a random variable. It's impractical to do an exact calculations about the distribution of \bar{X}_n^* as we did in Example 2.1 because there are so many possible values that \bar{X}_n^* might take—but it's straightforward to use Monte Carlo approximation instead, to find e.g. $\mathbb{P}(\bar{X}_n^* \geq x)$ or to draw a histogram.

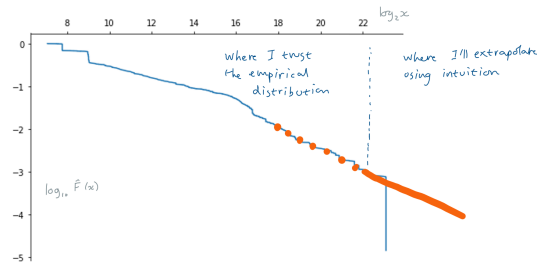
```

1 x = [13, 5, 2, ...] # the dataset
2 def sim_mean():
3     n = len(x)
4     X = random.choices(x, k=n) # k = number of samples to draw
5     return sum(X)/n
6
7 mc_samples = [sim_mean() for i in range(100000)]
8 matplotlib.pyplot.hist(mc_samples)

```

Example 2.3 (Parametric resampling). I collected a dataset x_1, \dots, x_n , and I found that the maximum value was m . If I repeat the exercise and collect further datasets, how much variability should I expect to see in the maximum? Resampling from the empirical distribution is unable to give an answer $> m$, but intuitively I feel that a new dataset might have larger values.

The real question here is: where does my intuition ‘larger values are possible’ come from, and how can I translate it into maths? Perhaps my intuition comes from seeing a straight line on a log-plot of the empirical distribution, as in Section 2.2. If this is the case, then I might construct a new semi-parametric distribution function, which starts with the empirical data and switches over at some point to a straight line, whose slope and intercept parameters are fitted from the data. Sampling from this semi-empirical distribution could potentially produce values $> m$.



* * *

When should we use parametric models and when should we use the dataset itself, in the form of the empirical distribution? There are no general rules.

- A dataset cannot tell us about values beyond the dataset. This has to come from our background knowledge or intuition. Integrating datasets and background knowledge is an art.
- A parametric distribution saves space: it only needs us to store a handful of parameters, rather than the full dataset. But this is often a premature optimization. For a small dataset of a few tens of thousands of values, on a modern computer, you should spend your time thinking about modeling and not about optimizing storage. For a large dataset, a model with a handful of parameters cannot hope to capture the richness of the data.
- Neural networks are parametric models. A neural network trained for simple image classification might take 140 million parameters, one for each connection in the network. The human brain has roughly 10^{15} connections, and a human lifetime is roughly 2.5×10^9 seconds. It seems that making sense of data is more about what you do with it than how you can compress it.
- High-dimensional modeling, i.e. modeling with more parameters than there are samples in the dataset, is an area of active research.

PARTICLE FILTERS

Application. Suppose we're trying to pinpoint a person's location, based on noisy GPS readings. First, imagine that the person is not moving, and that we simply want to draw a shape on a map to indicate where the person is likely to be. We could represent our uncertainty about the person's location by a probability density function, with $f_0(x)$ our initial belief (spread widely over a map region), and $f_t(x)$ our belief after t GPS readings. Using Bayes' rule,

$$f_t(x) = \frac{\mathbb{P}(r_t | x) f_{t-1}(x)}{\int_y \mathbb{P}(r_t | y) f_{t-1}(y) dy} \quad (8)$$

where r_t is the reading at time t . We could implement this for example by storing an array with the value of $f_t(x)$ at every location x on a fine enough mesh, and updating the entire array every timestep, replacing the integral by a sum over all points on the mesh.

To take account of motion, we could add dimensions to x for current velocity and current mode (walking, running, cycling); and we could change (8) to include the system dynamics—i.e. include terms describing how likely a new location is given the old location and velocity, and how likely a new velocity is given the old velocity and mode. The $f_t(\cdot)$ array would be gigantic.

If more sensors are available, e.g. the gyroscope and compass and accelerometer on a mobile phone, we could change (8) to include terms describing the chance of each of those readings given the present state. This is known as *sensor fusion*.

Instead of storing the density function $f_t(\cdot)$ as an array of values over mesh, why not approximate it by a random sample drawn from the distribution? This approach works, and it is called the *particle filter*. It involves finding an empirical equivalent of (8), i.e. an efficient way to generate a sample from $f_t(\cdot)$ given a sample from $f_{t-1}(\cdot)$ and an observation r_t .

Here is an illustration²³. It shows a cluster of particles representing the current belief about a person's position inside a building, and a line showing the path that the person actually followed. The readings are gyroscope and compass and accelerometer.



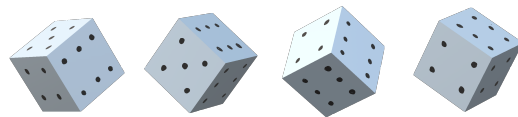
The message of this section is that samples and distributions are two sides of the same coin. In resampling, when we're faced with the problem that the true distribution is unknown, we can use the dataset instead. In the particle filter, when we're faced with the problem that the true distribution is intractable, we can use a sample instead.

²³Julian Straub. "Pedestrian Indoor Localization And Tracking Using A Particle Filter Combined with a Learning Accessibility Map". Bachelor thesis. Technische Universität München, 2010. URL: <http://people.csail.mit.edu/jstraub/pub/Pedestrian-Indoor-Localization-and-Tracking-using-a-Particle-Filter-combined-with-a-learning-Accessibility-Map/>. The author now works at Oculus.

3. Inference

Get experience of formulating questions about a dataset. Be able to assess the accuracy of your inferences, using Bayesian and frequentist methods. Be able to apply these methods for finding confidence intervals, conducting hypothesis tests, and making predictions.

Inference means reaching conclusions on the basis of data and reasoning. For example, if someone rolls a die and gets 6,4,6,6, what can we conclude about the chance that the next roll is a 6? Do we believe it's $\frac{1}{6}$ because that's how dice work? Do we conclude that this die is biased? Do we estimate from the data that the chance of a 6 is $\frac{3}{4}$, and are we confident enough to make a bet on odds of 3 to 1? How much more confident would we be if we saw the same frequencies from a million rolls of the die?



Inference is part science, part philosophy, part craft. The science is computation and probability theory; the philosophy is understanding what questions it is meaningful to ask, and thinking about what you want the answers for; the craft is being able to formulate questions in a way that makes the computing and maths tractable.

3.1. Quantifying a question

The UK Home Office makes available several datasets of police records, at data.police.uk. Here is a sample of rows from the stop-and-search dataset.

police force	operation	date-time object of search	lat	lng	gender	age	ethnicity	outcome
Hampshire	NA	2014-07-31T23:20:00 controlled drugs	50.93	-1.38	Male	25–34	Asian	nothing found
Hampshire	NA	2014-07-31T23:30:00 controlled drugs	50.91	-1.43	Male	34+	White	suspect summonsed
Hampshire	NA	2014-07-31T23:45:00 controlled drugs	51.00	-1.49	Male	10–17	White	nothing found
Hampshire	NA	2014-08-01T00:40:00 stolen goods	59.91	-1.40	Male	34+	White	nothing found
Hampshire	NA	2014-08-01T02:05:00 article for use in theft	50.88	-1.32	Male	10–17	White	nothing found

Suppose we want to investigate possible racial bias in policing. Are the police more likely to stop members of certain ethnic groups? The total number of stops in this dataset is

	Asian	Black	Mixed	Other	White
num. stops	79,492	163,856	350	18,480	483,472

Without knowing context, e.g. population breakdowns in the UK, or typical demographics of people in public spaces, this table is useless. Instead, let's look at the success rates for stop-and-searches. Label each row either **find** or **nothing** depending on the outcome of the search. The percentage of stop-and-searches that result in **find** is

	Asian	Black	Mixed	Other	White
% find	30.0	31.8	60.6	33.1	32.6

The probability of finding criminal activity is lower among Asian suspects, which means that the police are stopping relatively more non-criminals, which is an indicator of racial bias. But is this a significant difference, or is it within the bounds of random variation?

The starting point for quantifying uncertainty is a probabilistic model. Let Y_i be the outcome for row i of the dataset, either **find** or **nothing**, and let e_i be the ethnicity of the suspect. The simplest possible model is

$$\mathbb{P}(Y_i = \text{find}) = \beta_{e_i}$$

where β is a vector of probabilities, one per ethnic group. The maximum likelihood estimator is easy to calculate and reassuringly it turns out to be exactly what we would expect from the table: $\beta_{\text{Asian}} = 0.300$, $\beta_{\text{Black}} = 0.318$, etc.

Incorporating features It’s extremely unlikely that police behaviour is governed by only one feature in the data. For example, what if the police decision to stop someone is influenced by the suspect’s gender as well as ethnicity, and the gender breakdown is different in different ethnic groups?

	Asian	Black	Mixed	Other	White
% Male	96.9	95.2	93.7	93.5	89.4

If a police officer’s decision whether or not to stop someone is largely down to the suspect’s gender, and if police are relatively more likely to stop male suspects, might this be sufficient to account for the lower $\mathbb{P}(\text{find})$ among Asian suspects? To disentangle the two features, we can propose a model that takes account of both features simultaneously, e.g.

$$\mathbb{P}(Y_i = \text{find}) = \beta_{e_i} + \gamma_{g_i} \quad (9)$$

where g_i is the gender in row i of the dataset. This model allows the probability of find to depend on both ethnicity and gender. If it is indeed gender that is the dominant influence, and if different ethnic groups experience different $\mathbb{P}(\text{find})$ only because of their different gender breakdowns, then the model can accommodate this via $\beta_e = \text{const}$ for all e .

Natural parameters The model (9) has two problems. First, it has too many parameters: we could add 0.1 to every β coefficient and subtract 0.1 from both γ coefficients, and this change would leave absolutely every probability unchanged, and so it is impossible to identify the ‘correct’ values of the parameters. This issue is known as *non-identifiability*. A common trick is to rewrite the model as

$$\mathbb{P}(Y_i = \text{find}) = \alpha + \beta_{e_i} + \gamma_{g_i}, \quad \text{and require } \beta_{\text{Asian}} = \gamma_{\text{female}} = 0.$$

It doesn’t make any difference which reference levels we choose to set to 0; here I chose them alphabetically. We can unwrap this model:

$$\begin{aligned} \mathbb{P}(Y = \text{find for Asian female}) &= \alpha \\ \mathbb{P}(Y = \text{find for Asian male}) &= \alpha + \gamma_{\text{male}} \\ \mathbb{P}(Y = \text{find for Black female}) &= \alpha + \beta_{\text{Black}} \\ \mathbb{P}(Y = \text{find for Black male}) &= \alpha + \beta_{\text{Black}} + \gamma_{\text{male}} \\ &\dots \end{aligned}$$

The second problem with (9) is that it allows probabilities that are outside the range $[0, 1]$. We might fix this by changing to a model with explicit truncation,

$$\mathbb{P}(Y_i = \text{find}) = \max(0, \min(1, \alpha + \beta_{e_i} + \gamma_{g_i})).$$

This truncation turns out to be computationally awkward,²⁴ when we try to find maximum likelihood parameter estimates. A much better behaved model is

$$\mathbb{P}(Y_i = \text{find}) = \frac{e^{\xi_i}}{1 + e^{\xi_i}} \quad \text{where } \xi_i = \alpha + \beta_{e_i} + \gamma_{g_i}. \quad (10)$$

This is just an algebraic gimmick²⁵ that maps any real number $\xi \in (-\infty, \infty)$ to a value $e^\xi/(1+e^\xi)$ in the range $[0, 1]$. We can just plug the probability formula into a general-purpose

²⁴ What makes a model computationally awkward? Maximum likelihood estimation is based on optimization. Commonly, optimization libraries work best for functions that are differentiable, and where the partial derivatives are only zero at local optima, and where each argument is unconstrained i.e. permitted to take any floating point value. The model with explicit truncation has partial derivatives equal to zero over large parts of the parameter space.

²⁵ We often see medical results like “a Mediterranean diet halves your risk of heart attack”. There is usually a model behind this of the form $\mathbb{P}(\text{heart attack}) = e^{\xi + \mu d} / (1 + e^{\xi + \mu d})$ where ξ is made up of coefficients relating to other features such as age and gender and weight, μ is a coefficient for the feature “on Mediterranean diet”, and $d = 1$ if you follow that diet and 0 otherwise. This sort of study is usually done in populations where the risk of heart attack is fairly small, so the denominator is ≈ 1 . For those on the Mediterranean diet $\mathbb{P}(\text{heart attack}) \approx e^\mu e^\xi$ and for those not on it the probability is $\approx e^\xi$. So we can deduce from the headline that the study found the maximum likelihood estimator to be $\hat{\mu} = \log 1/2$. The study won’t report what the risk of heart attack was cut *from* or what it was cut *to*, since those numbers depend on ξ which depends on a person’s age and gender and weight and so on. The model says “Whatever your underlying risk, your risk would be roughly 50% lower if you were on a Mediterranean diet”.

unconstrained optimization routine, it finds the parameters that maximize the likelihood, and whatever parameters it finds we are guaranteed to end up with probabilities in $[0, 1]$. When we unwrap it,

$$\begin{aligned}\mathbb{P}(Y = \text{find for Asian female}) &= e^\alpha / (1 + e^\alpha) \\ \mathbb{P}(Y = \text{find for Asian male}) &= e^{\alpha + \gamma_{\text{male}}} / (1 + e^{\alpha + \gamma_{\text{male}}}) \\ \mathbb{P}(Y = \text{find for Black female}) &= e^{\alpha + \beta_{\text{Black}}} / (1 + e^{\alpha + \beta_{\text{Black}}}) \\ &\dots\end{aligned}$$

If for example $\gamma_{\text{male}} > 0$, then $\mathbb{P}(Y = \text{find})$ will be higher for male suspects than female suspects, across all ethnic groups. If we compute the maximum likelihood estimates and then unwrap them, we obtain

	Asian	Black	Mixed	Other	White
$\mathbb{P}_{\text{female}}(\text{find})$ %	29.7	31.5	46.0	33.2	32.6
$\mathbb{P}_{\text{male}}(\text{find})$ %	30.2	32.0	46.5	33.7	33.1

The model (10) is called a *logistic regression*. Logistic regression models are in widespread use, for example for estimating the probability that a web user will click on a certain ad. It's up to the data scientist to find good features to put into ξ , for example age and browsing history and purchase history and keywords in emails and location and everything else that a tech company might know about you, plus flashiness and screen size and keywords and everything else that distinguishes the ad.

