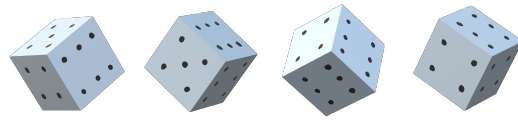


# IB Foundations of Data Science

Damon Wischik, Computer Science, Cambridge University

Michaelmas Term 2018



## Contents

<b>1</b>	<b>Probabilistic modelling</b>	<b>1</b>
1.1	Likelihood . . . . .	1
1.2	Random variables . . . . .	2
1.3	Independence and joint distributions . . . . .	5
1.4	Fitting distributions . . . . .	8
1.5	Custom distributions . . . . .	11
1.6	Fitting a model . . . . .	15
<b>2</b>	<b>How random variables behave</b>	<b>17</b>
2.1	Mean and variance . . . . .	17
2.2	A rule of thumb for confidence intervals . . . . .	19
2.3	Convergence theorems . . . . .	21
2.4	Monte Carlo integration . . . . .	23
2.5	Importance sampling * . . . . .	26
2.6	The empirical distribution . . . . .	28
<b>3</b>	<b>Inference</b>	<b>31</b>
3.1	Bayesianism . . . . .	33
3.1.1	Finding the posterior distribution . . . . .	33
3.1.2	Readouts from posterior distributions . . . . .	34
3.2	Frequentism . . . . .	37
3.2.1	Classic analysis . . . . .	37
3.2.2	Resampling . . . . .	38
3.2.3	The bootstrap . . . . .	39
3.3	Model selection . . . . .	41
3.3.1	Hypothesis testing and p-values . . . . .	43
3.3.2	Cross validation and perplexity * . . . . .	47
3.3.3	Bayesian model weighting * . . . . .	49
<b>4</b>	<b>Crafting a model</b>	<b>51</b>
4.1	Quantifying a question . . . . .	51
4.1.1	Review of maximum likelihood estimation . . . . .	53
4.1.2	Expressivity, identifiability, and contrasts . . . . .	54
4.1.3	Natural parameters . . . . .	56
4.1.4	Logistic regression . . . . .	57
4.2	Accounting for uncertainty * . . . . .	58
<b>5</b>	<b>Feature spaces</b>	<b>61</b>
5.1	Fitting a linear model . . . . .	62
5.2	Features . . . . .	64
5.2.1	One-hot coding . . . . .	64
5.2.2	Periodic and secular trends . . . . .	64
5.2.3	Non-linear response . . . . .	65
5.2.4	Discovering features . . . . .	66
5.3	Linear mathematics . . . . .	70

5.3.1	Definitions and useful properties *	71
5.3.2	Orthogonal projection and least squares	73
5.3.3	Advanced application: Fourier analysis *	75
5.4	Linear regression and least squares	76
5.4.1	Non-identifiability and confounded features *	77
5.4.2	Gauss's invention of least squares *	79
5.5	Generalized linear models *	81
<b>6</b>	<b>Random processes</b>	<b>83</b>
6.1	Markov chains	85
6.2	Calculations based on memorylessness	87
6.3	Application: double-spend in bitcoin *	90
6.4	Inference with Markov chains *	94
6.5	Limit theorems and equilibrium	96
6.5.1	Stationary behaviour and irreducibility	96
6.5.2	Detailed balance	98
6.5.3	Ergodic theorem	99
6.5.4	Limiting behaviour and aperiodicity	99
<b>A</b>	<b>Standard random variables</b>	<b>101</b>
A.1	Variables associated with waiting and counting	102
A.2	Variables associated with sizes	103
A.3	Variables for inference	104

## ACKNOWLEDGEMENTS

Thanks to Richard Gibbens, Jakub Perlin, Thomas Sauerwald, and Alicja Chaszczewicz for spotting bugs in earlier versions of these notes.

2018/09/29 at 21:16:00

# 1. Probabilistic modelling

## 1.1. Likelihood

tl;dr. In a probabilistic model, the *likelihood* is the probability of the observed outcome, viewed as a function of the unknown parameter. (This is an incomplete definition, which will be expanded on in Section 1.4.) The likelihood function measures how much evidence there is for a particular parameter value. The *maximum likelihood estimator* is the parameter value with maximum likelihood.

In an introductory probability course you might have been told the following: if we take a biased coin, which gives heads with probability  $p$  and tails with probability  $1 - p$ , and toss it  $n$  times, then the probability we get  $x$  heads is

$$\mathbb{P}(\text{num. heads} = x) = \binom{n}{x} p^x (1-p)^{n-x}, \quad x \in \{0, \dots, n\}.$$

$\binom{n}{x}$  is the binomial coefficient, also written  ${}^n C_x$ , equal to  $n! / x!(n-x)!$

What if we don't know  $p$ ? The heart of data science is to take the observed data ( $x$ , the number of heads we actually saw) and use it to make inferences about the unknown parameters. Define the likelihood function

$$\text{lik}(p) = \binom{n}{x} p^x (1-p)^{n-x}.$$

Intuitively, this function measures how much evidence there is for a particular value of  $p$ : the higher  $\text{lik}(p)$  is, the more likely that value of  $p$  is. It's common to write

$$\mathbb{P}(\text{num. heads} = x | p) \quad \text{or} \quad \text{lik}(p | x)$$

to emphasize that the formula involves both the unknown parameter  $p$  and the observed data  $x$ . This is NOT a conditional probability, it just happens to use the same vertical bar symbol.

A sensible way to estimate  $p$  is to find the value that maximizes the likelihood, by solving

$$\frac{d}{dp} \text{lik}(p) = \binom{n}{x} \left( x p^{x-1} (1-p)^{n-x} - (n-x) p^x (1-p)^{n-x-1} \right) = 0$$

which has the solution

$$p = \frac{x}{n}.$$

This is called the maximum likelihood estimator. It's usually easier to maximize  $\log(\text{lik}(\cdot))$  rather than  $\text{lik}(\cdot)$ , and it necessarily has the same solution. In this case,

$$\begin{aligned} \log \text{lik}(p) &= \kappa + x \log p + (n-x) \log(1-p) \\ \frac{d}{dp} \log \text{lik}(p) &= \frac{x}{p} - \frac{n-x}{1-p} = 0 \quad \implies \quad p = \frac{x}{n} \end{aligned}$$

where  $\kappa$  is a constant i.e. doesn't depend on  $p$ .

\* \* \*

A large part of this course (and of machine learning in general) is knowing enough building blocks to come up with useful probability models and likelihood functions. The rest of this section is about one of the core probability models, the random sample, and builds up to an explanation of 'softmax cross-entropy', the training objective for a neural network classifier.

But what does likelihood actually measure? Another goal of this course is to explore the paths that statisticians and philosophers have taken in thinking about likelihood and what it can be used for. Here are two remarks, to start you thinking.

- The maximum likelihood procedure is intuitively sensible, but what guarantees do we have about its accuracy? If I toss 3 coins and get 3 heads, the maximum likelihood estimator is  $p = 1$ . If I toss 1 million coins and get 1 million heads, it's still 1. I should be more confident in the latter case, but how do we measure confidence? And what would it take to persuade me I'll never see a tail?
- The likelihood is NOT a probability density. Probability density functions must integrate to 1, but in this example  $\int_0^1 \text{lik}(p) dp$  is equal to  $1/(1+n)$ . If it's not a probability density, what is it?

## 1.2. Random variables

tl;dr. A *random variable* is a function that can give different answers, e.g. a function that calls a random number generator. We say it *takes values in*  $\Omega$  to mean that the return value of the function is an element of the set  $\Omega$ . Every random variable  $X$  has a *probability distribution*

$$\mathbb{P}(X \in A)$$

which specifies the probability that the return value lies in a subset  $A \subseteq \Omega$ . If  $\Omega$  is countable e.g.  $\Omega = \{\dots, -1, 0, 1, 2, \dots\}$ , the random variable is said to be *discrete*, and then

$$\mathbb{P}(X \in A) = \sum_{x \in A} \mathbb{P}(X = x). \quad (1)$$

For very many applications we work with real-valued random variables, and define the *cumulative distribution function* (often shortened to *distribution function* or even *distribution*) to be

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

If this function is differentiable, then  $X$  is said to be a *continuous* random variable with density function  $f(x) = F'(x)$ , and

$$\mathbb{P}(X \leq x) = \int_{-\infty}^x f(x) dx. \quad (2)$$

For a continuous random variable,  $\mathbb{P}(X = x) = 0$  for every  $x$ , and so

$$\mathbb{P}(a \leq X \leq b) = \mathbb{P}(a < X \leq b) = \mathbb{P}(a \leq X < b) = \mathbb{P}(a < X < b).$$

**Notation.** In this course we'll write  $\Pr_X(x)$  for

$$\Pr_X(x) = \begin{cases} \mathbb{P}(X = x) & \text{when } X \text{ is a discrete random variable} \\ & (\Pr_X \text{ is called the } \textit{probability mass function}) \\ f(x) & \text{when } X \text{ is a continuous random variable with density } f \\ & (\Pr_X \text{ is called the } \textit{probability density function}) \end{cases}$$

Many results in data science and machine learning apply to both discrete and continuous random variables, but with slightly different meanings, and this notation helps us write formulae that work for both cases. The reason we need different definitions is that for a continuous random variable  $\mathbb{P}(X = x) = 0$  but  $\mathbb{P}(X \in [x, x + \delta]) \approx \delta f(x)$  for small  $\delta$ , and so it's the density function that's useful.

A *conditional random variable*  $(X | C)$  is just a random variable whose distribution is conditional: if  $Y = (X | C)$ , then  $\mathbb{P}(Y \in A) = \mathbb{P}(X \in A | C)$ . We write

$$\Pr_X(x | C) = \begin{cases} \mathbb{P}(X = x | C) & \text{when } X \text{ is discrete} \\ G'(x) \text{ where } G(x) = \mathbb{P}(X \leq x | C) & \text{when } X \text{ is continuous.} \end{cases}$$

It's worth mentioning some terminology for describing distribution functions for numerical random variables. 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 appendix on page 101 gives Python functions for finding percentiles, as well as densities and cumulative distributions and others.

**Exercise 1.1.** Here is a discrete integer-valued 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)`. What is its cumulative distribution function?