* * *

We have studiously avoided the question of which model is *true*. The dataset almost certainly has so much richness that any simple parametric model we invent is wrong—but a wrong model can still be useful.

We formulated the logistic regression (10) to answer the question “What is the impact of the suspect’s ethnicity on police behaviour, taking account of gender?” The β that we estimate from the model lets us compare ethnic groups. If police behaviour is mainly determined by gender, then the β coefficients will be nearly all the same. If police behaviour is mainly determined by some other feature F that we haven’t included in the model, then the β coefficients will reflect the breakdown of F in each ethnic group. If ethnicity truly is an influence on police behaviour, then the β coefficients will tell us which ethnic groups have higher $\mathbb{P}(\text{find})$.

Parametric models are a way to ask questions about a dataset. They are one of the best tools we have for asking questions about the dataset, far more subtle than simply tabulating outcomes. But if you give them useless features, they will give useless answers. Garbage in, garbage out.

CONSTRUCTS / LATENT VARIABLES

A *latent variable* is a variable whose value is unobserved (latent means ‘hidden’). A latent variable can be something that must exist but we just don’t have data for, e.g. the true location of a smartphone user with a noisy GPS. It can also be a *construct*, i.e. a concept constructed in the mind of the data scientist. Latent variables are often useful for linking together different pieces of data, and for explaining our findings in intuitive language. Here is an application, a richer way to investigate possible racial bias in stop-and-search.

We set out to investigate whether a suspect’s ethnicity influences a police officer’s decision to conduct a stop-and-search. Our thought process was this:

Build a model for the probability of finding criminal activity, among suspects who were stopped. If the probability of finding criminal activity is lower for suspects in one particular ethnic group, this means that police are stopping more non-criminals in that ethnic group, which is an indicator of racial bias.

This is a weird model, because it is ‘causally backwards’. A person either is or isn’t engaged in criminal activity, and this is not an outcome of the police officer’s decision to conduct a stop-and-search, so it’s weird to build a model for $\mathbb{P}(\text{find})$ for suspects who were stopped.

What’s the difference between a parameter and a latent variable? If the number of unknown quantities grows with the size of your dataset, call them latent variables, otherwise call them parameters.

Seeing as we set out to investigate possible police, how might we use constructs to build a model that explicitly describes the police officer's action and includes a term for bias? Let's invent the construct 'shiftiness'. Let every person in a public space have a shiftiness latent variable s , a floating point number. Suppose it affects two things:

- The higher your shiftiness, the more likely you are to be engaged in criminal activity, e.g. $\mathbb{P}(\text{criminal}) = e^s / (1 + e^s)$.
- A police officer will stop you if your shiftiness is above a threshold, say if $s > \alpha + \beta_{\text{ethnicity}} + \gamma_{\text{gender}}$.

This is an invention. It's not trying to be a true measure of some objective feature in criminal psychology. It's just trying to summarize in a variable 'the aggregate of all the various factors and propensities that together affect a police officer's decision to stop-and-search a suspect' so that we can reason more naturally about police behaviour.

Microsoft's Xbox Live uses an invented construct for 'skill of a gamer'²⁶. There is a simple probability model: the probability that Player 1 wins against Player 2 in a game is a certain function of the difference in their skill levels. Given a dataset of (`player1_id`, `player2_id`, `winner`) with one record for every game, we can write out

$$\mathbb{P}(\text{dataset of all games and winners}) = f(\text{skills of every gamer})$$

and make inferences about each gamer's skill. The results are used to make sure that players are matched with other players of comparable ability.

It can be tricky to design a model with constructs, because of identifiability issues. In the Xbox system, we could add a constant to every single gamer's skill, and it would make no difference to the outcome probabilities. In the police example, we could add a constant to every shiftiness score, and add it to α also, and the distribution would be unchanged. In order to get useful answers out of models with constructs, we need to 'anchor' the values. Bayesian reasoning, described in the next section, is a good way to do this.

²⁶TrueSkill, described at <https://www.microsoft.com/en-us/research/project/trueskill-ranking-system> and the subject of an engaging and programmer-friendly blog post <http://www.moserware.com/2010/03/computing-your-skill.html>. The original paper: Ralf Herbrich, Tom Minka, and Thore Graepel. "TrueSkill™: A Bayesian Skill Rating System". In: *NIPS*. 2006. URL: <http://papers.nips.cc/paper/3079-trueskilltm-a-bayesian-skill-rating-system.pdf>.

3.2. Bayesianism

Data science is the process by which we change our beliefs about the world, in the light of data. There's no such thing as objective truth, there's only subjective degree of belief. One should represent belief by using a probability distribution, and one should update it using Bayes' rule.

From the policing data in Section 3.1 we used maximum likelihood estimation to estimate the difference in stop-and-search outcomes between different groups. The difference between male and female suspects was around 0.5% (to be precise, the maximum likelihood estimator was $\hat{\gamma}_{\text{male}} = 0.020711$). How confident should we be in this number? Intuitively, when there are many datapoints (673,541 male suspects, 60,600 female suspects in this case) we should be very confident. If we're looking at a narrower question, e.g. changes in gender bias in stop-and-search incidents in Cambridge city center from week to week, then there will be fewer datapoints and we should be less confident.

In this section we'll study one technique for quantifying confidence. We'll work with a toy dataset:

I have a coin, which might be biased. I toss it $n = 10$ times, and get $k = 9$ heads and $n - k = 1$ tails.

Let's model the data as

$$\text{num. heads} \sim \text{Binom}(n, p), \quad \text{i.e.} \quad \mathbb{P}(\text{num. heads} = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

Prior belief. The probability p of heads is unknown, and Bayesianism requires us to set down a prior belief for it. If we can't quantify our prior belief, Bayesianism says, then there are no grounds for us to draw conclusions. Let's invent, out of thin air, a prior belief that $P \sim \text{Uniform}[0, 1]$, which has density function $f(p) = 1$.

In what follows, we'll write $\Pr(P = p)$ for the density function of P . It's not a probability—since P is a continuous random variable, $\mathbb{P}(P = p) = 0$ for every p . The benefit of this notation rather than $f(p)$ is that Bayesian data science formulae often refer to many different random variables, and it's best to be explicit about which random variable a given density refers to. We'll use the same notation for discrete random variables, where $\Pr(X = x)$ genuinely is $\mathbb{P}(X = x)$. It's convenient to use the same notation for discrete and continuous random variables, because Bayes's rule applies the same to both of them.

Bayesian update. Bayes' rule tells us the density of p , conditional on the observed data. Let K be the random variable 'number of heads'. Then

$$\Pr(P = p \mid K = k) = \frac{\mathbb{P}(K = k \mid P = p) \Pr(P = p)}{\int_{q=0}^1 \mathbb{P}(K = k \mid P = q) \Pr(P = q) dq}.$$

This is called the *posterior distribution* for P . It's a density function, a function of p , so it must integrate to 1, and so it's convenient to gather up all the terms that don't involve p into a constant and then say "this constant must be whatever it takes to make the posterior density integrate to 1".

$$\begin{aligned} \Pr(P = p \mid K = k) &= \kappa \mathbb{P}(K = k \mid P = p) \Pr(P = p) \\ &= \kappa \binom{n}{k} p^k (1-p)^{n-k} \\ &= \kappa' p^k (1-p)^{n-k} \end{aligned}$$

where

$$\kappa' = 1 / \left(\int_{q=0}^1 q^k (1-q)^{n-k} dq \right).$$

This particular density function is the density function of the $\text{Beta}(k+1, n-k+1)$ distribution (look it up on Wikipedia). We don't even need to work out κ' , all we need to do is recognize the terms involving p .

Draw conclusions. We posed the question “What is the probability of heads?” A Bayesianist says that there’s no such thing as the objective bias of the coin, there’s only our belief, expressed by the posterior distribution $(P | K = k) \sim \text{Beta}(k + 1, n - k + 1)$. So let’s report a 95% confidence interval for $(P | K = k)$. A computer can work out the relevant points on the distribution function: run

```
lo,hi = scipy.stats.beta.ppf([0.025, 0.975], a=k+1, b=n-k+1)
```

and then

$$\mathbb{P}(P \in [\text{lo}, \text{hi}] | K = k) = 95\%.$$

We can report all sorts of quantities about the probability of heads. For example, our subjective belief that the coin is biased is

$$\mathbb{P}(P > 0.5 | K = k) = 1 - \text{scipy.stats.beta.cdf}(0.5, a=k + 1, b=n - k + 1).$$

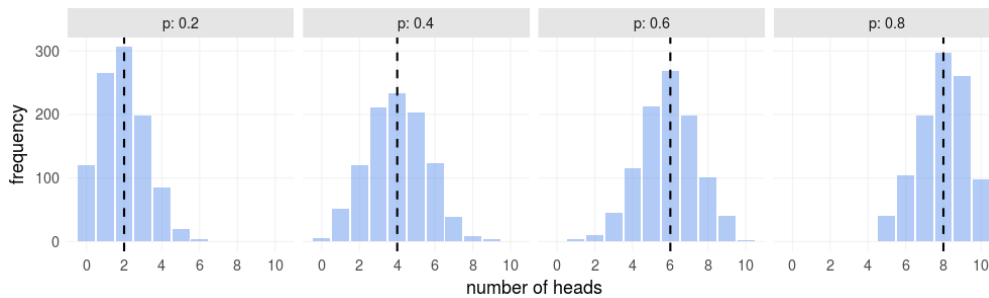
Nuisance parameters. If the problem has many parameters and we only want to report conclusions about one of them, we simply use Bayes’s rule to get a joint posterior distribution for all the parameters e.g. $\Pr(P = p, Q = q | K = k)$, and then use the law of total probability to find the marginal distribution of the parameter of interest, e.g. $\Pr(P = p | K = k) = \int_q \Pr(P = p, Q = q | K = k) dq$.

3.3. Frequentism

There is an objective world out there, with fixed but unknown parameters. By observing random phenomena, the data scientist can make inferences about those parameters

Given $k = 9$ heads out of $n = 10$ tosses of a coin, what is the probability of heads?

Worst-case procedures. The probability of heads, call it p , is fixed and unknown. We can't answer the question directly, and any range we propose for p might be right or wrong, we can't be sure. (Except for the range $[0, 1]$, which is always right and completely useless.) But *whatever* the value of p , simulations of $K \sim \text{Binom}(n, p)$ suggest we're likely to see K in the range $np \pm 2$.



The pivot. An exhaustive computation of $\mathbb{P}(|K - 10p| \leq 2)$ over all possible p shows that the lowest value it ever takes is 89%, at $p = 1/2$. We can *pivot* this probability statement:

$$\begin{aligned} & \mathbb{P}(|K - 10p| \leq 2) \geq 89\% && \text{for all } p && (11) \\ \Rightarrow & \mathbb{P}(-2 \leq K - 10p \leq 2) \geq 89\% && \text{for all } p \\ \Rightarrow & \mathbb{P}(K + 2 \geq 10p \geq K - 2) \geq 89\% && \text{for all } p \\ \Rightarrow & \mathbb{P}\left(p \in \left[\frac{K-2}{10}, \frac{K+2}{10}\right]\right) \geq 89\% && \text{for all } p. \end{aligned}$$

So, given that we saw 9 heads, can we conclude

$$\mathbb{P}(p \in [0.7, 1]) \geq 89\% ?$$

No. The parameter p is fixed and unknown, and it may be inside this range or it may be outside, and we don't know which. What we should really say is that the procedure

```
def confint(k): print(f"p is in [{max(k-2,0)/10}, {min(k+2,10)/10}]")
```

will print a true statement in at least 89% of coin-tossing trials, whatever the value of p , and in this particular trial it happens to print the range $[0.7, 1]$. This is usually abbreviated "An 89% confidence interval for p is $[0.7, 1]$ ".

Bootstrap resampling. Here is a general-purpose computational method, which removes any need for cleverness or exhaustive optimization for coming up with bounds like (11).

1. Start by writing out the probability you're interested in. Make sure it's a genuine probability, i.e. that there is a random variable inside.
2. Replace any unknown parameters by their maximum likelihood estimates given the data. Replace any random variables by their equivalents drawn from the empirical distribution. This rewritten expression is approximately equal to the probability from step 1.
3. Use the Monte Carlo method to estimate the probability of the expression in step 2.

This is called *bootstrap resampling*. 'Bootstrap' refers to the phrase 'pull yourself up by your bootstraps', in the sense that this method can give us probability answers without our having to even think up a model. 'Resampling' means drawing samples from the empirical distribution, the subject of Section 2.5.

Let's apply it to the problem at hand, 9 heads out of 10 tosses and we want to know the probability of heads. Step 1 says to write out a probability, and it takes some creativity to write down something useful. We'll see more examples in the rest of Section 3 and the example sheet. In this case, we want to make a confidence statement about p , the probability of heads, and a careful look at the pivot from equation (11) suggests that a statement about the maximum likelihood estimator \hat{p} is a useful starting point. Let's try

$$\mathbb{P}(\hat{p} \in [p - \delta, p + \delta]), \quad \text{where } \delta = 0.1. \quad (12)$$

Remember that \hat{p} is a function of the data, $\hat{p} = K/n$ in this case if we assume $K \sim \text{Binom}(n, p)$, and so the expression (12) is a genuine probability, as required by Step 1.

The next step is to replace terms. The maximum likelihood estimator (given the data) is $\hat{p} = k/n = 0.9$, so replace p by 0.9 in (12). The \hat{p} term is a random variable, $\hat{p} = K/n$, so replace it by its equivalent drawn from the empirical distribution, K^*/n . There are no definitive rules about how to do this; in Section 2.5 we saw three different approaches to resampling. The best way to resample is to ask ourselves "If this trial were run again, what is a good way to use the data at hand to synthesize a result that I might plausibly see?" A reasonable approach in this case is to let K^* be the number of heads in 10 values drawn at random from the observed sample, i.e. from 9 heads and 1 tail, which is $K^* \sim \text{Binom}(n, k/n)$. Putting all this together, we have obtained the expression

$$\mathbb{P}\left(\frac{K^*}{n} \in [k/n - \delta, k/n + \delta]\right), \quad \text{where } K^* \sim \text{Binom}(n, k/n). \quad (13)$$

The third step is to use the Monte Carlo method to estimate the probability (13). For $n = 10$, $k = 9$, $\delta = 0.1$, using 10,000 samples, I obtained the answer 92.8%.

```
1 n,k,delta = 10, 9, 0.1
2 Kstar = np.random.binomial(n, k/n, size=10000)
3 np.mean(np.logical_and(Kstar/n >= k/n-delta, Kstar/n <= k/n+delta))
```

Putting all these steps together, and pivoting expression (12) to emphasize p , we get

$$\mathbb{P}(p \in [\hat{p} - 0.1, \hat{p} + 0.1]) \approx 92.8\%, \quad [\hat{p} - 0.1, \hat{p} + 0.1] = [0.8, 1].$$

I have separated this into two separate statements. The probability statement on the left is about a procedure that we could run on any hypothetical dataset, and it uses \hat{p} to signify a random variable. The equality on the right is based on the actual value of \hat{p} that we get from the actual dataset.

Caveat programmer. Bootstrap resampling is a universal approximation technique. If you invent an unhelpful probability statement in step 1, or if you use a dodgy resampling method for step 2, you might end up with a useless answer. You always need to do a sanity check in your head and ask yourself "For the dataset and question at hand, is there any step in the approach I've taken that will likely give me nonsensical answers?" A data scientist keeps this question at the back of her mind, always. Meanwhile, it's a matter of research in theoretical statistics to find out which probability statements and resampling methods work robustly for which types of question.

3.4. Model selection

This is the topic of Example Sheet 2 questions 3 and 5, and also of Example Sheet 1 question 2.

3.5. Pragmatic inference

I hope this section leaves you uneasy. On one hand, a Bayesianist won't draw any conclusions at all without a prior—but where do we get prior beliefs from, if not data? On the other hand, a frequentist takes a straightforward question and produces such a contorted answer that you feel you need a hot shower to clean your mind afterwards. Is data science a house built on sand?²⁷

The pragmatic answer is that both approaches are different ways to account for uncertainty, and often in data science there are several different sources of uncertainty, and it's useful to be able to mix them.

Example (Bayesian hyperparameters). In the Bayesian approach to the coin question, I pick as my prior belief $P \sim \text{Beta}(\delta, \delta)$. This has the neat feature that the posterior belongs to the same family as the prior, $(P|K = k) \sim \text{Beta}(k + \delta, n - k + \delta)$. If $\delta = 1$ then the prior is uniform. But honestly I have no idea what δ should be. I declare δ to be a *hyperparameter*, which is a fancy way of saying “parameter that I don't have a prior for”, and I use non-Bayesian criteria to pick a value for it.

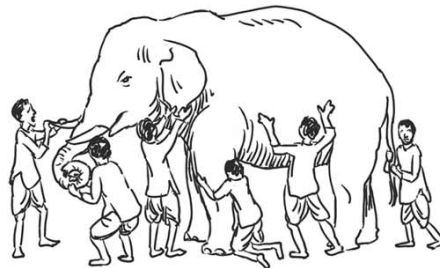
Example (Probability as an API). In the frequentist approach to the coin question, I work out that $[0, .8]$ is a 34% confidence interval, and $[0, .9]$ is a 74% confidence interval. I pass this information on to a Bayesian data scientist, who treats it like a distribution function, and uses it as a prior distribution for her next analysis. This doesn't make sense, but it gets the job done: I've expressed my uncertainty about the parameter, and she has incorporated uncertainty into her model. We are in effect using the language of probability as a communications API.

Sometimes there is prior data, e.g. someone has conducted a study of “typical bias in coins used in data science textbook illustrations”. A Bayesian data scientist might translate those observed frequencies directly into a prior distribution.

Example (Mixed effects modeling). I am analyzing data from a randomized controlled clinical trial, with some subjects taking active medication and some subjects on placebo. In this trial, each subject was assessed on ten visits to the clinic; the condition of patient i on visit j is $X_{i,j}$. I wish to know if there is a systematic difference between the two types of subject.

It's common that the measurements from a single individual are clustered together, so it's not useful to model all the $X_{i,j}$ as independent. Instead, I'll model them using a per-subject construct. Let patient i have a ‘wellness score’ $\Theta_i \sim \text{Normal}(\mu_{t_i}, \rho^2)$ where $t_i \in \{\text{active, placebo}\}$, and let $X_{i,j} \sim \text{Normal}(\Theta_i, \sigma^2)$ be independent given Θ_i . This model allows an individual subjects's measurements to be clustered tightly together (if σ is small), and it also allows for a systematic difference between the two types of subject (if $\mu_{\text{active}} \neq \mu_{\text{placebo}}$).

In this model, Θ_i is a parameter for $X_{i,j}$, and we are treating Θ_i as a random variable, which is what Bayesians do. But we can at the same time use maximum likelihood estimation and bootstrap resampling for μ and ρ and σ , like a frequentist. This is called *mixed effects modelling*. The Θ_i are called *random effects* and the other parameters are called *fixed effects*.



The final example is from work by Alan Turing and Irving Good on the Enigma ma-

²⁷Some great minds have gone down fruitless paths trying to understand inference. For an account of the history: Donald Gillies. *Philosophical theories of probability*. Routledge, 2000. And Ian Hacking. *The Emergence of Probability: A Philosophical Study of Early Ideas About Probability Induction and Statistical Inference*. 2nd ed. CUP, 2006.

chine²⁸. For each message, the German operator would choose a trigraph (sequences of three letters) from a book, the Kenngruppenbuch, which contained all possible trigraphs. The trigraph was used to initialize the wheel positions of the machine, after which the message could be encrypted. Each operator had his own copy of the Kenngruppenbuch, and marked every trigraph that he used and did not re-use it, though it might still be used by other operators. In order to tell the receiver which trigraph was being used, the operator encoded the trigraph using one of nine secret ‘digraph tables’, with a rule for which table to use on which day; the digraph tables were refreshed once a year or so. The operator would transmit this encoded version of the trigraph, and the receiver would use the digraph table to recover the trigraph. Every day, Bletchley Park had to guess which digraph table was in use that day. Turing devised a method for this, which relied on knowing the distribution of trigraphs. He found, for example, that trigraphs at the top of a page were more likely to be chosen. One step in the calculation was to estimate the probability that a previously unseen trigraph had been chosen. Turing never published his statistical work; it was left to Good to develop the ideas and publish them. Their estimation method is an example of what is now known as *empirical Bayesianism*. Extensions of this method are in use in linguistics (e.g. to estimate Shakespeare’s total vocabulary, based on the texts we have of his) and in ecology (to estimate species diversity, based on a sample).

Example (Empirical Bayesianism). I am catching butterflies. Each butterfly species i has frequency θ_i , so the probability that the next butterfly I catch belongs to species i is $\theta_i / \sum_j \theta_j$. What is the probability that the next butterfly I catch is of a species I haven’t seen before?

Let X_i be the number of butterflies I have seen so far of species i . Let’s model $X_i \sim \text{Poisson}(\theta_i)$. The Poisson random variable is a common modeling choice for discrete counts; its mean is $\mathbb{E}X_i = \theta_i$ and its density is $\mathbb{P}(X_i = x) = \theta_i^x e^{-\theta_i} / x!$. If we knew the θ_i , and we knew the total number of species n , then it would be easy to work out the probability of interest:

$$\mathbb{P}\left(\begin{array}{l} \text{next butterfly} \\ \text{is new species} \end{array}\right) = \frac{\sum_{i=1}^n \theta_i 1_{X_i=0}}{\sum_{i=1}^n \theta_i}. \quad (14)$$

But if we don’t know the θ_i and we don’t know n , what can we do?

Let’s adopt a Bayesian approach and treat the θ_i as random variables drawn independently from some common distribution, say with density function $g(\theta)$, and let Θ be a typical value, $\Pr(\Theta = \theta) = g(\theta)$, and let $X \sim \text{Poisson}(\Theta)$ be a typical count. Then the numerator of (14) is

$$\begin{aligned} \mathbb{E}\left(\sum_{i=1}^n \theta_i 1_{X_i=0}\right) &= n \mathbb{E}(\Theta 1_{X=0}) \\ &= n \mathbb{E}\left[\mathbb{E}(\Theta 1_{X=0} \mid \Theta)\right] \quad \text{by the law of total expectation} \\ &= n \mathbb{E}(\Theta e^{-\Theta}) = n \int_{\theta=0}^{\infty} \theta e^{-\theta} g(\theta) d\theta. \end{aligned}$$

This integral involves g and n , which we still don’t know. But there is a very clever trick:

$$\mathbb{E}\left(\sum_{i=1}^n 1_{X_i=1}\right) = n \mathbb{E}(1_{X=1}) = n \mathbb{E}[\mathbb{E}(1_{X=1} \mid \Theta)] = n \mathbb{E}(\mathbb{P}(X = 1 \mid \Theta)) = n \mathbb{E}(\Theta e^{-\Theta})$$

which suggests we approximate the numerator in (14) by $\sum_i 1_{X_i=1}$, i.e. the number of species for which we have seen exactly one butterfly. Using similar maths, we can approximate the denominator in (14) by the total number of samples we’ve seen, $\sum_i X_i$. Therefore,

$$\mathbb{P}\left(\begin{array}{l} \text{next butterfly} \\ \text{is new species} \end{array}\right) \approx \frac{\text{number of species we've seen once}}{\text{total number of butterflies seen so far}}.$$

What is remarkable in this example is that we used a genuine Bayesian model but without knowing the prior—and we don’t actually need to know the prior, because we can extract everything that matters about it from observed frequencies in the data. Large datasets of parallel situations ‘describe their own priors’.

²⁸I.J. Good. “Turing’s anticipation of empirical Bayes in connection with the cryptanalysis of the naval Enigma”. In: *Journal of Statistical Computation and Simulation* (2000). URL: <http://dx.doi.org/10.1080/00949650008812016>.

* * *

For a grand survey of how data science has been shaped by the interaction of Bayesian and frequentist thinking and by computing resources, see Efron and Hastie²⁹. They say

A good definition of a statistical argument is one in which many small pieces of evidence, often contradictory, are combined to produce an overall conclusion. In the clinical trial of a new drug, for instance, we don't expect the drug to cure every patient, or the placebo to always fail, but eventually perhaps we will obtain convincing evidence of the new drug's efficacy. The clinical trial is collecting direct statistical evidence, in which each subject's success or failure bears directly upon the question of interest. Direct evidence, interpreted by frequentist methods, was the dominant mode of statistical application in the twentieth century, being strongly connected to the idea of scientific objectivity.

Bayesian inference provides a theoretical basis for incorporating indirect evidence [...] The assertion of a prior density $g(\theta)$ amounts to a claim for the relevance of past data to the case at hand.

Empirical Bayes removes the Bayes scaffolding. In place of a reassuring prior $g(\theta)$, the statistician must put his or her faith in the relevance of the "other" cases in a large data set to the case of direct interest. [...]

The changes in twenty-first-century statistics have largely been demand driven, responding to the massive data sets enabled by modern scientific equipment. Philosophically, as opposed to methodologically, the biggest change has been the increased acceptance of indirect evidence, especially as seen in empirical Bayes and objective ("uninformative") Bayes applications.

Donald Rumsfeld, the former US Secretary of Defense, famously said³⁰

Reports that say that something hasn't happened are always interesting to me, because as we know, there are known knowns; there are things we know we know. We also know there are known unknowns; that is to say we know there are some things we do not know. But there are also unknown unknowns—the ones we don't know we don't know.

Bayesian calculations quantify uncertainty about parameters, and frequentist calculations quantify uncertainty about samples, which are both 'known unknowns'. Wrong models are the 'unknown unknowns'.

²⁹Bradley Efron and Trevor Hastie. *Computer age statistical inference: algorithms, evidence, and data science*. CUP, 2016. URL: <https://web.stanford.edu/~hastie/CASI/>.

³⁰U.S. Department of Defense news briefing, 12 February 2002, about the failure to find weapons of mass destruction in Iraq

4. Stochastic processes

Goals. Be able to formulate models for stochastic processes, and understand how they can be used for estimation and prediction. Be familiar with calculation techniques for memoryless stochastic processes. Understand the classification of discrete state space Markov chains, and be able to calculate the stationary distribution, and recognise limit theorems.

Science is often concerned with the laws that describe how a system changes over time, such as Newton's laws of motion. When we use probabilistic laws to describe how the system changes, the system is called a *stochastic process*. We used a stochastic process model in Section 1.2 to analyse Bitcoin; the probabilistic part of our model was the randomness of who generates the next Bitcoin block, be it the attacker or the rest of the peer-to-peer network. In Part II, you will come across stochastic process models in several courses:

- in *Computer Systems Modelling* they are used to describe discrete event simulations of communications networks
- in *Machine Learning and Bayesian Inference* they are used for computing posterior distributions
- in *Information Theory* they are used to describe noisy communications channels, and also the data streams sent over such channels.

4.1. Markov chains

Example 4.1. The Russian mathematician Andrei Markov (1856–1922) invented a new type of probabilistic model, now given his name, and his first application was to model Pushkin's poem *Eugeny Onegin*. He suggested the following method for generating a stream of text $C = (C_0, C_1, C_2, \dots)$ where each C_n is an alphabetic character:

```

1 alphabet = ['a', 'b', ...] # all possible characters incl. punctuation
2 next_char_prob = {('a', 'a'): [0, 0, .1, ...], ('a', 'b'): [.5, 0, ...]}
3 c = ['o', 'n'] # arbitrary starting string of length 2
4
5 while True:
6     p = next_char_prob[(c[-2], c[-1])] # the last two elements
7     c.append(random.choice(alphabet, weights=p))

```

As usual, we write C for the random variable and c for an actual value. Technically speaking, C is a function that returns an infinite sequence, and we ought to define it as a lazy list rather than writing out a non-terminating while loop.

In this code, `next_char_prob` is a dictionary where each value `p=next_char_prob[...]` is a vector of probabilities, and where `p[i]` is the probability that the next character is `alphabet[i]`.

We can measure `next_char_prob` for a piece of literature by looking at all trigrams i.e. sequences of three characters. Markov tabulated m -grams for several works by famous Russian authors, and suggested that the `next_char_prob` table might be used to identify an author.

Here is some Shakespeare generated in this method. The source is all of Shakespear's plays, with stage directions omitted, and converted to lowercase.

once. sen thery lost like kin ancry on; at froan, is ther page: good haves have emst upp'd ne kining, whows th lostruck-ace. 'llycur wer; hat behit mord. misbur greake, weave o'er, thousing i se to; ang shal spird

Here is some text generated with 5-grams rather than trigrams.

once is pleasurly. though the the with them with comes in hand. good. give and she story tongue. what it light, would in him much, behold of busin! how of ever to yearling with then, for he more riots annot know well.