To find the distribution of  $X$ , we can use simple probability calculation. To get  $X = 1$  we need the `random.random() > 1/3` test to fail on the first pass, which has probability  $1/3$ . To get  $X = 2$  we need the test to succeed on the first pass then fail on the second, which has probability  $2/3 \times 1/3$ . To get  $X = 3$  we need the test to succeed on the first two passes then fail on the third, which has probability  $2/3 \times 2/3 \times 1/3$ . Generalizing, if  $X$  is the outcome of `rgeom(p)` then

$$\mathbb{P}(X = k) = (1 - p)^{k-1}p.$$

From here, it's plain maths:

$$\begin{aligned}
 \mathbb{P}(X \leq x) &= 1 - \mathbb{P}(X > x) \\
 &= 1 - \sum_{k=x+1}^{\infty} \mathbb{P}(X = k) = 1 - (1 - p)^x p \sum_{k=0}^{\infty} (1 - p)^k \\
 &= 1 - (1 - p)^x p \frac{1}{1 - (1 - p)} = 1 - (1 - p)^x.
 \end{aligned}$$

Standard maths formula:  
 $1 + r + r^2 + \dots =$   
 $1/(1 - r)$  for  $|r| < 1$ .

**Exercise 1.2.** Consider these two random variables:

```

1 def rexp(λ):
2     u = random.random()
3     return - math.log(u) / λ
4 def rgeom2(p):
5     λ = - math.log(1-p)
6     x = rexp(λ)
7     return math.ceil(x)

```

Let  $X$  be the output of `rexp(λ)`. Find the density function of  $X$ . Hence show that `rgeom` and `rgeom2` have the same probability distribution.

Let's work out the cumulative distribution function for  $X$  first. (It's often easier to work with distribution functions rather than densities in problems like this.)

$$\mathbb{P}(X \leq x) = \mathbb{P}\left(-\frac{1}{\lambda} \log U \leq x\right) = \mathbb{P}(U \geq e^{-\lambda x}) = 1 - e^{-\lambda x}$$

where  $U$  is the output of `random.random()`. The density of  $X$  is thus

$$\Pr_X(x) = \frac{d}{dx} \mathbb{P}(X \leq x) = \lambda e^{-\lambda x}.$$

Now let  $Y$  be the output of `rgeom2(p)`.

$$\begin{aligned}
 \mathbb{P}(Y = k) &= \mathbb{P}(k - 1 < X \leq k) = \int_{x=k-1}^k \lambda e^{-\lambda x} dx \\
 &= (1 - e^{-\lambda k}) - (1 - e^{-\lambda(k-1)}) = (e^{-\lambda})^{k-1} e^{-\lambda} (e^{\lambda} - 1) \\
 &= (1 - p)^{k-1} p
 \end{aligned}$$

which is exactly the same as for the output of `rgeom(p)`. By (1),

$$\mathbb{P}(\text{rgeom}(p) \in A) = \mathbb{P}(\text{rgeom2}(p) \in A) \quad \text{for any set } A$$

i.e. they have the same probability distribution.

`math.log` is the natural logarithm i.e. to base  $e$ , and `math.ceil` rounds up to the nearest integer.

**Exercise 1.3.** Let  $X \sim \text{Exp}(\lambda)$ . Find  $\Pr_X(x \mid X \geq \nu)$ .

Exp is for the Exponential random variable. This and other standard distributions are listed in the appendix, page 102.

According to the definition of conditional random variables,

$$\Pr_X(x \mid X \geq \nu) = G'(x) \quad \text{where} \quad G(x) = \mathbb{P}(X \leq x \mid X \geq \nu).$$

By the definition of conditional probability,

$$G(x) = \frac{\mathbb{P}(X \leq x \text{ and } X \geq \nu)}{\mathbb{P}(X \geq \nu)} = \begin{cases} 0 & \text{if } x < \nu \\ (\mathbb{P}(X \leq x) - \mathbb{P}(X < \nu)) / \mathbb{P}(X \geq \nu) & \text{if } x \geq \nu \end{cases}$$

and differentiating this with respect to  $x$  gives

$$G'(x) = \begin{cases} 0 & \text{if } x < \nu \\ \Pr_X(x) / \mathbb{P}(X \geq \nu) & \text{if } x \geq \nu. \end{cases}$$

For the Exponential random variable with rate  $\lambda$ ,  $\Pr_X(x) = \lambda e^{-\lambda x}$  and  $\mathbb{P}(X \geq x) = e^{-\lambda x}$ , thus

$$\Pr_X(x \mid X \geq \nu) = \begin{cases} 0 & \text{if } x < \nu \\ \lambda e^{-\lambda(x-\nu)} & \text{if } x \geq \nu. \end{cases}$$

**Example 1.4.**

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,

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

\* \* \*

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 not describing a random variable. There was only one first lecture for this course, and so every time we ask the question "did two or more people in that lecture share a birthday?" we get the same answer. Either there was a shared birthday, or there wasn't, so the probability is either 1 or 0.

When we write for example "Let  $X$  be  $\text{rgeom}(1/3)$ ", 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.

So when a deep neural network tells us "confidence 99.3%" in its image classification<sup>1</sup>, what on earth does it mean? Is it referring to a probability, and if so then what's the random variable—is it the image, or the true label, or the (deterministic) output of the neural network classifier? If not a probability, then what does 'confidence' measure? We'll return to these questions in section 3. Today's neural networks have trouble reasoning about uncertainty—and anyway this is a panda not a gibbon.



"gibbon",  
99.3%  
confidence

<sup>1</sup>Example from I. J. Goodfellow, J. Shlens, and C. Szegedy. "Explaining and Harnessing Adversarial Examples". In: *ArXiv e-prints* (Dec. 2014). arXiv: 1412.6572 [stat.ML]

### 1.3. Independence and joint distributions

tl;dr. Any pair of random variables  $(X, Y)$  has a *joint distribution*

$$\mathbb{P}((X, Y) \in C)$$

which specifies the probability of any joint event  $C$ . For a pair of continuous random variables, the joint distribution can be specified by a joint probability density  $\Pr_{X,Y}(x, y)$ ,

$$\mathbb{P}((X, Y) \in C) = \int_{(x,y) \in C} \Pr_{X,Y}(x, y) dx dy.$$

The *marginal density* of  $X$  can be derived from the joint distribution; it is

$$\Pr_X(x) = \int_y \Pr_{X,Y}(x, y) dy.$$

When one or other of the two random variables is discrete, just replace integrals by sums as appropriate.

Two random variables  $X$  and  $Y$  are said to be *independent* if

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

Informally, it means “knowing the value of one of them gives no information about the other.” Two equivalent definitions are

$$\begin{aligned} \Pr_{X,Y}(x, y) &= \Pr_X(x) \Pr_Y(y) \quad \text{for all } x \text{ and } y, \\ \mathbb{P}(X \in A \mid Y \in B) &= \mathbb{P}(X \in A) \quad \text{for all } A \text{ and } B \text{ with } \mathbb{P}(Y \in B) > 0. \end{aligned}$$

The concept of independent random variables is fundamental in modelling. It was introduced informally in *IA Maths for NST*. Working with joint distributions is a key skill for Bayesian inference (section 3.1) and for analyzing ‘exotic’ random variables like random processes (section 6).

**Exercise 1.5.** Let  $X$  and  $Y$  be independent Uniform $[0, 1]$  random variables. Find  $\mathbb{P}(|X - Y| < \delta)$ . Hence find the density of  $|X - Y|$ .

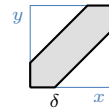
Because they are independent the joint density is

$$\Pr_{X,Y}(x, y) = \Pr_X(x) \Pr_Y(y) = 1.$$

Then,

$$\mathbb{P}(|X - Y| < \delta) = \int_{(x,y) : |x-y| < \delta} 1 dx dy = \int_{x=0}^1 \int_{y : |y-x| < \delta} 1 dy dx = 1 - (1 - \delta)^2$$

and the density is  $\Pr_{|X-Y|}(\delta) = 2(1 - \delta)$ . (You could perhaps guess this, without going via integrals. It's useful to see the integration, to see how it would work for less simple distributions.)



**Example 1.6.** I throw a fair die. Let  $Z$  be the result. Let  $X = Z \bmod 2$  and let  $Y = Z \div 3$ , so for example  $Z = 3$  gives  $X = 1$  and  $Y = 1$ . Show that  $X$  and  $Y$  are not independent. Show that  $X' = (Z - 1) \bmod 2$  and  $Y' = (Z - 1) \div 2$  are.

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. An exhaustive test of all  $x$  and  $y$  shows that  $X'$  and  $Y'$  are independent.

**Exercise 1.7 (Identifying independence by factorization).**

Suppose that  $P_{X,Y}(x, y) = g(x)h(y)$  for some functions  $g$  and  $h$ . Show that  $X$  and  $Y$  are independent, and

$$\Pr_X(x) \propto g(x), \quad \Pr_Y(y) \propto h(y).$$

**Example 1.8.**

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 (using the result from example 1.7) they are independent.

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’.

**RULES FOR CONDITIONAL PROBABILITY**

There are five core definitions and laws in probability theory. In these equations,  $A$  and  $B$  are events.

1.  $\mathbb{P}(\Omega) = 1$  where  $\Omega$  is the entire sample space
2. Conditional probability:  $\mathbb{P}(A | B) = \mathbb{P}(A \cap B) / \mathbb{P}(B)$ , when  $\mathbb{P}(B) > 0$
3. Sum rule: If  $\{B_1, B_2, \dots\}$  partition  $\Omega$  then  $\mathbb{P}(A) = \sum_i \mathbb{P}(A \cap B_i)$   
Law of total probability:  $\mathbb{P}(A) = \sum_i \mathbb{P}(B_i) \mathbb{P}(A | B_i)$
4.  $A$  and  $B$  are said to be *independent* if  $\mathbb{P}(A \cap B) = \mathbb{P}(A) \mathbb{P}(B)$
5. Bayes' rule:

$$\mathbb{P}(A | B) = \frac{\mathbb{P}(A) \mathbb{P}(B | A)}{\mathbb{P}(B)} \quad \text{if } \mathbb{P}(B) > 0$$

These rules have direct translations into statements about joint distributions of random variables. I've written the rules here for discrete random variables, but they also apply to continuous random variables—just replace the sums by integrals<sup>2</sup>.