Definition. A *Markov chain* is a sequence (X_0, X_1, X_2, \dots) where each X_n is a discrete random variable and

$$\mathbb{P}(X_{n+1} = x_{n+1} \mid X_0 = x_0, X_1 = x_1, \dots, X_n = x_n) = \mathbb{P}(X_{n+1} = x_{n+1} \mid X_n = x_n) \quad \text{for all } x_0, \dots, x_{n+1}. \quad (15)$$

(To be precise, the equation must hold for all x_0, \dots, x_{n+1} such that $\mathbb{P}(X_0 = x_0, \dots, X_n = x_n) > 0$, since otherwise the conditional probability isn't defined.) If equation (15) holds and furthermore the probability does not depend on n , i.e. if there is some matrix P such that

$$\mathbb{P}(X_{n+1} = y \mid X_n = x) = P_{xy}$$

then the process is called a *time-homogeneous Markov chain* with *transition matrix* P .

In IA *Discrete Mathematics* you learnt about finite automata. What is the relationship to Markov chains?

- Finite automata and Markov chains both have a set of possible states, and a lookup table / transition relation that describes progression from one state to the next.
- Finite automata are for describing algorithms that accept input, so the lookup table specifies ‘what happens next, based on the current state and the given input symbol?’ Markov chains are for describing systems that evolve by themselves, without input.
- Non-deterministic finite automata allow there to be several transitions out of a state, but they do not specify the probability of each transition, since they are intended to model ‘what are all the things that might happen?’ Markov chains do specify the transition probabilities, since they are intended to model ‘what are the things that typically happen?’
- Markov chains are allowed to have an infinite state space, e.g. the space of all integers. (They can even be defined with uncountable state spaces in which case X_n is a continuous random variable; the definition needs to be modified to refer to transition density functions rather than transition probabilities.)

The word *chain* means that the sequence $(X_n)_{n \geq 0}$ is indexed by an integer n . There are related definitions for continuous-time processes, and these will be used in Part II *Computer Systems Modelling*, but we will not study them further in this course.

In the Shakespeare example, the next character was chosen based on the previous *two* characters, which at first glance looks like it doesn't satisfy equation (15). The trick is to define X appropriately: in this case we should define $X_n = (C_n, C_{n+1})$. Then, the text generation rule can be rewritten as

```

1 x = [('o', 'n')] # arbitrary value for x[0]
2 c = x[0]
3
4 while True:
5     lastx = x[-1]
6     nextchar = random.choice(alphabet, next_char_prob[lastx])
7     nextx = (lastx[-1], nextchar)
8     x.append(nextx)
9     c.append(nextchar)

```

This way of writing the code makes it clear that X is a time-homogeneous Markov chain. The actual text C is a byproduct of X .

LAWS FOR CONDITIONAL PROBABILITY

All calculations with Markov chains make heavy use of conditional probability, so it's worth gathering together some results. Every law for plain probability still works if you just attach ‘given C ’ to every probability statement. For the laws in the following table, A and B and C are events, and X and Y are discrete random variables.

Definition of conditional probability

$$\mathbb{P}(A \cap B) = \mathbb{P}(A | B) \mathbb{P}(B) \qquad \mathbb{P}(A \cap B | C) = \mathbb{P}(A | B \cap C) \mathbb{P}(B | C)$$

Densities sum to 1

$$\sum_x \mathbb{P}(X = x) = 1 \qquad \sum_x \mathbb{P}(X = x | C) = 1$$

Law of total probability

$$\mathbb{P}(A) = \sum_x \mathbb{P}(A | X = x) \mathbb{P}(X = x) \qquad \mathbb{P}(A | C) = \sum_x \mathbb{P}(A | X = x, C) \mathbb{P}(X = x | C)$$

Definition of independence / conditional independence

$$\begin{aligned} \mathbb{P}(X = x, Y = y) &= \mathbb{P}(X = x) \mathbb{P}(Y = y) & \mathbb{P}(X = x, Y = y | C) &= \mathbb{P}(X = x | C) \mathbb{P}(Y = y | C) \end{aligned}$$

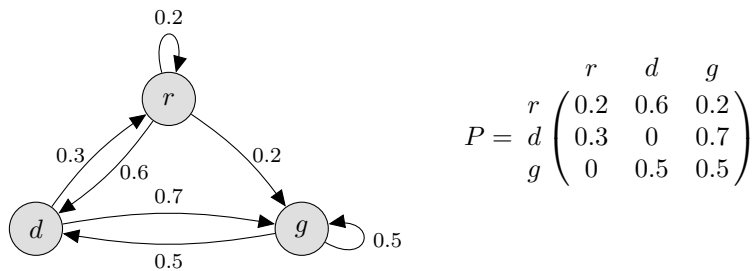
CALCULATIONS BASED ON MEMORYLESSNESS

Equation (15) says that if we know the present state X_n , then the past gives us no extra information about what will happen next. This is known as *memorylessness*, or as the *Markov property*. Most calculations with Markov chains revolve around conditioning on a previous step and then applying memorylessness. The general strategy for Markov problems (and indeed for almost any probability calculation) is

Draw the causal diagram that underlies the model. Write out a probability expression that you want to calculate, and identify the random variables whose distributions you want to derive. Add in values for any random variables that are their direct antecedents in the causal graph. Rewrite probabilities to have the form ‘probability of a value at one node of the causal diagram, given its immediate antecedents’.

Here are some examples where we apply this strategy. Look back at Section 1.2 to see how we applied it to the Bitcoin example.

Example 4.2 (Multistep transition probabilities). The winter weather in Cambridge varies from grey (g) to drizzle (d) to rain (r). Suppose that the weather changes from day to day according to the Markov chain drawn below. If it is grey today, what’s the chance that it will be grey three days from today?



The state transition diagram can also be written as a matrix P of transition probabilities. When you write out a Markov transition diagram or matrix, double-check that every row sums to 1, i.e. that all the total probability of all edges out of a node is equal to 1.

The question is asking us to calculate

$$\mathbb{P}(X_3 = g | X_0 = g).$$

The underlying mechanism of a Markov chain is ‘choose the next state based on the current state’, which we can draw as a causal diagram

$$X_0 \rightarrow X_1 \rightarrow X_2 \rightarrow X_3 \rightarrow \dots$$

We want to calculate something about the distribution of X_3 , and X_3 depends on X_2 , so we should introduce a value for X_2 . But X_2 depends on X_1 , so we should also introduce a value for X_1 . Finally, X_1 depends on X_0 , and the question tells us what X_0 is. Putting in all these extra values, via the law of total probability (conditional form),

$$\mathbb{P}(X_3 = g | X_0 = g) = \sum_{x_1, x_2} \mathbb{P}(X_3 = g | X_2 = x_2, X_1 = x_1, X_0 = g) \mathbb{P}(X_2 = x_2, X_1 = x_1 | X_0 = g).$$

There are two quite separate diagrams involved here: the state space diagram which shows transition probabilities between states; and the causal diagram which shows which random variables depend on which other random variables.

The first term has achieved exactly what we want: it's about X_3 , and it conditions on the antecedent X_2 , and the Markov property tells us that it simplifies:

$$\mathbb{P}(X_3 = g \mid X_0 = g) = \sum_{x_1, x_2} P_{x_2 g} \mathbb{P}(X_2 = x_2, X_1 = x_1 \mid X_0 = g).$$

The general strategy says we should rewrite the second term so that it has a random variable on the left and its immediate causal antecedents on the right. We can achieve this using the definition of conditional probability (conditional form),

$$\mathbb{P}(X_3 = g \mid X_0 = g) = \sum_{x_1, x_2} P_{x_2 g} \mathbb{P}(X_2 = x_2 \mid X_1 = x_1, X_0 = g) \mathbb{P}(X_1 = x_1 \mid X_0 = g).$$

Another application of the Markov property, and then rearranging terms, gives us an explicit formula for the probability we want,

$$\mathbb{P}(X_3 = g \mid X_0 = g) = \sum_{x_1, x_2} P_{x_2 g} P_{x_1 x_2} P_{g x_1} = \sum_{x_1, x_2} P_{g x_1} P_{x_1 x_2} P_{x_2 g}$$

which may be written in matrix form as $[P^3]_{gg}$. To compute it in Python,

```

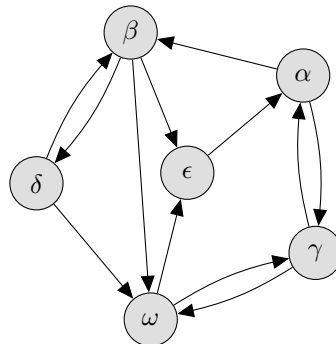
1 P = np.array([[0.2, 0.6, 0.2], [0.3, 0, 0.7], [0, 0.5, 0.5]])
2 assert all(P.sum(axis=1) == 1) # check row sums are all equal to 1
3 (P @ P @ P)[2,2] # compute P^3 then pick out element at [2,2]
4 np.linalg.matrix_power(P, 3)[2,2] # another way to compute P^3
5 # returns the answer: 0.505

```

Exercise 4.3 (Extended Markov property). Let X be a Markov chain. The Markov property, equation (15) says that if we know the present state X_n , then the past (X_0, \dots, X_{n-1}) gives us no extra information about the next step X_{n+1} . Prove that the same holds true further into the future, i.e. for any (x_0, \dots, x_{n+m}) ,

$$\begin{aligned} \mathbb{P}(X_{n+m} = x_{n+m}, \dots, X_{n+1} = x_{n+1} \mid X_n = x_n, \dots, X_0 = x_0) \\ = \mathbb{P}(X_{n+m} = x_{n+m}, \dots, X_{n+1} = x_{n+1} \mid X_n = x_n). \end{aligned}$$

Example 4.4 (Hitting probabilities). A web surfer starts at page α , and from each page picks an outgoing link at random from that page. What is the chance they hit ω before returning to α ?



Let X_n be the page that the web surfer is on after n clicks, $X_0 = \alpha$, and write X for the entire process $X = (X_n)_{n \geq 0}$. We want to calculate

$$\mathbb{P}\left(\begin{array}{l} X \text{ hits } \omega \text{ at some } n \geq 1 \\ \text{before hitting } \alpha \end{array} \mid X_0 = \alpha\right).$$

This is open-ended— X could first hit those two destinations at any $n \geq 1$ —so there's no clean way for us to condition on the entire path, as we did in Example 4.2. Instead, let's condition just on X_1 . Using the law of total probability (conditional form),

$$\begin{aligned} \mathbb{P}\left(\begin{array}{l} X \text{ hits } \omega \text{ at some } n \geq 1 \\ \text{before hitting } \alpha \end{array} \mid X_0 = \alpha\right) \\ = \sum_{x_1} \mathbb{P}\left(\begin{array}{l} X \text{ hits } \omega \text{ at some } n \geq 1 \\ \text{before hitting } \alpha \end{array} \mid X_1 = x_1, X_0 = \alpha\right) \mathbb{P}(X_1 = x_1 \mid X_0 = \alpha). \end{aligned}$$

The first term involves conditioning on both X_0 and X_1 , and the extended Markov property (Example 4.3) says that conditional on X_1 the future is independent of X_0 . Thus

$$\mathbb{P}\left(X \text{ hits } \omega \text{ at some } n \geq 1 \mid X_0 = \alpha\right) = \sum_{x_1} P_{\alpha x_1} \mathbb{P}\left(X \text{ hits } \omega \text{ at some } n \geq 1 \mid X_1 = x_1\right).$$

The final trick is to ‘reset the clock’. Let’s define

$$\pi_x = \mathbb{P}\left(X \text{ hits } \omega \text{ at some } n \geq 0 \mid X_0 = x\right).$$

It doesn’t make any difference whether we start measuring time from $n = 0$ or from $n = 1$, since the process follows the same dynamics regardless, thus

$$\mathbb{P}\left(X \text{ hits } \omega \text{ at some } n \geq 1 \mid X_0 = \alpha\right) = \sum_x P_{\alpha x} \pi_x \quad (16)$$

and furthermore (by repeating the entire conditioning argument we have just been through)

$$\pi_x = \sum_y P_{xy} \pi_y \quad (17)$$

and clearly $\pi_\alpha = 0$ and $\pi_\omega = 1$. Rewriting equations (16) and (17) as matrix equations, π is a 6-dimensional vector with elements in $[0, 1]$ and

$$\mathbb{P}\left(X \text{ hits } \omega \text{ at some } n \geq 1 \mid X_0 = \alpha\right) = [P\pi]_\alpha, \quad \pi = P\pi.$$

In Python,

```

1 # States are in the order [\alpha, \beta, \gamma, \delta, \epsilon, \omega]
2 # Set up an adjacency matrix for the graph,
3 # then scale it so rows sum to 1 (and assert they do)
4 P = np.array([[0,1,1,0,0,0], [0,0,0,1,1,0], [1,0,0,0,0,1],
5              [0,1,0,0,0,1], [1,0,0,0,0,0], [0,0,1,0,1,0]])
6 P = P / P.sum(axis=1)[:, np.newaxis]
7 assert all(P.sum(axis=1) == 1)
8 # We want to solve P.\pi = \pi i.e. (P-I).\pi = 0, and also \pi[0]=0 and \pi[5]=1
9 # Bundle all the equations together in a matrix, and solve with np.linalg.lstsq
10 A = np.concatenate((P - np.eye(6), [[1,0,0,0,0,0], [0,0,0,0,0,1]]))
11 \pi = np.linalg.lstsq(A, [0,0,0,0,0,0, 0,1])[0]
12 # Return the hitting probability we wanted to calculate
13 (P @ \pi)[0]
14 # returns the answer: 0.4300

```

MEMORY LENGTH

The rule we used to generate pseudo-Shakespeare, “pick the next character based on the preceding m ”, produces better-looking results for larger m —but the larger m is, the more storage space we need for the lookup table, and the fewer $(m + 1)$ -grams we have with which to estimate frequencies. If m gets even larger, the algorithm can’t do much more than regurgitate the input text on which it was trained.

Neural networks can be used to get around these limitations: they can learn how much information from preceding elements in the sequence should be incorporated into the state of the Markov chain, and they’re not limited to fixed- m state descriptor. Here is an example of Shakespeare generated using a neural network rather than trigram frequencies³¹.

³¹Andrej Karpathy, *The unreasonable effectiveness of recurrent neural networks*, May 2015, <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>. He writes “There’s something magical about Recurrent Neural Networks (RNNs) ... We’ll train RNNs to generate text character by character and ponder the question ‘how is that even possible?’ ”

PANDARUS:

*Alas, I think he shall be come approached and the day
When little strain would be attain'd into being never fed,
And who is but a chain and subjects of his death, I
should not sleep.*

Second Senator:

*They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.*

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

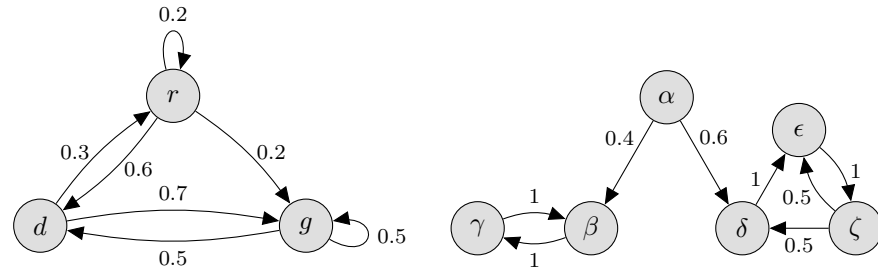
4.2. Estimation in a hidden Markov model

Example. A person is moving about. At each timestep, we receive a noisy GPS reading of their current location. How can we estimate their current location?

This is the topic of Example Sheet 3.

4.3. Limit theorems

When analysing Markov chains, it's often useful to be able to ask about their long-run average behaviour. We asked the same question in Section 2 about sums of independent random variables. Markov chains however have a richer range of possible behaviours, and it turns out there are three separate ways to ask ‘what is average behaviour?’ To reduce the mathematical overhead we will restrict attention in this section to time-homogeneous Markov chains with a finite state space (though most of the results also hold for infinite state spaces). We will illustrate with two examples, the Markov chain for Cambridge weather from Section 4.1, and a pathological case.



4.3.1. STATIONARY BEHAVIOUR

Definition. A Markov chain is said to be *stationary* if its distribution does not change over time, i.e. if there is a vector π such that $\mathbb{P}(X_n = x) = \pi_x$ for all n . Conversely, if π is a probability distribution such that

$$\mathbb{P}(X_0 = x) = \pi_x \text{ for all } x \quad \implies \quad \mathbb{P}(X_n = x) = \pi_x \text{ for all } x \text{ and } n$$

then π is called a *stationary distribution* or *equilibrium distribution*.

The word ‘stationary’ does not mean that the Markov chain has somehow stopped—a Markov chain is defined to go on forever, always stepping randomly from state to state. It is the *distribution* that is stationary i.e. unchanging.

If π is a stationary distribution, and we pick the Markov chain's initial state X_0 randomly according to π , then X_1 will have distribution π and so will X_2 and so on, i.e. the chain itself will be stationary. If we pick the initial state in some other way, it's typically not the case that X_1, X_2 etc. have distribution π —but it turns out that stationary distributions are still useful for understanding the long-run behaviour of the chain, as we will see in the rest of Section 4.3

We can find a stationary distribution using the same sort of calculations based on memorylessness that we used in Section 4.1. If X is a stationary Markov chain then

$$\mathbb{P}(X_n = x) = \sum_y \mathbb{P}(X_n = x \mid X_{n-1} = y) \mathbb{P}(X_{n-1} = y) \quad \text{for all } x, n$$

hence a stationary distribution π must satisfy

$$\pi_x = \sum_y \pi_y P_{yx} \quad \text{for all } x \tag{18}$$

where P is the transition matrix. For the Cambridge weather Markov chain, writing out in longhand the equations from (18),

$$\begin{aligned} \pi_r &= 0.2\pi_r + 0.3\pi_d \\ \pi_d &= 0.6\pi_r + 0.5\pi_g \\ \pi_g &= 0.2\pi_r + 0.7\pi_d + 0.5\pi_g. \end{aligned}$$

Although there are three equations and three unknowns, when we try to solve them we find there is not a unique solution: if the vector π is a solution then so is $\kappa\pi$ for any constant κ . To pin π down we need an extra equation, an equation that comes from the fact that π is a probability distribution:

$$\sum_x \pi_x = 1.$$

Rather than solve all these simultaneous equations with algebra, we can turn them into matrix notation and then ask the computer to solve them. Equation (18) becomes $\pi = \pi P$, or equivalently $(P - I)^\top \pi = 0$. The normalizing equation is $1^\top \pi = 1$. In Python,

```
1 P = np.array([[0.2, 0.6, 0.2], [0.3, 0, 0.7], [0, 0.5, 0.5]])
2 A = np.concatenate(((P - np.eye(3)).transpose(), [[1, 1, 1]]))
3 pi = np.linalg.lstsq(A, [0, 0, 0, 1])[0]
```

In `numpy`, if π is a one-dimensional array then it can be used either as a row vector or a column vector. In πP it is treated as a row vector, and in $(P - I)^\top \pi$ it is treated as a column vector.

We can compute a stationary distribution for the pathological Markov chain using exactly the same method, but there is a problem: equation (18) has multiple solutions, even after imposing the extra equation $\sum_x \pi_x = 1$. If we just write out all the equations longhand,

$$\begin{aligned}\pi_\alpha &= 0 \\ \pi_\beta &= 0.4\pi_\alpha + \pi_\gamma \\ \pi_\gamma &= \pi_\beta \\ \pi_\delta &= 0.6\pi_\alpha + 0.5\pi_\zeta \\ \pi_\epsilon &= \pi_\delta + 0.5\pi_\zeta \\ \pi_\zeta &= \pi_\epsilon \\ \pi_\alpha + \pi_\beta + \pi_\gamma + \pi_\delta + \pi_\epsilon + \pi_\zeta &= 1\end{aligned}$$

and solve these equations simultaneously, we discover that the general solution is

$$[\pi_\alpha, \pi_\beta, \pi_\gamma, \pi_\delta, \pi_\epsilon, \pi_\zeta] = a [0, 1/2, 1/2, 0, 0, 0] + (1 - a) [0, 0, 0, 1/5, 2/5, 2/5] \quad (19)$$

for any real value a (though only $a \in [0, 1]$ will yield a legitimate probability distribution). In Python, if we look carefully at the output of `np.linalg.lstsq()` and read the documentation, we see it telling us that the linear equation does not have a unique solution; there are further `np.linalg` tools that can extract the general form of the solution.

Equation (19) actually has a nice intuitive explanation. The Markov chain could be spending all its time in states $\{\beta, \gamma\}$ with stationary distribution $[1/2, 1/2]$, or it could be spending all its time in states $\{\delta, \epsilon, \zeta\}$ with stationary distribution $[1/5, 2/5, 2/5]$.

Theorem (uniqueness of stationary distribution). Consider a Markov chain with transition matrix P and a finite state space. The Markov chain is called *irreducible* if it is possible to get from any state to any other. If the Markov chain is irreducible, then there is a unique stationary distribution, and it is the unique solution π to

$$\pi = \pi P, \quad \pi^\top \mathbf{1} = 1. \quad (20)$$

Example. The Cambridge weather Markov chain can get from any state to any other state; to get from g to r takes two steps, and all the others can be achieved in one step. Therefore it is irreducible, therefore it has a unique stationary distribution.

The pathological Markov chain is not irreducible, because it is impossible to get from β to α .

4.3.2. DETAILED BALANCE

Often, when we want to find the stationary distribution, there's nothing for it but to use `np.linalg` and solve a matrix equation. In some special cases the Markov chain has a form that lets us find the stationary distribution with very little algebra. This seems like a curiosity, not worth mentioning in a data science course—except that there is a clever trick for generating random variables from general Bayesian posterior distributions that relies on exactly this special case. The clever trick is called Gibbs sampling, and it is taught in Part II *Machine Learning and Bayesian Inference*.

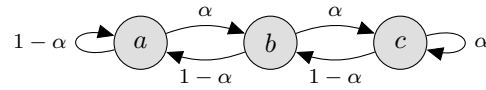
Theorem (detailed balance). Let X be a Markov chain with transition matrix P . If there is a vector π such that

$$\pi_x P_{xy} = \pi_y P_{yx} \quad \text{for all states } x \text{ and } y \quad (21)$$

then π solves $\pi = \pi P$. Equation (21) is called the *detailed balance* condition. This theorem is trivial to prove: just write out (18) and substitute in (21).

If the chain is irreducible, then the theorem of Section 4.3.1 tells us that there is a unique stationary distribution. If we have found a distribution π that solves the detailed balance condition, then π must be that unique stationary distribution.

Exercise 4.5. Calculate the stationary distribution of the following Markov chain.



Is it irreducible? Actually, if $\alpha = 0$ or $\alpha = 1$ then the chain is not irreducible: if $\alpha = 0$ then it gets stuck in state a , so the stationary distribution is $\pi_a = 1$, $\pi_b = \pi_c = 0$. If $\alpha = 1$ then it gets stuck in state c , so the stationary distribution is $\pi_a = \pi_b = 0$, $\pi_c = 1$.

In the case $0 < \alpha < 1$, it's easy to see that it's possible to get from any state to any other. Therefore the theorem of Section 4.3.1 applies, and so there is a unique stationary distribution. It never hurts to try to solve the detailed balance equations; either we find the stationary distribution without much work, or we quickly discover that they can't be solved and we have to solve the full equations (20). In this case, the detailed balance equations are

$$\begin{aligned} \text{for } (a, b) \text{ and } (b, a): \quad & \pi_a \alpha = \pi_b (1 - \alpha) \\ \text{for } (a, c) \text{ and } (c, a): \quad & \pi_a 0 = \pi_c 0 \\ \text{for } (b, c) \text{ and } (c, b): \quad & \pi_b \alpha = \pi_c (1 - \alpha) \\ \text{for } (a, a) \text{ etc.:} \quad & \pi_a (1 - \alpha) = \pi_a (1 - \alpha) \text{ etc.} \end{aligned}$$

and they have the solution

$$\pi_b = \pi_a \frac{\alpha}{1 - \alpha}, \quad \pi_c = \pi_a \left(\frac{\alpha}{1 - \alpha} \right)^2.$$

Putting in the constraint $\pi_a + \pi_b + \pi_c = 1$, we get

$$[\pi_a, \pi_b, \pi_c] = \frac{1}{1 + \alpha/(1 - \alpha) + \alpha^2/(1 - \alpha)^2} \left[1, \frac{\alpha}{1 - \alpha}, \left(\frac{\alpha}{1 - \alpha} \right)^2 \right].$$

Exercise 4.6 (Random walk on an undirected graph). A knight moves on an otherwise empty chessboard, each timestep picking one of its legal moves at random (out of 8 legal moves if it is in the center of the board, and 2 legal moves if it is in a corner). Show that the stationary probability of being in position x is $m_x/336$, where m_x is the number of legal moves out of position x .

We should first check whether the Markov chain described in the question is irreducible, since otherwise there isn't even a unique stationary distribution. This is just a matter of sketching a chessboard and persuading ourselves that a knight can indeed get from any position to any other position, given enough moves.

The question tells us the stationary distribution and asks us to verify it. We could plug it into the full equations (20), but if it happens to solve the detailed balance equations then that is sufficient and our work will be simpler. The detailed balance equations are

$$\begin{aligned} \frac{m_x}{336} \times \frac{1}{m_x} &= \frac{m_y}{336} \times \frac{1}{m_y} \quad \text{if } x \leftrightarrow y \text{ is legal,} \\ \frac{m_x}{336} \times 0 &= \frac{m_y}{336} \times 0 \quad \text{if } x \leftrightarrow y \text{ is illegal.} \end{aligned}$$

These equations are certainly true, and they are the only equations that need to be satisfied, since $x \rightarrow y$ is legal if and only if $y \rightarrow x$ is legal. Therefore the suggested distribution solves detailed balance.

Finally, we need to verify that the suggested distribution is indeed a distribution, i.e. that it sums to 1. Counting the number of possible moves from every position on the chessboard gives a total of 336, thus $\sum_x m_x/336 = 1$.

It's easy to check that the result described here can be generalised to a random walk on any undirected graph.

4.3.3. ERGODIC THEOREM

Theorem (ergodicity). Let X be an irreducible Markov chain with stationary distribution π . Then the long-run average of time spent in each state converges to π . Mathematically,

$$\mathbb{E}\left(\frac{1}{n}\sum_{i=1}^n 1_{X_i=x}\right) \rightarrow \pi_x \quad \text{as } n \rightarrow \infty, \text{ for all states } x. \quad (22)$$

If the Markov chain's initial state X_0 were chosen from distribution π , then we know from Section 4.3.1 that X_n would have distribution π for every n , thus $\mathbb{E}1_{X_i=x} = \mathbb{P}(X_i = x) = \pi_x$ for all i , and so (22) would be true exactly, no need for a limit. What's remarkable is that the theorem holds regardless of how the initial state is chosen.

Example. Consider the pathological Markov chain, starting at $X_0 = \alpha$. This chain is not irreducible, so the ergodic theorem doesn't apply directly. But we can still say how the chain behaves: with probability 0.4 it jumps to $X_1 = \beta$, and thereafter it behaves just like an irreducible chain on $\{\beta, \gamma\}$ and spends half its time in each of those two states; or with probability 0.6 it jumps to $X_1 = \delta$, and thereafter it behaves just like an irreducible chain on $\{\delta, \epsilon, \zeta\}$ and spends roughly 20% of its time in δ , 40% in ϵ , and 40% in ζ .

4.3.4. LIMITING BEHAVIOUR

In the Cambridge weather Markov chain, the ergodic theorem tells us that the long-run fraction of rainy days is equal to π_r , where π is the stationary distribution. So we'd expect that, if we pick a day arbitrarily, the probability of rain is π_r . This works for the Cambridge weather example, but it doesn't always work... the caveat is illustrated by states β and γ in the pathological Markov chain: if the chain starts in $X_0 = \beta$ then $X_n = \beta$ for even n and $X_n = \gamma$ for odd n , and so we can't make a blanket claim about 'typical X_n '. The following theorem gives a general condition under which time-averages correspond to typical values.

Theorem. Let X be a Markov chain. A state x is said to be *aperiodic* if there exists an n_0 such that $\mathbb{P}(X_n = x \mid X_0 = x) > 0$ for all $n \geq n_0$. If the chain is irreducible and has an aperiodic state, then all its states are aperiodic, and furthermore

$$\mathbb{P}(X_n = x \mid X_0 = y) \rightarrow \pi_x \quad \text{as } n \rightarrow \infty, \text{ for all states } x \text{ and } y.$$

Note that $\mathbb{P}(X_n = x \mid X_0 = y) = [P^n]_{xy}$ where P is the transition matrix, according to our calculations in Example 4.2.