- 1'. Densities sum to one:  $\sum_x \Pr_X(x) = 1$
- 2'. Conditional density:  $\Pr_X(x | Y = y) = \Pr_{X,Y}(x, y) / \Pr_Y(y)$ , when  $\Pr_Y(y) > 0$
- 3'. Marginal density:  $\Pr_X(x) = \sum_y \Pr_{X,Y}(x, y)$   
Law of total probability:  $\mathbb{P}(A) = \sum_x \Pr_X(x) \mathbb{P}(A | X = x)$
- 4'.  $X$  and  $Y$  are said to be *independent* if  $\Pr_{X,Y}(x, y) = \Pr_X(x) \Pr_Y(y)$
- 5'. Bayes' rule:

$$\Pr_X(x | Y = y) = \frac{\Pr_X(x) \Pr_Y(y | X = x)}{\Pr_Y(y)} \quad \text{if } \Pr_Y(y) > 0$$

All of the laws still work if we stick ‘conditional on  $C$ ’ onto them, if  $\mathbb{P}(C) > 0$ . Such conditional laws are easy to derive. Here are some of them.

- 1''. Densities sum to one:  $\sum_x \Pr_X(x | C) = 1$
- 2''. Conditional probability:  $\mathbb{P}(A | B \cap C) = \mathbb{P}(A \cap B | C) / \mathbb{P}(B | C)$ , when  $\mathbb{P}(B | C) > 0$
- 3''. Law of total probability:  $\mathbb{P}(A | C) = \sum_x \mathbb{P}(X = x | C) \mathbb{P}(A | \{X = x\} \cap C)$
- 4''.  $X$  and  $Y$  are said to be *conditionally independent given  $C$*  if  
 $\Pr_{X,Y}(x, y | C) = \Pr_X(x | C) \Pr_Y(y | C)$ .

<sup>2</sup>The definition of  $\Pr_X(x | Y = y)$  needs subtlety when  $Y$  is a continuous random variable. The event  $\{Y = y\}$  has probability 0 for every  $y$ , so the straightforward definition of conditional probability doesn't work.

‘partition  $\Omega$ ’ means the  $B_i$  are mutually exclusive and  $\bigcup_i B_i = \Omega$



## CONDITIONING ON A RANDOM VARIABLE \*

If we ever try to compute a probability and we end up with a random variable on the right hand side, we've made a mistake. Probabilities are numbers, and random variables are functions, and we should be hyper-vigilant about which is which.

In machine learning we often 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? What we really mean is

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

and as a shorthand for this, we write

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

Here's a formal definition, *conditioning on a random variable*.

$$\mathbb{P}(A \mid Y) \quad \text{means} \quad \text{"Define } h(y) = \mathbb{P}(A \mid Y = y) \text{ then return } h(Y)\text{"}$$

This is a random variable—it's a function of  $Y$ , and  $Y$  is a random variable.

## 1.4. Fitting distributions

**tl;dr.** A *random sample* is a collection of random variables all drawn from the same distribution, and all independent of each other. In mathematical notation, if  $Y = (X_1, \dots, X_n)$  is the random sample and  $y = (x_1, \dots, x_n)$  is a collection of values (i.e. a *dataset*), then

$$\Pr_Y(y) = \Pr_X(x_1) \times \dots \times \Pr_X(x_n)$$

where  $X$  is the common distribution. We also say that  $X_1, \dots, X_n$  are *independent and identically distributed* or *i.i.d.*

Suppose that the distribution of  $X$  depends on some unknown parameter  $\theta$  which we'd like to estimate, given a dataset. The *likelihood* given a single observation is

$$\text{lik}(\theta | x) = \Pr_X(x | \theta)$$

and the likelihood given a dataset is

$$\text{lik}(\theta | x_1, \dots, x_n) = \Pr_X(x_1 | \theta) \times \dots \times \Pr_X(x_n | \theta).$$

*Fitting the distribution* means finding the maximum likelihood estimator for  $\theta$ , i.e. solving

$$\hat{\theta} = \arg \max_{\theta} \log \text{lik}(\theta | x_1, \dots, x_n).$$

`scipy.optimize.fmin` can be used for this.

This definition of likelihood works for both discrete and continuous random variables, whereas the working definition in Section 1.1 only worked for discrete random variables.

**Application.** Suppose 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 random variable (i.e. a random number generator with a particular probability distribution for its output) that produces output consistent with the data. Many standard random variables 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.

\* \* \*

**Example 1.9.** I collected 69,719 lines from the request log of a webserver, and extracted the size in bytes of the response content. The dataset can be found at [https://teachingfiles.blob.core.windows.net/founds/weblog\\_sizes.txt](https://teachingfiles.blob.core.windows.net/founds/weblog_sizes.txt). Fit a geometric distribution to this data. Also fit a normal distribution. Which is the better fit?

parameter. The log likelihood function given a dataset  $x_1, \dots, x_n$  is

$$\begin{aligned}\log \text{lik}(p \mid x_1, \dots, x_n) &= \sum_{i=1}^n \log \Pr_X(x_i \mid p) \\ &= \sum_i (\log p + (x_i - 1) \log(1 - p)) \\ &= n \log p + \left( \sum_i x_i - n \right) \log(1 - p).\end{aligned}$$

To find the maximum likelihood estimator, differentiate with respect to  $p$  and find where the derivative is equal to zero: the answer is

$$\hat{p} = \frac{n}{\sum_i x_i}.$$

For all but the simplest distributions, we'll have to use numerical optimization rather than algebra and calculus.

```
1 URL = "https://teachingfiles.blob.core.windows.net/founds/weblog_sizes.txt"
2 df = pandas.read_csv(URL, header=None, names=['size'])
3 x = df['size'].values
4 x = x[x>0] # restrict attention to non-empty response (why do you think?)
5
6 # Exact solution
7 p = 1 / numpy.mean(x) # less danger of overflow than len(x)/sum(x)
8
9 # Numerical solution (using a scaled loglik, for numerical stability)
10 def loglik(p, x):
11     return numpy.log(p) + (numpy.mean(x) - 1) * numpy.log(1-p)
12 initial_guess = numpy.array([0.5])
13 mle = scipy.optimize.fmin(lambda p: -loglik(p,x), initial_guess)
14 (p,) = mle # unpack mle, which is a list of length 1
```

For the normal distribuion: if  $X \sim \text{Normal}(\mu, \sigma^2)$  then  $X$  is a continuous random variable with probability density function

The Normal distribution:  
page 103 in the appendix

$$\Pr_X(x \mid \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}$$

where  $-\infty < \mu < \infty$  and  $0 \leq \sigma < \infty$  are parameters. The log likelihood function given a dataset  $x_1, \dots, x_n$  is

$$\begin{aligned}\log \text{lik}(\mu, \sigma \mid x_1, \dots, x_n) &= \sum_{i=1}^n \log \Pr_X(x_i \mid \mu, \sigma) \\ &= \sum_i \left( -\frac{1}{2} \log(2\pi\sigma^2) - \frac{(x_i - \mu)^2}{2\sigma^2} \right) \\ &= -\frac{n}{2} \log(2\pi) - n \log \sigma - \frac{\sum_i (x_i - \mu)^2}{2\sigma^2}.\end{aligned}$$

To find the maximum likelihood estimator, differentiate with respect to  $\mu$  and  $\sigma$  and find where the derivative is equal to zero. There are two parameters, so we have a pair of simultaneous equations to solve:

$$\begin{aligned}\frac{d}{d\mu} \log \text{lik} &= -\frac{\sum_i 2(x_i - \mu)}{2\sigma^2} = 0 \\ \frac{d}{d\sigma} \log \text{lik} &= -\frac{n}{\sigma} + \frac{\sum_i (x_i - \mu)^2}{\sigma^3} = 0.\end{aligned}$$

The solution is

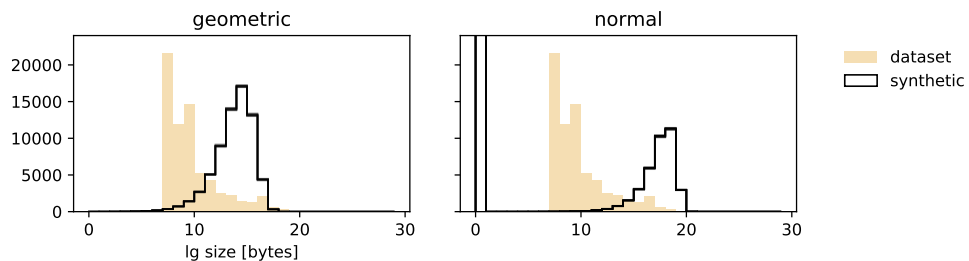
$$\hat{\mu} = \frac{\sum_i x_i}{n}, \quad \hat{\sigma} = \sqrt{\frac{1}{n} \sum_i (x_i - \hat{\mu})^2}.$$

```

15 # Exact solution
16  $\mu$  = np.mean(x)
17  $\sigma$  = np.sqrt(np.mean((x- $\mu$ )**2))
18
19 # Numerical solution (using a scaled loglik, for numerical stability)
20 def loglik( $\mu$ ,  $\sigma$ , x):
21     return -numpy.log( $\sigma$ ) - numpy.mean((x- $\mu$ )**2) / (2* $\sigma$ **2)
22 initial_guess = numpy.array([2**20, 2**10])
23 mle = scipy.optimize.fmin(lambda  $\theta$ : -loglik( $\theta$ [0], $\theta$ [1],x), initial_guess)
24 ( $\mu$ , $\sigma$ ) = mle

```

We are asked “Which is the better fit?” The unstated goal is to find a random number generator whose output is consistent with the dataset, so let’s evaluate this in the most straightforward way we can think of: run the RNG to produce a synthetic dataset, then plot histograms of the real dataset and the synthetic dataset, and compare them by eye. Better, plot multiple synthetic datasets for each RNG—this shows us how much variability there is from one synthetic dataset to another, and lets us judge whether the RNG could plausibly have produced the real dataset. This plot shows 20 synthetic datasets for each RNG (which all overlap), and we can see that neither RNG is any good for this dataset. (The normal distribution plot has a spike at 0: can you work out why, by looking at the code below?)



```

25 # Prepare a list with the two rngs –
26 # It's good practice to separate data science logic from plotting logic.
27 rngs = [( 'geometric', lambda n: numpy.random.geometric(p, size=n)),
28         ( 'normal', lambda n: numpy.random.normal( $\mu$ , $\sigma$ , size=n))]
29
30 import matplotlib.pyplot as plt
31 with plt.rc_context({'figure.figsize': (8,2), 'figure.subplot.wspace': 0.15}):
32     fig = plt.figure()
33     for i,(lbl,rng) in enumerate(rngs):
34         ax = fig.add_subplot(1, 2, i+1)
35         # On a linear x-scale, nearly everything ends up in the lowest bin.
36         # The plot looks better if we use a log scale.
37         ax.hist(numpy.log2(x), bins=numpy.arange(0,30), color='wheat', label='dataset')
38         ax.set_title(lbl)
39         # Generate 20 synthetic datasets, and superimpose them.
40         # We need maximum(y,1) for a log plot, in case the rng produces values ≤ 0
41         for j in range(20):
42             y = rng(len(x))
43             ax.hist(numpy.log2(numpy.maximum(y,1)), bins=numpy.arange(0,30),
44                     histtype='step', color='k', alpha=.2, label='synthetic')
45         ax.set_ylim(0,24000)
46         if i == 0:
47             ax.set_xlabel('lg size [bytes]')
48         else:
49             ax.set_yticklabels([])
50             handles, labels = ax.get_legend_handles_labels()
51             handles[1].set_alpha(1)
52             ax.legend(handles[:2], labels[:2], bbox_to_anchor=(1.1, 1), loc=2, frameon=False)
53 plt.show()

```

## 1.5. Custom distributions

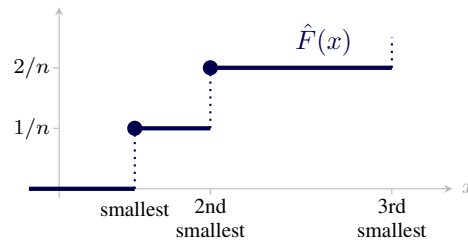
tl;dr. If there aren't standard off-the-shelf distributions that fit our dataset, we can design our own. A good way to design a distribution function is to plot the *empirical cumulative distribution function*

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

and then design a cumulative distribution function  $F(x) = \mathbb{P}(X \leq x)$  that fits.

There is a universal way to generate a random variable given its cumulative distribution function, called the *inversion method*.

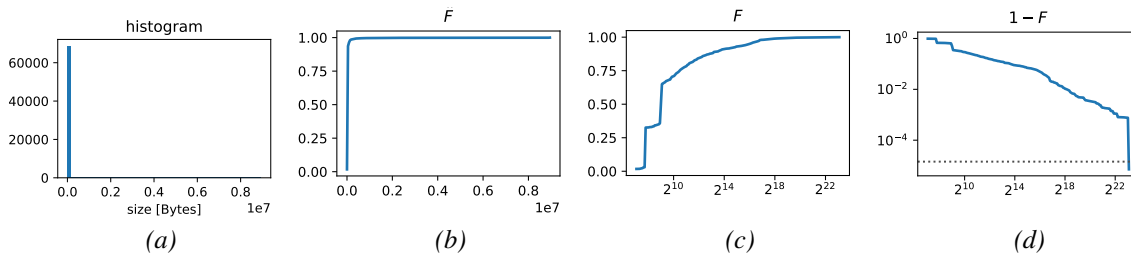
It's easy to plot the empirical distribution function: just sort the data and put it on the  $x$ -axis.



**Example 1.10.** I collected 69,719 lines from the request log of a webserver, and extracted the size in bytes of the response content. The dataset can be found at [https://teachingfiles.blob.core.windows.net/founds/weblog\\_sizes.txt](https://teachingfiles.blob.core.windows.net/founds/weblog_sizes.txt). Design a random number generator that fits this distribution of sizes.

Lets 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. The empirical distribution  $\hat{F}(x)$  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. It looks like there's a slowly decaying curve, perhaps something like  $z \mapsto 1 - e^{-\lambda z}$ , so to see it more clearly I've switched in (d) to plotting  $1 - \hat{F}(x)$  on a log axis.

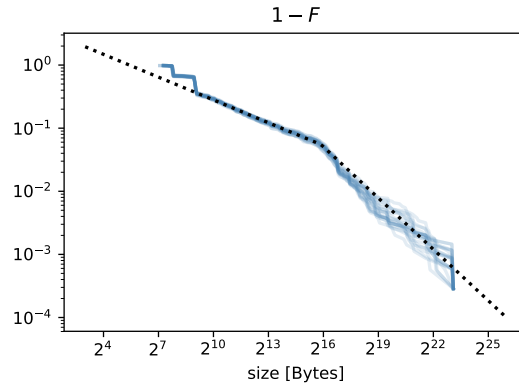
The dotted reference line in (d) is at  $1/n$  where  $n$  is the size of the dataset—it marks 'single datapoint'. Every tick on the  $y$  axis corresponds to 100 times more datapoints. From this, we can read off that the precipitous drop at the right hand side of (d) is around 50 datapoints, all with the same very large size.



Plot (d) looks like the data is trying to tell me that there are two straight lines (plus a handful of huge files, which I'll ignore for now), i.e. that for some parameters  $\alpha$ ,  $\beta$ ,  $\gamma$  and  $\theta$  which I can fit from the data,

$$\log(1 - \hat{F}(x)) \approx \alpha - \beta \log x - \gamma \max(\log x - \theta, 0).$$

Here's what the fit might look like. The slope on the left is  $-\beta$ , the slope on the right is  $-(\beta + \gamma)$ , and the slope changes at  $x = e^\theta$ . I split the data into 20 pieces and superimposed the empirical distributions for each of them, to give a sense of how confident we should be in the exact shape.



The plot shows parameters fitted by eye, but they should really be fitted by maximum likelihood estimation.

```

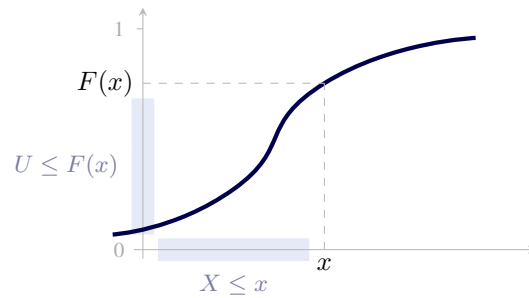
1  URL = "https://teachingfiles.blob.core.windows.net/founds/weblog_sizes.txt"
2  sizes = pandas.read_csv(URL, header=None, names=['size'])['size'].values
3  x = numpy.sort(sizes)
4  ef = numpy.arange(1, len(x)+1)/len(x)
5
6  with plt.rc_context({'figure.figsize': (8,2), 'figure.subplot.wspace': .4}):
7      fig, (ax1, ax2, ax3, ax4) = plt.subplots(1,4)
8      ax1.hist(x, bins=50)
9      ax1.set_xlabel('size [Bytes]')
10     ax1.set_title('histogram')
11     ax2.plot(x, ef, drawstyle='steps-post', linewidth=2)
12     ax2.set_title('$\hat{F}$')
13     ax3.semilogx(x, ef, drawstyle='steps-post', linewidth=2, basex=2)
14     ax3.set_title('$\hat{F}$')
15     ax4.loglog(x, 1-ef, drawstyle='steps-post', linewidth=2, basex=2, basey=10)
16     ax4.axhline(1/len(x), linestyle='dotted', color='0.3')
17     ax4.set_title('$1-\hat{F}$')
18 plt.show()
19
20 alpha, beta, gamma, theta = (1.5, 0.4, 0.5, 11) # picked by eye
21 def F(x):
22     y = alpha - beta*numpy.log(x) - gamma*numpy.maximum(numpy.log(x)-theta, 0)
23     return 1 - numpy.exp(y)
24 newx = numpy.power(2, numpy.linspace(3,26,200))
25 xsplit = numpy.array_split(numpy.random.permutation(sizes), 20)
26
27 for xs in xsplit:
28     efs = numpy.arange(0, len(xs))/len(xs)
29     plt.loglog(numpy.sort(xs), 1-efs, basex=2, basey=10, alpha=0.15, color='steelblue', lin
30 plt.loglog(newx, 1-F(newx), basex=2, basey=10, linestyle='dotted', linewidth=2, zorder=2,
31 plt.xlabel('size [Bytes]')
32 plt.title('$1-\hat{F}$')
33 plt.show()

```

For this problem, we arbitrarily invented a “two straight line” distribution function. Why straight lines, and why only two of them? Is there a systematic way to pick the best RNG to fit a dataset? This is a fundamentally wrong-headed question, as we’ll see in section 2.6.

## THE INVERSION METHOD

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



it ensures that for every  $x$  the event  $\{X \leq x\}$  is precisely the event  $\{U \leq F(x)\}$ , which has probability  $F(x)$ . Intuitively, in regions where the density  $\Pr_X$  is high then  $F$  will be steep, and so  $U$  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<sup>3</sup> for solving  $F(X) = U$  rather than algebra. One very easy case of inverting a staircase will appear in section 2.6.

### COMPARING DISTRIBUTIONS \*

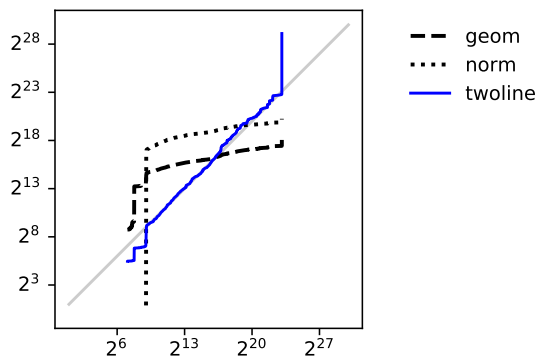
There is a sophisticated plot for comparing distributions, the q-q (quantile-quantile) plot.

Quantile is another name for percentile; the  $q$ -quantile of a random variable  $X$  is the value  $x$  such that  $\mathbb{P}(X \leq x) = q$ , thus  $x = F^{-1}(q)$  where  $F$  is the cumulative distribution function.

Suppose the two cumulative distribution functions we want to compare are  $F(x)$  and  $G(x)$ . The q-q plot shows  $F^{-1}(q)$  against  $G^{-1}(q)$  as  $q$  varies in the range  $[0, 1]$ . If the two distributions are identical, this will produce a straight  $45^\circ$  line. The q-q plot is better than a histogram because it doesn't involve arbitrary binning.