This is most useful as a tool for generating a random variable from distribution π . In many applications, from statistical physics to Bayesian inference, we want to be able to generate a random variable from a distribution π that we can't even write out explicitly. Suppose we can cunningly devise transition probabilities of a Markov chain to ensure that it has stationary distribution π . Then we can generate a random variable from distribution π just by starting the Markov chain in an arbitrary state, and running it for a large number of steps n , and returning the state X_n .

Exercise. Consider the three-state Markov chain consisting of states δ , ϵ , and ζ from the pathological Markov chain. Show that all three states are aperiodic.

First we verify that it is irreducible. We can get from any state to any other by following links $\delta \rightarrow \epsilon \rightarrow \zeta \rightarrow \delta \rightarrow \dots$, so yes it is irreducible.

For aperiodicity, let's pick one state arbitrarily, say δ , and work out if that state is aperiodic. The theorem says that if one state is aperiodic then all states are aperiodic. Can we get from δ to δ in n steps?

n	can get from δ to δ in n steps?
1	no
2	no
3	yes, $\delta \rightarrow \epsilon \rightarrow \zeta \rightarrow \delta$
4	no
5	yes, going around the loop $\epsilon \rightarrow \zeta \rightarrow \epsilon$ once
6	yes, going around the loop $\delta \rightarrow \epsilon \rightarrow \zeta \rightarrow \delta$ twice
7	yes, using two loops $\epsilon \rightarrow \zeta \rightarrow \epsilon$
8	yes, since $8 = 5 + 3$ and 5 and 3 are possible
9	yes, $3+3+3$
$n \geq 8$	yes, by mixing loops of length 5 and length 3

(This is related to IA *Discrete Mathematics*. We can go from δ to δ in 3 steps, and in 5 steps. The greatest common divisor of 3 and 5 is 1, therefore there is an integer linear combination equal to 1, in this case $2 \times 3 - 1 \times 5 = 1$. We can achieve any $n = 5m$ by m copies of the 5-step loop, and we can achieve $n = 5m + l$ by adding l copies of the $2 \times 3 - 1 \times 5$ widget.)

5. Feature spaces

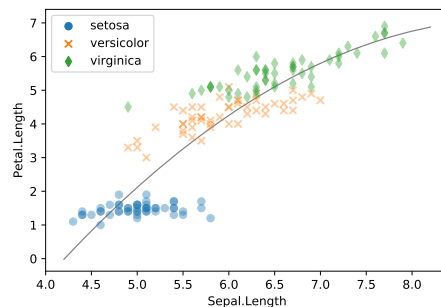
Goals. Refresh your memory of IA Maths for NST, where you were taught about linear spaces, bases, inner products, and projections. Understand the link between linear regression and inner products. Get experience of feature engineering.

In data science, a *feature* is any measurable property of the objects being studied. A *linear model* is a model with unknown parameters in which the parameters are weighted by features and combined linearly. Here’s a very simple example.

The Iris dataset was collected by the botanist Edgar Anderson and popularized³² by Ronald Fisher in 1936. Fisher has been described as a “genius who almost single-handedly created the foundations for modern statistical science”. The dataset consists of 50 samples from each of three species of iris, each with four measurements.

Petal length	Petal width	Sepal length	Sepal width	species
1.0	0.2	4.6	3.6	setosa
5.0	1.9	6.3	2.5	virginica
5.8	1.6	7.2	3.0	virginica
1.7	0.5	5.1	3.3	setosa
4.2	1.2	5.7	3.0	versicolor
⋮				

Suppose we’re interested in how petal length depends on sepal length. Here is a plot:



The plot also shows a smooth curve for the fitted model

$$\text{Petal.Length} \approx \alpha + \beta \text{Sepal.Length} + \gamma (\text{Sepal.Length})^2.$$

By *fitted model* we mean that the parameters α , β and γ have been chosen so as to make the approximation as good as possible.

What does “make the approximation as good as possible” mean? In Section 5.4 we will see how to formulate linear models probabilistically. This will let us be precise about what the best fit is—namely, the maximum likelihood fit.

* * *

In this model, we would say there are two features, `Sepal.Length` and $(\text{Sepal.Length})^2$. Rows in the dataset have other attributes, and they can be transformed to create an infinite

³²It’s tempting for computer scientists and mathematicians to think that data science is about algorithms and calculating with distributions and so on, but shared datasets are arguably more important. C.P. Scott, the former editor of *The Guardian*, said “Comment is free, but facts are sacred”.

Modern advances in neural networks and deep learning were propelled by two shared datasets: the MNIST database of handwritten digits, and the ImageNet database of labelled photos. The story of ImageNet and of Fei-Fei Li, the researcher who collected it, is told in *The data that transformed AI research—and possibly the world*, <https://qz.com/1034972/the-data-that-changed-the-direction-of-ai-research-and-possibly-the-world/>.

In addition to shared datasets, it’s also useful to have a shared challenge, what David Donoho calls a *common task framework*. See David Donoho. *50 years of Data Science*. Presentation at the Tukey centennial workshop. 2015. URL: <http://courses.csail.mit.edu/18.337/2015/docs/50YearsDataScience.pdf>

variety of features, but we'll only use the word *feature* for data attributes that are being used in a model. We call `Petal.Length` the *outcome* or *label* of this model, not a feature.

Why two features, and not one, or three? From the perspective of the person preparing the dataset, there is only one feature, `Sepal.Length`. From the perspective of the person computing α , β , and γ , there are two data features that have to be accounted for, and it's irrelevant that they came from the same column in the dataset. From the perspective of a stickler for definitions, the definition of 'linear model' says that parameters are weighted by features, so there is really a third feature, the constant feature 1 that is being used as a weight for the α parameter. Don't get uptight about defining the word 'feature', just write out your models explicitly, and there will be no confusion.

* * *

The model is linear because it combines the unknown parameters α , β and γ in a linear formula. There's no reason to think this is in any way a 'true' model, and we could equally well have proposed a non-linear model e.g.

$$\text{Petal.Length} \approx \alpha - \beta e^{-\gamma \text{Sepal.Length}}.$$

Linear models are especially convenient to work with. They lend themselves to efficient algorithms; linear models strained the computing capabilities of a desktop PC in the late 1980s, and non-linear models in the form of neural networks strain the computing capabilities of clusters of GPUs in server farms today.

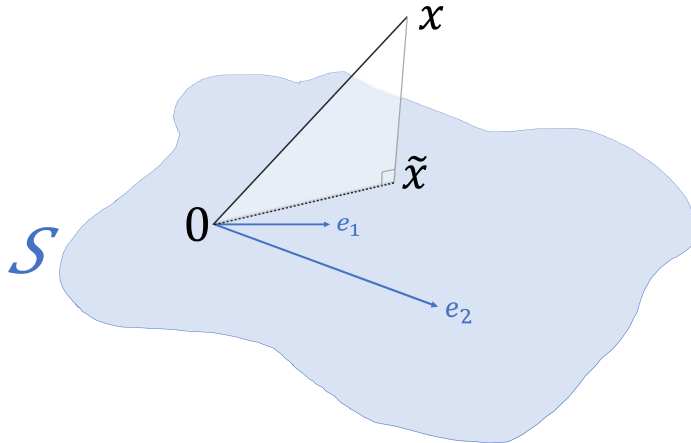
Linear models are also an easy way to explore aspects of the dataset, and they are the building block for many other models, some of which you will study in Part II *Machine Learning and Bayesian Inference*: support vector machines, perceptrons, and deep neural networks.

The intuition behind linear modelling comes from the mathematics of linear spaces. We'll revise this maths first, before returning to linear modelling.

5.1. Linear mathematics

This picture illustrates all the concepts from vector spaces and linear mathematics that we'll need for this data science course:

- Linearly independent basis vectors e_1 and e_2
- The linear subspace spanned by those vectors, $S = \{\lambda_1 e_1 + \lambda_2 e_2 : -\infty < \lambda_1, \lambda_2 < \infty\}$
- Another vector x can be projected onto the subspace, by finding the point $\tilde{x} = \hat{\lambda}_1 e_1 + \hat{\lambda}_2 e_2$ in S that is closest to x
- The residual $x - \tilde{x}$ is orthogonal to the basis vectors



We'll now define some these concepts abstractly and mathematically (leaving projections to Section 5.4). It's good to get intuition from three dimensional Euclidean space—but it's also useful to have abstract definitions so that the concepts can be applied to more general settings, as you will see in Part II *Digital Signal Processing* and *Computer Vision* (Fourier transforms and wavelets) and *Quantum Computing*.

5.1.1. ABSTRACT DEFINITIONS

- Let V be a set whose elements are called *vectors*, denoted by Roman letters u, v, w , etc.
- Let F be a field whose elements are called *scalars*, denoted by Greek letters λ, μ , etc. For our purposes, take F to be either the real numbers or the complex numbers.
- Let there be a binary operation $V \times V \rightarrow V$, called *addition*, written $v + w$.
- Let there be a binary operation $F \times V \rightarrow V$, called *scalar multiplication*, written λv .
- Let there be a binary operation $V \times V \rightarrow F$, called *inner product*, written $v \cdot w$.

Vector space. V is called a *vector space* over F if the following properties hold:

1. Associativity: $(u + v) + w = u + (v + w)$ for all vectors u, v, w .
2. Commutativity: $u + v = v + u$ for all vectors u, v
3. Zero vector: there is a vector 0 such that $v + 0 = v$ for all vectors v
4. Inverse: for every vector v there is a vector denoted $-v$ such that $v + (-v) = 0$
5. $\lambda(v + w) = \lambda v + \lambda w$ for every scalar λ and vectors v, w
6. $(\lambda + \mu)v = \lambda v + \mu v$ and $(\lambda\mu)v = \lambda(\mu v)$ for all scalars λ, μ and vector v
7. $1v = v$ for every vector v , where 1 is the unit scalar (i.e. $1\lambda = \lambda$ for every scalar λ).

Linear combinations and bases. Let v_1, \dots, v_n be vectors in a vector space and $\lambda_1, \dots, \lambda_n$ be scalars. Then the vector $\lambda_1 v_1 + \dots + \lambda_n v_n$ is called a *linear combination* of v_1, \dots, v_n . The set of all linear combinations

$$S = \{\lambda_1 v_1 + \dots + \lambda_n v_n : \lambda_i \in F \text{ for all } i\}$$

is called the *span* of $\{v_1, \dots, v_n\}$, and the vectors v_i are said to *span* S . Clearly $S \subseteq V$, and it is not hard to check that S is also a vector space. It is called a *subspace* of V .

In introductory geometry it's common to use bold symbols for vectors, e.g. $\mathbf{v} + \mathbf{0} = \mathbf{v}$ and $1\mathbf{v} = \mathbf{v}$. This notation makes it clear that $\mathbf{0}$ is a vector and 1 is a scalar. The bold notation is less common in more advanced applications, so you have to rely on type inference to spot that $\mathbf{0}$ is a vector and 1 is a scalar.

Vectors v_1, \dots, v_n in a vector space are said to be *linearly independent* if

$$\lambda_1 v_1 + \dots + \lambda_n v_n = 0 \implies \lambda_1 = \dots = \lambda_n = 0.$$

If this is not the case, then they are said to be *linearly dependent*.

If there is a finite set of vectors e_1, \dots, e_n that span a vector space V , and they are linearly independent, then they are called a *basis* for V . It can be shown that any two bases for a vector space must have the same number of elements; this number is called the *dimension* of the vector space.

Given a basis $\{e_1, \dots, e_n\}$ of a vector space, it can be proved that any vector x can be uniquely written as

$$x = \lambda_1 e_1 + \dots + \lambda_n e_n \quad \text{for some scalars } \lambda_1, \dots, \lambda_n.$$

The n -tuple $(\lambda_1, \dots, \lambda_n)$ is called the *coordinates* of x with respect to the given basis. If we pick a different basis we'll get different coordinates, but of course the vector x itself is still the same regardless of the basis.

Inner products and orthogonality. Consider a vector space V over the field of real numbers. It is said to be an *inner product space* if the inner product satisfies these properties:

8. $v \cdot v \geq 0$ for all vectors v , and $v \cdot v = 0$ if and only if $v = 0$
9. $(\lambda u + \mu v) \cdot w = \lambda(u \cdot w) + \mu(v \cdot w)$ for all vectors u, v, w and scalars λ, μ
10. $v \cdot w = w \cdot v$ for all vectors v and w

An inner product space over the field of complex numbers is defined similarly, except that condition 10 is replaced by $v \cdot w = \overline{w \cdot v}$ where $\bar{\lambda}$ is the complex conjugate of the complex number λ . Also, the first part of condition 8 should be interpreted as $\text{Im}(v \cdot v) = 0$ and $\text{Re}(v \cdot v) \geq 0$.

Two vectors v and w in an inner product space are said to be *orthogonal* if $v \cdot w = 0$. A set of vectors (which may be finite or infinite) is said to be an *orthogonal system* if none of them is equal to 0 and in addition every pair of vectors in the set is orthogonal.

The *Euclidean norm* for an inner product space is

$$\|v\| = \sqrt{v \cdot v}.$$

A vector v with $\|v\| = 1$ is called a *unit vector*. An orthogonal system is said to be an *orthonormal system* if every vector in it is a unit vector.

5.1.2. USEFUL PROPERTIES

Here are some useful properties that can be proved from the abstract definitions.

11. $0v = 0$, for every vector v in a vector space.
12. $(-\lambda)v = -(\lambda v)$, for every vector v in a vector space and every scalar λ .
13. $(\lambda v) \cdot w = \lambda(v \cdot w)$, for all scalars λ and vectors v, w in an inner product space.
14. $0 \cdot v = 0$, for every vector v in an inner product space.
15. For all n and all scalars $\lambda_1, \dots, \lambda_n$ and vectors v_1, \dots, v_n, w in an inner product space,

$$\left(\sum_{i=1}^n \lambda_i v_i \right) \cdot w = \sum_{i=1}^n \lambda_i (v_i \cdot w).$$

16. If $\{e_1, \dots, e_n\}$ is an orthonormal system in an inner product space, then for every vector x in the span of $\{e_1, \dots, e_n\}$, the coordinates of x are given by

$$x = \sum_{i=1}^n (x \cdot e_i) e_i.$$

17. $\|u + v\| \leq \|u\| + \|v\|$ for all vectors u, v ; this is known as the *triangle inequality*.