Here is a q-q plot comparing the empirical distribution of the weblog file sizes ( $x$ -axis) to the fitted Geometric and Normal distributions from section 1.4, and to the two-straight-line custom distribution we've just calculated ( $y$ -axis). Our custom distribution is much better, except that it does poorly for small values and large values. If we think hard about what the axes mean, we can read off:

- For  $x \approx 2^7$ , the two-straight-line fit is producing too few values around  $x$ , compared to the data.
- For  $x > 2^{24}$ , the two-straight-line fit is producing too many values around  $x$ .
- The Geometric and Normal fits have the opposite problem.



<sup>3</sup>'percentile': section 1.2.  
In Python, the quantile function is called `ppf`: appendix, page 101.

```

1  p, μ, σ, α, β, γ, θ = ... # estimated parameters from all the fits
2
3  n = len(sizes)
4  q = numpy.arange(1, n+1) / n
5
6  # To compare an empirical distribution to a theoretical distribution, pick q ∈ {1/n, 2/n, ...}
7  # Then,  $\hat{F}^{-1}(i/n)$  is just the  $i$ th smallest value in the dataset,
```

<sup>3</sup>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.

```

8  # so there's no need for any tricky interpolation.
9
10 fig, ax = plt.subplots()
11 ax.loglog([2**1, 2**30], [2**1, 2**30], color='0.8')
12 ax.loglog(numpy.sort(sizes), scipy.stats.geom.ppf(q, p=p), label='geom', linestyle='dashed')
13 ax.loglog(numpy.sort(sizes), scipy.stats.norm.ppf(q, loc= $\mu$ , scale= $\sigma$ ), label='norm', linestyle='solid')
14 # Let twoline_ppf be the quantile function for the two-straight-line fit
15 ax.loglog(numpy.sort(sizes), twoline_ppf(q,  $\alpha$ ,  $\beta$ ,  $\gamma$ ,  $\theta$ ), label='twoline', basex=2, basey=2)
16 ax.legend()
17 ax.set_aspect('equal')
18 ax.legend(bbox_to_anchor=(1.1, 1), loc=2, frameon=False)

```

\* \* \*

In advanced machine learning, it becomes important to be able to measure how close two distributions are. One of the most important measures is called the Kullback-Leibler divergence. It has deep links to average log likelihood, which will be touched on in section 3.3.2.



## 1.6. Fitting a model

**tl;dr.** A *multivariate* dataset contains records each consisting of a tuple of values. Think of it as a spreadsheet: each row is a record, and the columns are fields in the tuple.

Often we want to understand how one item in the tuple depends on the others. The item we want to understand is called the *response variable* or *label* and the others are called *covariates* or *predictors*.

We might invent a probabilistic model in which we treat the response as a random variable, whose distribution depends on both the covariates and on unknown parameters. This is called a *regression model*. We can use maximum likelihood estimation to estimate the unknown parameters; this is called *fitting the model*.

This is all there is to much of machine learning, especially Kaggle-style competitions—the art is inventing models that fit the data well, and for which the parameters give insight.

### Exercise 1.11 (Binomial regression model).

The UK Home Office makes available several datasets of police records, at `data.police.uk`. The stop-and-search dataset has been preprocessed to list the number of stops and the number of those that led to the police finding something suspicious, for each police force and each year.

police_force	year	stops	find
bedfordshire	2017	786	231
cambridgeshire	2016	1691	621
cambridgeshire	2017	581	264

Fit the model  $Y_i \sim \text{Binom}(x_i, p)$  where  $Y_i$  is the number of ‘find’ incidents in a given police force and year,  $x_i$  is the number of stops, and  $p$  is the parameter to estimate.

The binomial distribution is a discrete random variable commonly used for counting the number of successes in a sequence of yes-no trials. If  $X \sim \text{Binom}(n, p)$  then  $n$  is the number of trials,  $0 \leq p \leq 1$  is the success probability, and the probability mass function is

$$\Pr_X(r \mid n, p) = \binom{n}{r} p^r (1-p)^{n-r}, \quad r \in \{0, \dots, n\}.$$

We’ll assume the records in the dataset are independent, since we’re not told otherwise. In maths notation,

$$\Pr(y_1, \dots, y_n \mid p) = \prod_{i=1}^n \binom{x_i}{y_i} p^{y_i} (1-p)^{x_i-y_i}.$$

(Covariates are fixed and known so we’re treating them as constants in this equation, not as parameters.) The log likelihood is

$$\begin{aligned} \log \text{lik}(p \mid y_1, \dots, y_n) &= \sum_i \left( \log \binom{x_i}{y_i} + y_i \log p + (x_i - y_i) \log(1-p) \right) \\ &= \kappa + \left( \sum_i y_i \right) \log p + \left( \sum_i x_i - \sum_i y_i \right) \log(1-p) \end{aligned}$$

where  $\kappa$  is a constant i.e. doesn’t depend on  $p$ . The maximum likelihood estimator for  $p$  solves

$$\frac{d}{dp} \log \text{lik}(p \mid y_1, \dots, y_n) = 0$$

and the solution is

$$\hat{p} = \frac{\sum_i y_i}{\sum_i x_i}.$$

This is not a very interesting model. The only slightly non-obvious thing it’s told us is “don’t estimate  $p$  separately for each police force and year, then average these estimates; instead estimate  $p$  from the whole aggregated data”. The modelling exercise becomes much more interesting when we use it to investigate the influence of multiple covariates, e.g. how gender and race interact. See section 4.1 for a deeper investigation.

Binom is for the  
Binomial random  
variable: appendix  
page 102

Example 1.12 (Softmax cross-entropy loss function \*).

The ImageNet dataset consists of over 14 million images, each hand-annotated with a label. Here are some sample rows:

image	category
	"otter"
	"otter"
	"cello"

Suppose we've built a machine learning algorithm, e.g. a deep convolutional neural network, which takes an image as input and outputs a vector of real-valued scores  $(s_1(x), \dots, s_K(x))$  where  $x$  is an image,  $s_k(x)$  is its score for label  $k$ , and  $K$  is the number of labels. Suppose that this algorithm has parameters that we'd like to estimate from the data.

A data scientist's first instinct is to invent a probability model. The scores are real values, perhaps not in the range  $[0, 1]$ , so we can't use them directly as probabilities. Here's a handy trick: define probabilities by

$$p_k(x) = \frac{e^{s_k(x)}}{e^{s_1(x)} + \dots + e^{s_K(x)}}.$$

This is just an algebraic gimmick, a way to map a vector in  $\mathbb{R}^K$  to probability vector, and it doesn't have any deeper meaning. Any such map would work as long as it's differentiable. This particular map is called *softmax* in machine learning and *multinomial logit* in statistics.

Now, let's propose the probability model

$$\mathbb{P}(Y_i = y \mid \theta) = p_y(x_i), \quad y \in \{1, \dots, K\}$$

where  $x_i$  is the  $i$ th image in the dataset,  $Y_i$  is a random variable for what the label might have been, and  $\theta$  is the vector of parameters that we want to estimate. (We should really write  $p_y(x_i \mid \theta)$  to emphasize that the scoring algorithm depends on the unknown parameters, but the equations get cumbersome.) A fancy way to write this is

$$\mathbb{P}(Y_i = y \mid \theta) = p_1(x_i)^{1_{y=1}} p_2(x_i)^{1_{y=2}} \dots p_K(x_i)^{1_{y=K}} = \prod_{k=1}^K p_k(x_i)^{1_{y=k}}$$

where  $1_{\{\cdot\}}$  stands for the *indicator function*,  $1_{\text{true}} = 1$  and  $1_{\text{false}} = 0$ .

The log likelihood of  $\theta$  given  $n$  image labels  $(y_1, \dots, y_n)$  is

$$\log \text{lik}(\theta \mid y_1, \dots, y_n) = \sum_{i=1}^n \sum_{k=1}^K 1_{y_i=k} \log p_k(x_i),$$

and we should pick  $\theta$  to maximize this. In machine learning, it's more common to see it written as a loss function,  $L(\theta) = -\log \text{lik}(\theta \mid y_1, \dots, y_n)$ , and we should pick  $\theta$  to minimize loss. This particular loss function is called "softmax cross entropy".

In this problem there's no hope of solving for the maximum likelihood estimator using calculus and algebra, for anything other than the most trivial score functions. For deep neural networks,  $\theta$  represents the link weights in the network, and the score function is decidedly non-trivial, and libraries such as TensorFlow are used to find a numerical solution.

What is deeply mysterious in this example is where the probability model comes from. The concept of likelihood only makes sense in a world of probability, but all we were given is a dataset, so we *invented* a counterfactual multiverse of what *might have been*. What's the sense in saying that the first image in the table above might have had some other label like "stoat" or "furniture"? Despite the philosophical shakiness, this seems to work—it's the best approach we have so far for training deep neural networks.

## 2. How random variables behave

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.

### 2.1. Mean and variance

tl;dr. The *mean* or *expectation* of a random variable  $X$  is

$$\mathbb{E} X = \begin{cases} \sum_x x \Pr_X(x) & \text{for a discrete random variable} \\ \int_x x \Pr_X(x) dx & \text{for a continuous random variable.} \end{cases}$$

The *variance* and *standard deviation* are

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

The *conditional expectation* given either an event  $A$  or a random variable  $Y$  is<sup>4</sup>

$$\begin{aligned} \mathbb{E}(X | A) &= \sum_x x \Pr_X(x | A) \quad \text{when } \mathbb{P}(A) > 0 \\ \mathbb{E}(X | Y = y) &= \sum_x x \Pr_X(x | Y = y) \quad \text{when } \Pr_Y(y) > 0 \end{aligned}$$

(and if  $X$  is a continuous random variables, just replace the sum by an integral).

Here are some useful properties about mean and variance. First, a link between probability and expectation:

$$\mathbb{E} 1_{X \in A} = 1 \times \mathbb{P}(1_{X \in A} = 1) + 0 \times \mathbb{P}(1_{X \in A} = 0) = \mathbb{P}(X \in A). \quad (3)$$

$1_{\{\cdot\}}$  stands for the *indicator function*,  $1_{\text{true}} = 1$  and  $1_{\text{false}} = 0$ . Sometimes it's written  $1[\cdot]$ .