These properties are mostly obvious when we're working with finite dimensional Euclidean space. For abstract vector spaces, they must be proved directly from the defining properties 1–10. The proofs are dull definition-pushing, but it's reassuring to know that it can be done. Here are some examples.

Exercise (Prove useful property 11). In this equation, the left hand side must be referring to the scalar $0 \in F$ and the right hand side to the vector $0 \in V$, where V is the vector space over field F , because otherwise the equation doesn't make sense—the abstract definitions don't define multiplication of vectors, and scalar multiplication yields a vector.

In both the real numbers and the complex numbers (and indeed in any field F), $0 = 0+0$. So, by property 6,

$$0v = (0 + 0)v = 0v + 0v.$$

By property 4, there is some vector $-(0v)$ such that $0v + (-(0v)) = 0$. Adding this to each side of the equation,

$$0v + (-(0v)) = (0v + 0v) + (-(0v))$$

and so, using property 1,

$$0 = 0v + (0v + (-(0v))) = 0v + 0.$$

Finally, by property 3,

$$0 = 0v.$$

Exercise (Prove useful property 12). Property 6 says that

$$\lambda v + (-\lambda)v = (\lambda + (-\lambda))v.$$

In both the real numbers and the complex numbers (and indeed in any field F), $\lambda + (-\lambda) = 0 \in F$, thus

$$\lambda v + (-\lambda)v = 0v$$

which we showed in the previous exercise to be equal to $0 \in V$. So $(-\lambda)v$ satisfies property 4 and it is therefore $-(\lambda v)$.

Exercise (Prove useful property 13).

$$\begin{aligned} (\lambda v) \cdot w &= ((\lambda + 0)v) \cdot w \quad \text{since } \lambda = \lambda + 0 \in F \\ &= (\lambda v + 0v) \cdot w \quad \text{by property 6} \\ &= \lambda(v \cdot w) + 0(v \cdot w) \quad \text{by property 9} \\ &= \lambda(v \cdot w) \quad \text{since } 0\mu = 0 \in F. \end{aligned}$$

5.1.3. ADVANCED APPLICATION: FOURIER ANALYSIS

In this course on data science, the only vector space we're interested in is a simple finite-dimensional Euclidean space over the real numbers. Section 5.2 will go into detail. But first, to illustrate that there's some merit in defining vector spaces abstractly, here's an advanced application, a step on the way to Fourier analysis.

Inner product space. Let V consist of all continuous complex-valued functions on the interval $[-\pi, \pi]$. Define addition of functions in the obvious way, define multiplication by a complex number in the obvious way, and define the inner product to be

$$f \cdot g = \frac{1}{\pi} \int_{-\pi}^{\pi} f(\tau) \overline{g(\tau)} d\tau.$$

It is easy to check that properties 1–7 are satisfied, i.e. that this is a vector space over the field of complex numbers. Using some standard results about integration one can also show that properties 8–10 are also satisfied, therefore this is an inner product space. (A typical result: if f is a continuous function, then it is integrable over a finite interval.)

Orthonormal system. Every vector in V is a continuous function. Consider the vectors

$$\{e_1, e_2, \dots\} = \left\{ \frac{1}{\sqrt{2}}, \cos(\tau), \sin(\tau), \cos(2\tau), \sin(2\tau), \cos(3\tau), \dots \right\}.$$

(The first element $1/\sqrt{2}$ is a way of writing the constant function $f(\tau) = 1/\sqrt{2}$.) With some A-level trigonometry and calculus, it can be shown that $e_i \cdot e_j = 0$ if $i \neq j$, and $e_i \cdot e_i = 1$ for every i , i.e. that this set is an orthonormal system.

Fourier series. This orthonormal system spans the subspace of V consisting of ‘well-behaved’ functions, and such functions can be written in coordinate form as

$$f = \sum_{i=1}^{\infty} (f \cdot e_i) e_i \quad (23)$$

or equivalently

$$f(\tau) = \frac{a_0}{2} + \sum_{i=1}^{\infty} (a_i \cos(i\tau) + b_i \sin(i\tau))$$

where

$$\begin{aligned} a_0 &= \frac{1}{\pi} \int_{-\pi}^{\pi} f(\tau) d\tau, \\ a_i &= \frac{1}{\pi} \int_{-\pi}^{\pi} f(\tau) \cos(i\tau) d\tau \quad \text{for } i \geq 1 \\ b_i &= \frac{1}{\pi} \int_{-\pi}^{\pi} f(\tau) \sin(i\tau) d\tau \quad \text{for } i \geq 1. \end{aligned}$$

This is known as the *Fourier series* for f . There are however some technical caveats associated with infinite series—Useful Property 16 only applies to finite bases, but equation (23) is an infinite series corresponding to an infinite orthonormal system, and this is why we need the restriction ‘well-behaved functions’. In Part II *Computer Vision and Digital Signal Processing* you will learn more about Fourier analysis and other related ways to decompose functions.

5.2. Features in data

Key idea. A numerical feature can be seen as a vector, with one real number per object in the dataset. A linear model can be seen as a linear combination of feature vectors, in which the model's parameters act as scalar multipliers.

Feature vectors are a fundamental concept in machine learning. You will see them again in Part II *Machine Learning and Bayesian Inference, Natural Language Processing, Information Retrieval*, and anything at all to do with neural networks.

In this section we'll showcase several different ways to use feature vectors to model data.

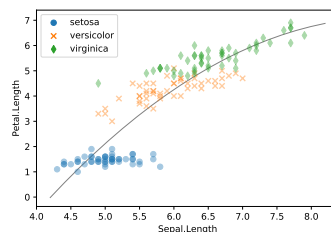
5.2.1. POLYNOMIAL FUNCTIONS

For the Iris dataset on page 63, we proposed the model

$$\text{Petal.Length} \approx \alpha + \beta \text{Sepal.Length} + \gamma (\text{Sepal.Length})^2.$$

Interpret this as a vector equation, where `Petal.Length` and `Sepal.Length` refer to entire columns from the dataset, and the square $(\text{Sepal.Length})^2$ is applied pointwise. All these vectors are n -dimensional, where n is the number of rows in the dataset. There is an implied constant vector `One` in this equation, consisting of n entries all of them 1, and the equation should really be

$$\text{Petal.Length} \approx \alpha \text{One} + \beta \text{Sepal.Length} + \gamma (\text{Sepal.Length})^2.$$



The code to fit this model is

```
1 # columns from the dataset
2 one,x,y = np.ones(len(data)), data['Sepal.Length'], data['Petal.Length']
3 # Fit the model: takes a feature matrix, and a col. vector of outcomes
4 model = sklearn.linear_model.LinearRegression(fit_intercept=False)
5 model.fit(np.column_stack([one, x, x**2]), y[:, np.newaxis])
6 alpha,beta,gamma = model.coef_.squeeze()
7 # plot a curve to depict the fitted values from the model
8 newx = np.linspace(4.2, 8.2, 200)
9 plt.plot(newx, alpha + beta*newx + gamma*newx**2)
```

In fact the model fitting function always includes a `One` vector, unless we explicitly tell it otherwise with `fit_intercept=False`. Another way to write this code, using the default `One` vector and also using `model.predict()` to relieve us from repeating the model formula, is

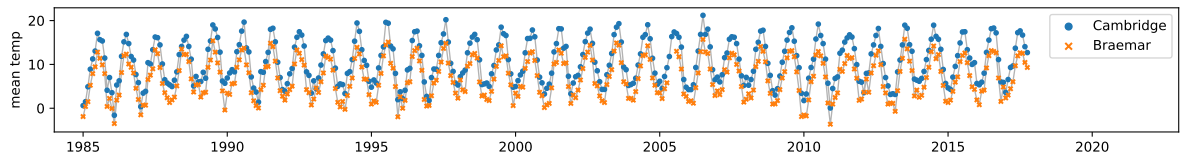
```
4 model = sklearn.linear_model.LinearRegression()
5 model.fit(np.column_stack([x, x**2]), y[:, np.newaxis])
6 newx = np.linspace(4.2, 8.2, 200)
7 plt.plot(newx, model.predict(np.column_stack([newx, newx**2])))
```

5.2.2. DISENTANGLING PERIODIC AND SECULAR EFFECTS

The UK Met Office makes available historic data³³ from 37 stations around the UK. Each station has monthly records for mean daily maximum temperature **tmax**, mean daily minimum temperature **tmin**, days of air frost **af**, total rainfall **rain**, and total sunshine duration **sun**. Coverage varies; the longest records are from Oxford and from Armagh, going back to 1853.

month	tmax	tmin	af	rain	sun	station	lat	lng	alt_m
1963 Sep	14.7	5.9	0	126.4	127.7	Eskdalemuir	55.311	-3.206	242
1955 Aug	–	–	–	35.1	194.7	Shawbury	52.794	-2.663	72
1937 May	15.3	8.4	0	59.8	184.8	Lowestoft	52.483	1.727	18
2007 Aug	20.6	11.8	0	40.3	204.6	Waddington	53.175	-0.522	68
1925 July	21.8	12.6	0	23.2	–	Sheffield	53.381	-1.490	131
⋮									

Here are two stations, Cambridge (measured at the National Institute of Agricultural Botany, between Churchill and Girton colleges), and Braemar in the Scottish highlands. The plot shows the mean temperature $\text{temp} = (\text{tmin} + \text{tmax})/2$ as a function of date.



Are temperatures increasing? It's tricky to read this directly off the plot, because of the annual cycle and because of noise. A crude solution is to simply average over the 12 months of each year, and plot this average over time. This isn't ideal, because averaging is lossy i.e. we'd be throwing away data; and because a missing value for one month will cause the entire year to be missing.

A cleverer solution is to use features to model the effects we're trying to capture. There are two effects, an annual cycle, and a (hypothetical) increasing trend, which we can describe by the model

$$\text{temp} \approx \alpha + \beta \sin(2\pi t) + \gamma t$$

where t is the date in years, and α , β , γ , and θ are unknown parameters. (The plot suggests that α is different for different stations, and the other parameters might also be different, so let's concentrate on a single station for now.)

Linear models are much easier to fit than non-linear models. The model we've proposed for tmean is linear in α and β and γ and not in θ —but there is a cunning trick from A-level trigonometry that lets us rewrite it as a linear model. The trick is

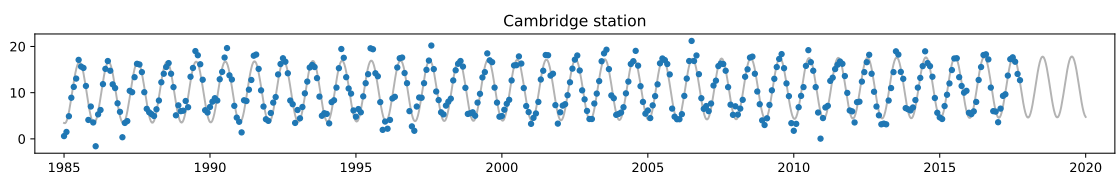
$$\sin(A + B) = \sin A \cos B + \cos A \sin B$$

and so our model can be rewritten

$$\text{temp} \approx \alpha + \beta_1 \sin(2\pi t) + \beta_2 \cos(2\pi t) + \gamma t.$$

This model has three feature vectors: the vector \mathbf{s} whose i th element is $s_i = \sin(2\pi t_i)$; the vector \mathbf{c} whose i th element is $c_i = \cos(2\pi t_i)$; and the vector \mathbf{t} . There is also one outcome vector \mathbf{tmean} . Here is the fitted model for Cambridge:

Why is α so extreme? It is the temperature in the year 1 BC (there was no year 0 AD), based on linearly extrapolating the rate γ . It's daft to trust that the model will predict well for such a wild extrapolation!



The parameters of the fitted model are $\alpha = -63.9^\circ\text{C}$, $\beta_1 = -1.07^\circ\text{C}$, $\beta_2 = -6.52^\circ\text{C}$, and $\gamma = 0.0372^\circ\text{C}/\text{year}$. The code is

³³<https://www.metoffice.gov.uk/public/weather/climate-historic>


```
1 # The data columns and features we'll use
2 t = data['yyy'] + (data['mm']-1)/12
3 temp = (data['tmin']+data['tmax'])/2
4 s,c = np.sin(2* $\pi$ *t), np.cos(2* $\pi$ *t)
5 # We'll restrict attention to a subset of rows
6 i = np.logical_and(data['station'] == 'Cambridge', t >= 1985)
7 # Fit a linear model
8 model = sklearn.linear_model.LinearRegression()
9 model.fit(np.column_stack([s,c,t])[i,:], temp[i, np.newaxis])
10 print(model.intercept_, model.coef_)
11 # output: [-63.89684304] [[-1.06641763 -6.5198444  0.03720952]]
```

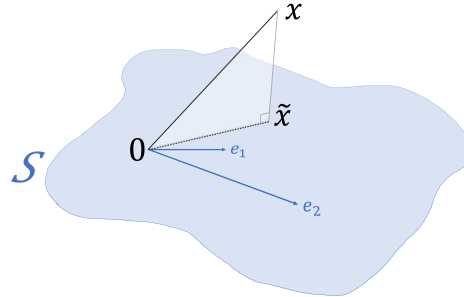
5.2.3. ONE-HOT CODING, TIME SERIES, LINEARITY OF TRENDS

Example sheet 3b shows you how features can be used for a variety of other modeling tasks.

5.3. Orthogonal projection

Let's return to the key picture that illustrates linear mathematics.

- Linearly independent basis vectors e_1 and e_2
- The linear subspace spanned by those vectors, $S = \{\lambda_1 e_1 + \lambda_2 e_2 : -\infty < \lambda_1, \lambda_2 < \infty\}$
- Another vector x can be projected onto the subspace, by finding the point $\tilde{x} = \hat{\lambda}_1 e_1 + \hat{\lambda}_2 e_2$ in S that is closest to x
- The residual $x - \tilde{x}$ is orthogonal to the basis vectors



In Section 5.1 we reviewed vector spaces and bases. We'll now define projection in inner product spaces.

The Projection Theorem. Let V be an inner product space, let $\{e_1, \dots, e_n\}$ be a finite collection of vectors, and let S be the subspace spanned by these vectors. Given a vector $x \in V$, there is a unique vector \tilde{x} that is closest to x , i.e. that achieves

$$\|x - \tilde{x}\| = \min_{x' \in S} \|x - x'\|.$$

Furthermore, $x - \tilde{x}$ is orthogonal to S , i.e.

$$(x - \tilde{x}) \cdot y = 0 \quad \text{for all } y \in S.$$

The element \tilde{x} is called the *orthogonal projection* of x onto S , and $x - \tilde{x}$ is called the *residual*.