For all constants  $a$  and  $b$ , and for any two random variables  $X$  and  $Y$ ,

$$\begin{aligned} \mathbb{E}(aX + b) &= a(\mathbb{E} X) + b \\ \mathbb{E}(X + Y) &= (\mathbb{E} X) + (\mathbb{E} Y). \end{aligned} \quad (4)$$

These two equations are known as “linearity of expectation”. Also, for all constants  $a$  and  $b$ ,

$$\begin{aligned} \text{Var}(aX + b) &= a^2 \text{Var } X \\ \text{std. dev}(aX + b) &= a \text{std. dev}(X). \end{aligned} \quad (5)$$

For any two independent random variables  $X$  and  $Y$ ,

$$\begin{aligned} \mathbb{E}(XY) &= (\mathbb{E} X)(\mathbb{E} Y) \\ \text{Var}(X + Y) &= \text{Var } X + \text{Var } Y \\ \text{std. dev}(X + Y) &= \sqrt{\text{std. dev}(X)^2 + \text{std. dev}(Y)^2}. \end{aligned} \quad (6)$$

When  $X$  and  $Y$  are not independent, it is sometimes useful to measure their dependence by *covariance* or *correlation*,

$$\text{Cov}(X, Y) = \mathbb{E}((X - \mathbb{E} X)(Y - \mathbb{E} Y)), \quad \text{corr}(X, Y) = \frac{\text{Cov}(X, Y)}{\text{std. dev}(X) \text{std. dev}(Y)}.$$

For a function of a random variable  $Y = h(X)$ , there are two ways to work out its expectation:

$$\mathbb{E} Y = \sum_y y \Pr_Y(y) = \sum_x h(x) \Pr_X(x) \quad (7)$$

<sup>4</sup>The definition of  $\mathbb{E}(X | Y = y)$  is subtle, when  $Y$  is a continuous random variable. The event  $\{Y = y\}$  has probability 0 for every  $y$ , so the straightforward definition of conditional expectation doesn't work.

(with sums replaced by integrals for continuous random variables). The first version is the definition of expectation, and the second version is often easier to calculate. This equality is called the *law of the unconscious statistician*, because it's easy to interchange the two versions without even realising one is doing so.

All the properties above also apply to conditional expectation. There is one extra property, the *law of total expectation*, analogous to the law of total probability. Again, replace the sum by an expectation for a continuous random variable.

law of total probability:  
 $\mathbb{P}(A) = \sum_x \Pr_X(x) \mathbb{P}(A|X=x)$ , page 6

$$\mathbb{E} X = \sum_y \mathbb{E}(X | Y = y) \Pr_Y(y).$$

## CONDITIONING ON A RANDOM VARIABLE \*

When we write a conditional expectation given a random variable,  $\mathbb{E}(X | Y)$ , we mean

“Define  $h(y) = \mathbb{E}(X | Y = y)$  then return  $h(Y)$ ”

This is a random variable—it's a function of  $Y$ , and  $Y$  is a random variable. It is analogous to how we defined probability conditional on a random variable. With this notation the law of total expectation can be rewritten as

$$\mathbb{E} X = \mathbb{E}(\mathbb{E}(X | Y)). \quad (8)$$

Sometimes this is written  $\mathbb{E} X = \mathbb{E}_Y(\mathbb{E} X | Y)$ , to emphasize that the outside expectation is over values of  $Y$ . Combining it with indicator functions, equation (3), gives a neat way of writing the law of total probability:

$$\mathbb{P}(X \in A) = \mathbb{E} 1_{X \in A} = \mathbb{E}_Y(\mathbb{E} 1_{X \in A} | Y) = \mathbb{E}_Y \mathbb{P}(X \in A | Y).$$

\* \* \*

**Example 2.1 (Centering and scaling).**

Let  $X$  be a random variable,  $\mu = \mathbb{E} X$  and  $\sigma = \text{std. dev}(X)$ : then

$$\begin{aligned} \mathbb{E}\left(\frac{X - \mu}{\sigma}\right) &= \frac{\mathbb{E} X}{\sigma} - \frac{\mu}{\sigma} = 0 \quad \text{by linearity of expectation} \\ \text{Var}\left(\frac{X - \mu}{\sigma}\right) &= \frac{\text{Var} X}{\sigma^2} = 1 \quad \text{by the rule for variance, equation (5).} \end{aligned}$$

For this reason,  $(X - \mu)/\sigma$  is called the *centered and scaled* version of  $X$ .

**Exercise 2.2.** Show that  $\text{Var} X = \mathbb{E}(X^2) - (\mathbb{E} X)^2$ . Use this to calculate the variance of a  $\text{Binom}(1, p)$  random variable.

*The first part of the question is straight algebra, starting with the definition of variance:*

$$\begin{aligned} \text{Var} X &= \mathbb{E}((X - \mathbb{E} X)^2) \quad \text{by definition of variance} \\ &= \mathbb{E}(X^2 - 2X\mu + \mu^2) \quad \text{where } \mu = \mathbb{E} X \\ &= \mathbb{E}(X^2) - 2\mu \mathbb{E} X + \mu^2 \quad \text{by linearity of expectation} \\ &= \mathbb{E}(X^2) - \mu^2. \end{aligned}$$

If  $X \sim \text{Binom}(1, p)$  then

$$X = \begin{cases} 1 & \text{with probability } p \\ 0 & \text{with probability } 1 - p \end{cases}$$

so  $\mathbb{E} X = p$ . Also  $X^2 = X$  because it only takes values 0 and 1, thus  $\mathbb{E} X^2 = p$ , so  $\text{Var} X = p - p^2 = p(1 - p)$ .

Probability conditional  
on a random variable:  
page 7

Binomial distribution:  
appendix page 102

## 2.2. A rule of thumb for confidence intervals

tl;dr. Here's a rule of thumb that's accurate in a surprisingly wide range of circumstances: a random variable  $X$  can be approximated by

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

$$\text{and so} \quad \mathbb{P}(\mu - 1.96\sigma \leq X \leq \mu + 1.96\sigma) \approx 95\% . \quad (9)$$

(This is why variance is a useful measure of variability.)

If  $X \sim \text{Normal}(\mu, \sigma^2)$  then the confidence interval (9) is exact; and also

$$aX + b \sim \text{Normal}(a\mu + b, a^2\sigma^2) \quad \text{for constants } a \text{ and } b$$

$$X + Y \sim \text{Normal}(\mu + \nu, \sigma^2 + \rho^2) \quad \text{if } Y \text{ is an independent } \text{Normal}(\nu, \rho^2).$$

The approximate confidence interval is so useful and simple that it can't possibly be always true—but what's remarkable is that it's often nearly true. We'll see circumstantial evidence for why the approximation is so good in section 2.3.

**Example 2.3.** Let  $X \sim \text{Bin}(1, 1/2)$ , i.e.  $X = 0$  with probability  $1/2$  and  $X = 1$  with probability  $1/2$ . The rule of thumb suggests  $X \approx \text{Normal}(1/2, 1/4)$ , and a 95% confidence interval  $[-0.48, 1.48]$ . This is clearly rubbish. The rule of thumb doesn't work very well when all we're interested in is a single simple random variable.

**Exercise 2.4.** I throw a die 100 times and compute the total score. What range of values should I expect, assuming the rule of thumb holds?

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, equations (4) and (6),

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

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

**Example 2.5 (Arbitrary confidence intervals).**

Assuming that a random variable  $X$  can be approximated by  $\text{Normal}(\mathbb{E} X, \text{Var} X)$ , find  $\gamma$  such that

$$\mathbb{P}(\mathbb{E} X - \gamma \text{std. dev}(X) \leq X \leq \mathbb{E} X + \gamma \text{std. dev}(X)) \approx 99\% .$$

Give  $\gamma$  in terms of the `scipy.stats.norm.ppf` library routine, which returns  $\Phi^{-1}(x)$  where  $\Phi$  is the cumulative distribution function for a  $\text{Normal}(0, 1)$  random variable. Explain your answer carefully.

This question requires us to unravel the thinking behind the rule of thumb, equation (9). The reference to  $\Phi$  suggests we have to relate  $X$  to a  $\text{Normal}(0, 1)$  random variable. We're told

$$X \approx \text{Normal}(\mu, \sigma^2) \quad \text{where} \quad \mu = \mathbb{E} X, \sigma = \text{std. dev}(X).$$

Thus

$$aX + b \approx a \text{Normal}(\mu, \sigma^2) + b \sim \text{Normal}(a\mu + b, a^2\sigma^2)$$

and by choosing  $a = 1/\sigma$  and  $b = -\mu/\sigma$  we get

$$\frac{X - \mu}{\sigma} \approx \text{Normal}(0, 1).$$

Now we have to figure out how to use the *ppf* function. A quick look at the density function shows us that the  $\text{Normal}(0, 1)$  distribution is symmetric about 0, so in order for

$$\mathbb{P}(-\gamma \leq \text{Normal}(0, 1) \leq \gamma)$$

to be a 99% confidence interval we should pick  $\gamma$  so as to put 0.5% of the probability mass on each of the tails, i.e.

$$-\gamma = \text{ppf}(0.005) \quad \text{and} \quad \gamma = \text{ppf}(0.995).$$

Reassuringly both of these equations yield the same value,  $\gamma = 2.58$ . Putting all of this together,

$$\mathbb{P}\left(-\gamma \leq \frac{X - \mu}{\sigma} \leq \gamma\right) \approx 99\%.$$

All that's left is to rearrange the inequalities:

$$\begin{aligned} (X - \mu)/\sigma \leq \gamma &\iff X \leq \mu + \gamma\sigma \\ (X - \mu)/\sigma \geq -\gamma &\iff X \geq \mu - \gamma\sigma \end{aligned}$$

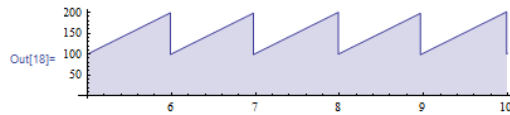
thus

$$\mathbb{P}(\mu - \gamma\sigma \leq X \leq \mu + \gamma\sigma) \approx p \quad \text{for } \gamma = \text{ppf}((1 + p)/2).$$

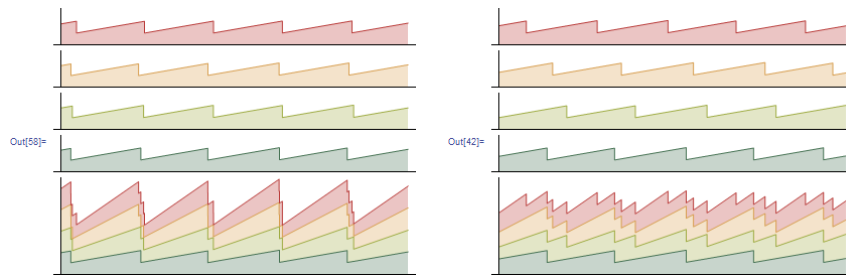


#### Exercise 2.6 (Statistical multiplexing).

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  $x_{\text{peak}}$  be the peak rate. The trough then is  $x_{\text{trough}} = x_{\text{peak}}/2$  because of TCP’s congestion rule. The average is  $x = (x_{\text{trough}} + x_{\text{peak}})/2$  and this we’re told is 30 kB/sec. Solving for  $x_{\text{trough}}$  and  $x_{\text{peak}}$  gives  $x_{\text{trough}} = 2x/3$  and  $x_{\text{peak}} = 4x/3$  which is 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?



Hint. Consider an arbitrary instant in time, and let  $X_1, \dots, X_n$  be the sending rate of each of the  $n = 1000$  flows at this time. Each  $X_i$  might take any value between the trough and the peak, and each value is equally likely, so take  $X_i$  to be a  $\text{Uniform}(2x/3, 4x/3)$  random variable. Find a 95% confidence interval for  $X_1 + \dots + X_n$ .

## 2.3. Convergence theorems

tl;dr. Let  $X_1, X_2, \dots$  be independent identically distributed random variables. Let  $\bar{X}_n$  be the average of the first  $n$  of these,

$$\bar{X}_n = \frac{X_1 + \dots + X_n}{n}.$$

Then

$$\bar{X}_n \rightarrow \mu \quad \text{as } n \rightarrow \infty \quad (10)$$

and furthermore

$$\bar{X}_n \approx \text{Normal}\left(\mu, \frac{\sigma^2}{n}\right) \quad (11)$$

where  $\mu = \mathbb{E} X$  and  $\sigma = \text{std. dev}(X)$ , and  $X$  is the common distribution of the  $X_i$ . Equation (10) is called the *law of large numbers*, and (11) is called the *central limit theorem*.

The rest of this section makes precise exactly what we mean by  $\rightarrow$  in (10) and by  $\approx$  in (11). It's (only just!) beyond the scope of this course to prove the central limit theorem, but we will prove a version of the law of large numbers.

Before being rigorous, let's get intuition for where the parameters in (11) come from. In section 2.2 we saw a general purpose rule of thumb,  $X \approx \text{Normal}(\mathbb{E} X, \text{Var } X)$  — and also saw that it can be crummy for a single random variable! If it did apply here then  $X_i \approx \text{Normal}(\mu, \sigma^2)$  and so, using the rules in section 2.2 for adding and scaling normal random variables,

$$X_1 + \dots + X_n \approx \text{Normal}(n\mu, n\sigma^2) \quad \implies \quad \bar{X}_n \approx \text{Normal}\left(\mu, \frac{\sigma^2}{n}\right).$$

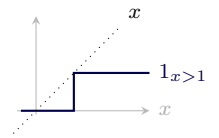
I find it easier to remember the rule of thumb than to remember exactly where  $n$  goes in equation (11).

What's the relationship between the law of large numbers and the central limit theorem? It looks like (11) implies (10), and that's a good way to think about it. There are however some mathematical niceties which make it worth separating the two: for example, there are some random variables with  $\sigma = \infty$  where (11) becomes useless but where (10) still applies.

### LAW OF LARGE NUMBERS

We want to prove that  $\bar{X}_n \rightarrow \mu$ . Let's calculate the probability that they differ by more than some amount  $\varepsilon > 0$ :

$$\begin{aligned} \mathbb{P}(|\bar{X}_n - \mu| > \varepsilon) &= \mathbb{P}\left(\frac{(\bar{X}_n - \mu)^2}{\varepsilon^2} > 1\right) \quad \text{by simple algebra} \\ &= \mathbb{E}\left(1_{\left[\frac{(\bar{X}_n - \mu)^2}{\varepsilon^2} > 1\right]}\right) \quad \text{from page 17, } \mathbb{E} 1_A = \mathbb{P}(A) \\ &\leq \mathbb{E}\left(\frac{(\bar{X}_n - \mu)^2}{\varepsilon^2}\right) \quad \text{since } 1_{x>1} \leq x \text{ for } x \geq 0 \\ &= \frac{1}{\varepsilon^2} \text{Var } \bar{X}_n \quad \text{by linearity of } \mathbb{E} \text{ and definition of Var} \\ &= \frac{1}{n^2 \varepsilon^2} n \text{Var } X \quad \text{by linearity of Var, and independence} \\ &= \sigma^2 / n \varepsilon^2 \\ &\rightarrow 0 \quad \text{as } n \rightarrow \infty. \end{aligned}$$



Thus  $\bar{X}_n$  approaches  $\mu$  as  $n \rightarrow \infty$ ; and the smaller  $\sigma$  the smaller the error is likely to be. The central limit theorem additionally lets us find a confidence interval for the error.

This result, that  $\mathbb{P}(|\bar{X}_n - \mu| > \varepsilon) \rightarrow 0$ , is known as the *weak law of large numbers*. There is related result which says  $\mathbb{P}(\bar{X}_n \rightarrow \mu) = 1$ , called the *strong law of large numbers*. The strong law implies the weak law, but is harder to prove.

## CENTRAL LIMIT THEOREM

Consider  $\sqrt{n}(\bar{X}_n - \mu)$ . It's easy to use the rules for mean and variance to check that

$$\mathbb{E}(\sqrt{n}(\bar{X}_n - \mu)) = 0, \quad \text{Var}(\sqrt{n}(\bar{X}_n - \mu)) = \sigma^2.$$

It can be proven (assuming  $\sigma < \infty$ ) that

$$\mathbb{P}(\sqrt{n}(\bar{X}_n - \mu) \leq x) \rightarrow \mathbb{P}(\text{Normal}(0, \sigma^2) \leq x) \quad \text{as } n \rightarrow \infty \text{ for all } x.$$

This is the precise meaning of the approximation (11).

## CONVERGENCE IN DISTRIBUTION \*

Suppose we have a sequence of random variables  $Y_1, Y_2, \dots$  and another random variable  $Y$ , and

$$\mathbb{P}(Y_n \leq y) \rightarrow \mathbb{P}(Y \leq y) \quad \text{for all } y \text{ where the c.d.f. of } Y \text{ is continuous.}$$

Then we say that the  $Y_i$  converge in distribution to  $Y$ . The central limit theorem says that  $\sqrt{n}(\bar{X}_n - \mu)$  converges in distribution to  $\text{Normal}(0, \sigma^2)$ .

The weak law of large numbers is also a statement about convergence in distribution: it says that  $\bar{X}_n$  converges in distribution to the “constant-valued random variable”  $\mu$ . In maths,

$$\mathbb{P}(\bar{X}_n \leq x) \rightarrow \begin{cases} 0 & \text{if } x < \mu \\ 1 & \text{if } x > \mu \end{cases} \quad \text{as } n \rightarrow \infty.$$



## 2.4. Monte Carlo integration

tl;dr. Let  $X$  be a random variable, and suppose we're interested in  $\mathbb{E} h(X)$  for some function  $h$ . If it's difficult to calculate exactly, we can approximate it by

$$\mathbb{E} h(X) \approx \frac{1}{n} \sum_{i=1}^n h(X_i)$$

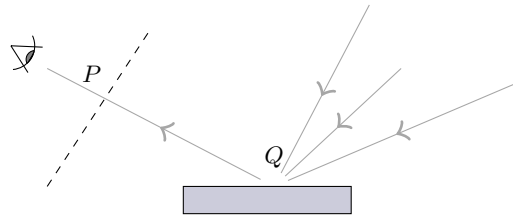
where  $X_1, \dots, X_n$  is a random sample drawn from distribution  $X$ . This is called *Monte Carlo integration*. As a special case,

$$\mathbb{P}(X \in A) = \mathbb{E} 1_{X \in A} \approx \frac{1}{n} \sum_{i=1}^n 1_{X_i \in A}.$$

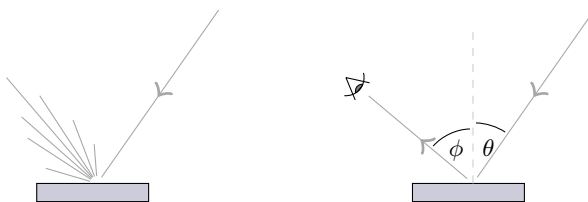
Monte Carlo integration is just a direct application of the limit theorems from section 2.3—so let's focus on what it's used for.

### AN 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  $P$ .



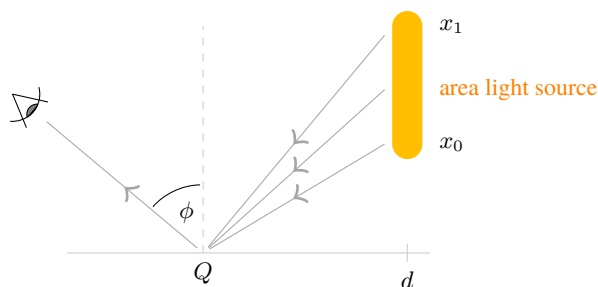
The surface might be perfectly reflective, or perfectly diffuse, or more generally we can model it with a specular lobe function  $\text{BRDF}(\theta, \phi)$ , which measures how much light is emitted at angle  $\phi$  when it comes in at angle  $\theta$ .