Let's illustrate this theorem in three-dimensional Euclidean space. Let $e_1 = [1, 1, 0]$, let $e_2 = [1, 0, -1]$, and let $x = [1, 2, 3]$.

FINDING THE CLOSEST POINT

What is the closest point to x in the span of $\{e_1, e_2\}$? Just write out the optimization problem we want to solve:

$$\min_{\lambda_1, \lambda_2} \|x - (\lambda_1 e_1 + \lambda_2 e_2)\|.$$

We can compute the solution directly:

```

1 e1, e2, x = np.array([1, 1, 0]), np.array([1, 0, -1]), np.array([1, 2, 3])
2 lambda1, lambda2 = scipy.optimize.fmin(lambda lambda: np.linalg.norm(x - lambda[0]*e1 - lambda[1]*e2), [0, 0])
3 lambda1*e1 + lambda2*e2 # outputs: array([ 0.33332018,  2.66666169,  2.33334151])

```

Or we can try algebra. Expanding the definition of $\|\cdot\|$, we want to minimize

$$x \cdot x - 2(\lambda_1 x \cdot e_1 + \lambda_2 x \cdot e_2) + (\lambda_1^2 e_1 \cdot e_1 + 2\lambda_1 \lambda_2 e_1 \cdot e_2 + \lambda_2^2 e_2 \cdot e_2).$$

Differentiating with respect to λ_1 and λ_2 and setting the derivatives equal to 0,

$$\begin{aligned} \frac{\partial}{\partial \lambda_1} = 0 : & \quad -2x \cdot e_1 + 2\lambda_1 e_1 \cdot e_1 + 2\lambda_2 e_1 \cdot e_2 = 0 \\ \frac{\partial}{\partial \lambda_2} = 0 : & \quad -2x \cdot e_2 + 2\lambda_1 e_1 \cdot e_2 + 2\lambda_2 e_2 \cdot e_2 = 0 \end{aligned} \tag{24}$$

or equivalently

$$\begin{aligned} \lambda_1 e_1 \cdot e_1 + \lambda_2 e_1 \cdot e_2 &= x \cdot e_1 \\ \lambda_1 e_1 \cdot e_2 + \lambda_2 e_2 \cdot e_2 &= x \cdot e_2. \end{aligned}$$

We can compute the solution to these equations:

Mathematicians prefer to write \inf rather than \min in equations like this, where the minimum is being taken over an infinite set and it hasn't yet been established that the minimum is attained.

```

1  λ1, λ2 = np.linalg.solve([[e1 @ e1, e1 @ e2], [e1 @ e2, e2 @ e2]],
2  [x @ e1, x @ e2])
3  λ1*e1 + λ2*e2 # outputs: array([ 0.33333333, 2.66666667, 2.33333333])

```

Or, for geometrical insight, we can rearrange equations (24) to get

$$\begin{aligned}(x - (\lambda_1 e_1 + \lambda_2 e_2)) \cdot e_1 &= 0 \\ (x - (\lambda_1 e_1 + \lambda_2 e_2)) \cdot e_2 &= 0\end{aligned}$$

In other words, the residual is orthogonal to e_1 and to e_2 , and hence it's orthogonal to every linear combination of e_1 and e_2 .

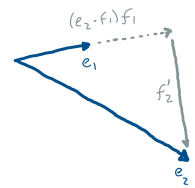
EXPLICIT PROJECTION VIA AN ORTHONORMAL BASIS

Another way to find \tilde{x} is by creating an orthonormal basis out of $\{e_1, e_2\}$ and then applying Useful Property 16 on page 66 to get the coordinates of \tilde{x} . Let's create an orthonormal basis first. Start by setting f_1 to be a unit vector in the same direction as e_1 :

$$f_1 = \frac{e_1}{\|e_1\|}.$$

Next, construct f_2 by subtracting the part that's parallel to f_1 :

$$f'_2 = e_2 - (e_2 \cdot f_1)f_1, \quad f_2 = \frac{f'_2}{\|f'_2\|}.$$



This construction ensures that $f'_2 \cdot f_1 = 0$ therefore $f_2 \cdot f_1 = 0$, and it also ensures that both f_1 and f_2 are unit vectors. We've written f_1 and f_2 as linear combinations of e_1 and e_2 , and it's easy to check that e_1 and e_2 can be written as linear combinations of f_1 and f_2 , thus $\text{span}\{e_1, e_2\} = \text{span}\{f_1, f_2\} = S$.

Useful Property 16 now tells us exactly what the coordinates are for \tilde{x} :

$$\tilde{x} = (\tilde{x} \cdot f_1)f_1 + (\tilde{x} \cdot f_2)f_2.$$

Furthermore, the Projection Theorem tells us that the residual is orthogonal to $S = \text{span}\{f_1, f_2\}$, which means $(x - \tilde{x}) \cdot f_1 = (x - \tilde{x}) \cdot f_2 = 0$, thus

$$\tilde{x} = (x \cdot f_1)f_1 + (x \cdot f_2)f_2.$$

In numpy,

```

1  f1 = e1 / np.linalg.norm(e1)
2  f'2 = e2 - (e2 @ f1) * f1
3  f2 = f'2 / np.linalg.norm(f'2)
4  (x@f1)*f1 + (x@f2)*f2 # outputs: array([ 0.33333333, 2.66666667, 2.33333333])

```

COLINEARITY

In this example, we projected onto linearly independent basis vectors e_1 and e_2 . What happens if we want to project onto a collection of linearly dependent vectors, e.g. if $e_2 = \alpha e_1$?

The Projection Theorem doesn't assume linear independence, so the overall result still holds: there is still a unique projection \tilde{x} . The explicit projection method would still work, but it would give $f'_2 = 0$, so we'd just discard that vector from the orthonormal basis. Equations (24) would still be correct, but they would have multiple solutions for λ_1 and λ_2 .

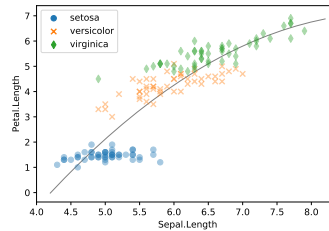
In summary, we can always project x onto a subspace $S = \text{span}\{e_1, \dots, e_n\}$. If the e_i are linearly independent, i.e. if they form a basis for S , then we can find the coordinates of \tilde{x} with respect to the e_i , and the coordinates are unique. If the e_i are linearly dependent, then there are multiple ways to write \tilde{x} as a linear combination of the e_i .

5.4. Linear regression and least squares

Key idea. Suppose we want to fit a linear model to a dataset. If we model the outcome as a normal random variable, then maximum likelihood estimation of the unknown parameters is exactly the same as an orthogonal projection of the outcome vector onto feature vectors.

In the Iris dataset on page 63, we investigated how petal length depends on sepal length, and we proposed the model

$$\text{Petal.Length} \approx \alpha + \beta \text{Sepal.Length} + \gamma (\text{Sepal.Length})^2.$$



Let's be explicit and propose a full probabilistic model:

$$\text{Petal.Length}_i \sim \text{Normal}\left(\alpha + \beta \text{Sepal.Length}_i + \gamma (\text{Sepal.Length}_i)^2, \sigma^2\right) \quad (25)$$

where $i \in \{1, \dots, n\}$ indexes the rows of the dataset, and each Petal.Length_i is an independent random variable, and Sepal.Length_i is being treated as a non-random value.

A full probabilistic model lets us be precise about how to estimate parameters—we should simply look for maximum likelihood estimators. For brevity, let $y_i = \text{Petal.Length}_i$, let $e_i = \text{Sepal.Length}_i$, and let $f_i = (\text{Sepal.Length}_i)^2$. Then the density function for a single observation is

$$f(y_i) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\left(y_i - (\alpha + \beta e_i + \gamma f_i)\right)^2 / 2\sigma^2}$$

and the log likelihood is

$$\log \text{lik}(\alpha, \beta, \gamma, \sigma | y) = -\frac{n}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^n \left(y_i - (\alpha + \beta e_i + \gamma f_i)\right)^2.$$

We can maximize the log likelihood in two steps. The first step is to maximize the last term, i.e. find $\hat{\alpha}$, $\hat{\beta}$, and $\hat{\gamma}$ that solve

$$\min_{\alpha, \beta, \gamma} \|y - (\alpha \mathbf{1} + \beta e + \gamma f)\|^2.$$

In this equation we have switched to vector notation, and $\mathbf{1}$ means the vector $[1, 1, \dots, 1]$. This is nothing other than finding the orthogonal projection of the outcome vector y onto the space spanned by the feature vectors $\{\mathbf{1}, e, f\}$. Another name for this procedure is the *method of least squares*, invented by Gauss.

The second step is to find σ to maximize what's left, i.e. to solve

$$\min_{\sigma > 0} \left\{ \frac{n}{2} \log(2\pi\sigma^2) + \frac{1}{2\sigma^2} \|y - (\hat{\alpha} \mathbf{1} + \hat{\beta} e + \hat{\gamma} f)\|^2 \right\}.$$

This is a trivial one-parameter optimization problem, once we know $\hat{\alpha}$, $\hat{\beta}$, and $\hat{\gamma}$.

CONFOUNDED FEATURES

When is there a unique solution for the maximum likelihood estimators of the parameters in a linear model?

The Projection Theorem says that there is always a unique projection of y onto the feature space. The remark about colinearity at the end of Section 5.3 says that if the features are linearly independent, then the feature coefficients i.e. the maximum likelihood estimators are unique. If the features are linearly dependent, then the maximum likelihood

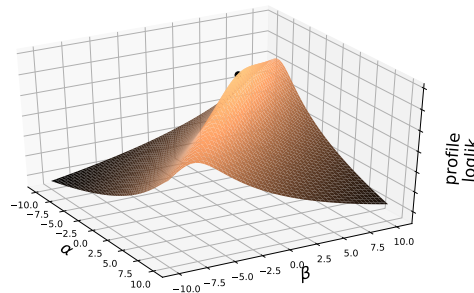
estimators are not unique, and we say the features are *confounded*. Sometimes, geometrical intuition about colinearity can help us debug what's going wrong with a probabilistic model fit. Example sheet 3b asks you to look at examples of confounding.

But data science is all about noise and uncertainty, whereas linear independence is a strict clean mathematical definition, so we shouldn't entirely trust it. An example can be found on example sheet 2 question 5, which highlights the challenges of inference when features are 'linearly independent, but only just'. Here is another example.

Example (Correlated features). Let the ground truth be as follows: let e and f be two features of length 20 differing only in one coordinate, $e = [1, 1, \dots, 1]$ and $f = [0, 1, 1, \dots, 1]$, and suppose we generate a vector y consisting of 20 values from $\text{Normal}(0.5e_i + 1.5f_i, 1)$, $i = 1, \dots, 20$. Can we recover the coefficients 0.5 and 1.5?

Pretend we don't know the ground truth, and fit the model $Y_i \sim \text{Normal}(\alpha e_i + \beta f_i, \sigma^2)$. Since e and f are linearly independent, we will get a unique solution when we solve for the maximum likelihood estimators $\hat{\alpha}$ and $\hat{\beta}$. To see some context, let's plot the log likelihood after optimizing out the nuisance parameter σ . (When we maximize out nuisance parameters, what's left is called the *profile log likelihood*.)

$$\text{profile log lik}(\alpha, \beta | y) = \max_{\sigma > 0} \text{log lik}(\alpha, \beta, \sigma | y).$$



The maximum likelihood estimators in this case are $\hat{\alpha} = 0.70$, $\hat{\beta} = 1.50$. The plot shows a peak at this value, and it also shows a long 'ridge' where $\alpha + \beta$ is roughly constant. The plot is telling us that we can be very confident about the value of $\alpha + \beta$, but not very confident about either of them individually.

INFERENCE

An explicit probabilistic model like equation (25) lets us make inferences, using the techniques from Section 3 and Example Sheet 2.

- We can use Bayesian reasoning: invent prior distributions for the unknown parameters, and calculate their posterior distributions.
- We can use frequentist reasoning: compute confidence intervals for the unknown parameters, using bootstrap resampling. Resampling means generating synthetic datasets based on the data we actually saw; the natural resampling method here is to compute the maximum likelihood estimates for the parameters, and then to plug these estimates into the model (25) and generate new random variables. This is known as *parametric resampling*.
- We can conduct hypothesis tests, along the lines of example sheet 2 question 3.

Example sheet 3b asks you to look at these types of inference.

5.5. Feature engineering

In Section 5.2, we cleverly designed features to allow us to extract an underlying linear trend from climate data, taking account of the annual cycle. In general, we design features for several purposes:

- features to extract a particular summary from the data, e.g. the linear trend in the climate data;
- features that correspond to a causal model for which we want to estimate parameters, e.g. the transition probabilities for a Markov chain;
- ‘black box’ features that capture enough detail for us to be able to make good predictions or extrapolations—we don’t have to understand such features, we just want them to work well;
- features that turn arbitrary objects like tweets or sentence fragments into numbers that can be put into quantitative models, e.g. distributional semantics which you will study in Part II *Natural Language Processing*, and term frequency models for documents which you will study in Part II *Information Retrieval*.

* * *

The more features we add, the better the fit i.e. the smaller the residual we can achieve. But models with too many features tend to be bad at generalizing to new data (see example sheet 2 question 5). It’s an art to design sets of features that are expressive enough to capture the meaningful variation in the data, while being parsimonious enough to generalize well. Here are two strategies that are sometimes helpful. You will learn more about them in further courses on machine learning and data science.

Feature selection. Start with a long list of possible features. Pick m , a number of features to use, and find the best fitting model subject to the constraint that it’s only allowed to use m of the possible features. This is called *feature selection*.

Dimension reduction. Start with a long list of possible features $\{e_1, \dots, e_n\}$. Pick m , a number of features to use, and construct a set of m vectors $\{f_1, \dots, f_m\}$ that capture the features as well as possible. For example, we might set $m = 2$ and pick $\{f_1, f_2\}$ to minimize $\sum_{i=1}^n \|e_i - \tilde{e}_i\|$, where \tilde{e}_i is the projection of vector e_i onto the span of $\{f_1, f_2\}$. This would be called a *two-dimensional embedding* of the features, and it is an example of *dimension reduction*.