When we take into account the intensity of light glancing the surface as a function of angle  $\theta$ , the total light reflected at angle  $\phi$ , 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 a though the total intensity  $I$  is smeared across a set of point light sources:



$$\begin{aligned} \text{reflected light} \\ \text{at angle } \phi \end{aligned} = \int_{x=x_0}^{x_1} \frac{I}{x_1 - x_0} \cos(\tan^{-1}(d/x)) \text{BRDF}(\tan^{-1}(d/x), \phi) dx.$$

Let's streamline this expression. Define

$$h(x) = I \cos(\tan^{-1}(d/x)) \text{BRDF}(\tan^{-1}(d/x), \phi)$$

so that the integral we want is

$$\int_{x=x_0}^{x_1} \frac{1}{x_1 - x_0} h(x) dx.$$

This is exactly  $\mathbb{E} h(X)$  where  $X \sim \text{Uniform}[x_0, x_1]$ . The Monte Carlo procedure says we can approximate it by taking a random sample  $X_1, \dots, X_n$  from distribution  $X$  and computing the average of  $h(X_i)$ . In words,

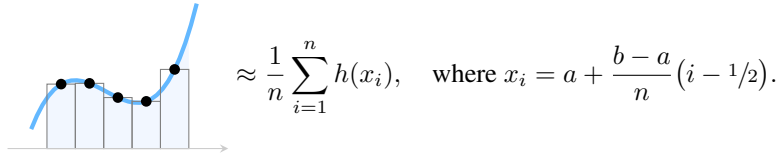
$$\begin{aligned} \text{reflected light} \\ \text{at angle } \phi \end{aligned} \approx \frac{1}{n} \sum_{i=1}^n \begin{array}{l} \text{light due to a simulated ray coming from} \\ \text{a random point } X_i \text{ on the light source.} \end{array}$$

### WHY DOES THIS WORK?

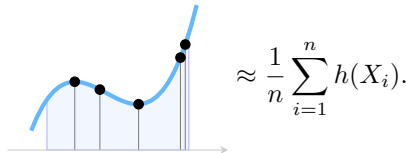
He's some intuition about why this method works, for the case where  $X$  is a uniform random variable. Suppose we're interested in the integral

$$\int_{x=a}^b \frac{1}{b-a} h(x) dx.$$

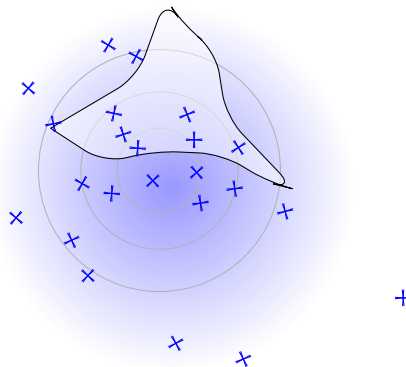
The method you might have learnt at school is to split the  $x$  range into  $n$  equally sized pieces, and approximate the function by a series of rectangles, e.g. taking the height of the rectangle to be the value of  $h$  at the midpoint.



But there's actually nothing special about sampling  $h$  at grid points. Why not just pick the sampling points at random? In other words, pick  $n$  independent  $\text{Uniform}[a, b]$  random variables  $X_1, \dots, X_n$ , and approximate



Here's a different intuition, for the case of estimating  $\mathbb{P}(X \in A)$ . Imagine throwing darts randomly at a dartboard<sup>5</sup>. The distribution of where the dart lands is probably a Normal random variable, centered near the middle of the dartboard, and with a standard deviation that reflects your skill at darts. The probability density is shown by shading in the picture.



<sup>5</sup>The word *stochastic*, a synonym for random, derives from the Greek word *στόχος* meaning 'aim, guess'.

Now imagine there's a picture of a weasel painted on the dartboard. What's the probability that a randomly thrown dart will hit the weasel? There's an easy way to estimate this probability: throw lots of darts, and count what fraction of them hit the weasel. That's all there is to the Monte Carlo probability estimate

$$\mathbb{P}(X \in A) \approx \frac{1}{n} \sum_{i=1}^n 1_{X_i \in A}.$$

#### HOW ACCURATE IS IT?

As a sanity check, does the Monte Carlo estimator even have the right expectation? By linearity of expectation, and using the fact that the  $X_i$  are a random sample i.e. independent and identically distributed,

$$\mathbb{E}\left(\frac{1}{n} \sum_{i=1}^n h(X_i)\right) = \frac{1}{n} \sum_i \mathbb{E} h(X_i) = \mathbb{E} h(X).$$

We can be more precise. Applying the central limit theorem to  $Y_i = h(X_i)$ ,

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

where  $\mu = \mathbb{E} Y = \mathbb{E} h(X)$  and  $\sigma = \text{std. dev}(h(X))$ . By the standard confidence interval for a Normal random variable, the error of the Monte Carlo estimator is  $O(\sigma/\sqrt{n})$ .

Confidence interval for  
Normal: page 19

\* \* \*

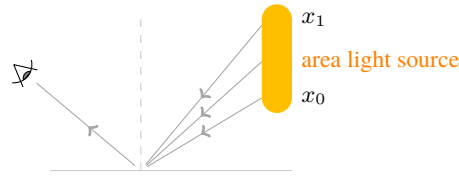
We're using Monte Carlo to approximate  $\mu$ . It's even harder to use maths to calculate  $\sigma$  than it is to calculate  $\mu$ , so the error bound  $O(\sigma/\sqrt{n})$  isn't something we can use directly. But why not just estimate  $\sigma$  with the Monte Carlo method?—it's just

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

and we can plug in the Monte Carlo estimate for  $\mu$  also. In the computer graphics example, we could keep generating more light rays and keep a running estimate of  $\sigma$ , and stop when it implies the error is small enough as to be imperceptible.

## 2.5. Importance sampling \*

**Application.** In the computer graphics problem of Section 2.4, we studied how to use Monte Carlo integration to calculate the reflected light seen by a viewer at a certain pixel.



The answer we want is an integral, and it can be approximated by

$$\int_{x=x_0}^{x_1} \frac{1}{x_1 - x_0} h(x) dx \approx \frac{1}{n} \sum_{i=1}^n \text{light } h(X_i) \text{ from a simulated ray coming from a random point } X_i \text{ on the light source}$$

where  $h(x)$  is some complicated formula involving the specular characteristics of the surface and the angle of illumination. We found that the approximation error is of the order of  $\sigma/\sqrt{n}$ , where  $\sigma$  is the standard deviation of the answer from a single light ray,  $\sigma = \text{std. dev}(h(X_1))$ .

This suggests two questions. First, is there a way to change the simulation to reduce  $\sigma$ ? 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 into the eye of the viewer: 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. This way we won't waste time simulating lots of light rays that don't reflect much in the direction we care about. But we'll need to compensate for this change in distribution. How?

\* \* \*

**General procedure.** Suppose we wish to approximate  $\mathbb{E} h(X)$  using Monte Carlo integration, and suppose we have a *tilted* distribution  $\tilde{X}$  we'd like to simulate from. (In the ray tracing application,  $X$  is uniformly distributed over the light source and  $\tilde{X}$  is some other distribution, it doesn't matter what, that aims to produce more useful simulated light rays.) Now consider the random variable

$$Y = h(\tilde{X}) \frac{\Pr_X(\tilde{X})}{\Pr_{\tilde{X}}(\tilde{X})}.$$

This is cunningly designed with a correction factor so that

$$\mathbb{E} Y = \int_{\tilde{x}} h(\tilde{x}) \frac{\Pr_X(\tilde{x})}{\Pr_{\tilde{X}}(\tilde{x})} \Pr_{\tilde{X}}(\tilde{x}) d\tilde{x} = \int_{\tilde{x}} h(\tilde{x}) \Pr_X(\tilde{x}) d\tilde{x} = \mathbb{E} h(X).$$

Thus we can use Monte Carlo integration, with a random sample drawn from  $\tilde{X}$ :

$$\mathbb{E} h(X) \approx \frac{1}{n} \sum_{i=1}^n h(\tilde{x}_i) \frac{\Pr_X(\tilde{x}_i)}{\Pr_{\tilde{X}}(\tilde{x}_i)}. \quad (12)$$

This is called *importance sampling*. It doesn't matter what distribution we choose for  $\tilde{X}$ , we always get  $\mathbb{E} h(X)$  thanks to the correction factor.

Now, we have a whole design space of tilted distributions to choose from. We should choose the distribution of  $\tilde{X}$  to minimize  $\text{Var } Y$ . It's surprisingly easy to design the optimal tilted distribution.

First,  $\text{Var } Y = \mathbb{E}(Y - \mu)^2 = \mathbb{E} Y^2 - \mu^2$  where  $\mu = \mathbb{E} Y$  doesn't depend on the tilted distribution, so the only term left to minimize is

$$\mathbb{E} \left( h(\tilde{X}) \frac{\Pr_X(\tilde{X})}{\Pr_{\tilde{X}}(\tilde{X})} \right)^2 = \int_{\tilde{x}} \left( \frac{h(\tilde{x}) \Pr_X(\tilde{x})}{\Pr_{\tilde{X}}(\tilde{x})} \right)^2 \Pr_{\tilde{X}}(\tilde{x}) d\tilde{x} = \int_{\tilde{x}} \frac{h(\tilde{x})^2 \Pr_X(\tilde{x})^2}{\Pr_{\tilde{X}}(\tilde{x})} d\tilde{x}$$

and it turns out (using some standard tools from optimization theory) that to minimize this we should pick a distribution for  $\tilde{X}$  with

$$\Pr_{\tilde{X}}(\tilde{x}) \propto h(\tilde{x}) \Pr_X(\tilde{x}). \quad (13)$$

Now the bad news: sampling from this optimal distribution is hard. In fact it's exactly as hard as solving the original problem—if we want to use the inversion method, we have to first find the cumulative distribution function by integrating the density  $h(\tilde{x}) \Pr_X(\tilde{x})$ , which is nothing other than finding  $\mathbb{E} h(X)$ !

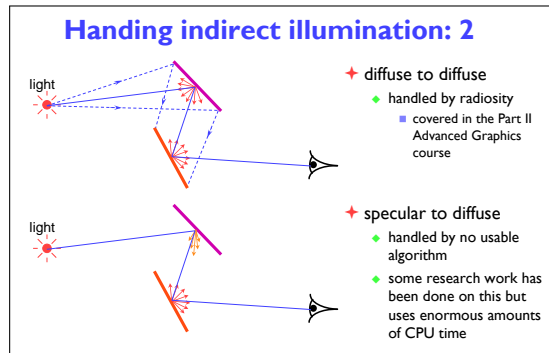
The inversion method, section 1.5, is for sampling from an arbitrary distribution function

\* \* \*

Importance sampling is useful as a heuristic, even if we don't manage to get the perfect tilted distribution. Equation (13) says “Try to sample from a distribution that tilts the original distribution of  $X$  in favour of values with larger  $h(x)$ ”, and equation (12) says “No matter how you sample, you'll still get the right answer (for large enough  $n$ ).” If the tilting is good, it will give the right answer for small  $n$ .

The importance sampling heuristic, applied to distributed ray tracing and 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.”

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.<sup>6</sup>



<sup>6</sup>Slide from IA Introduction to Graphics.

## 2.6. The empirical distribution

tl;dr. The *empirical distribution* of a dataset  $x_1, \dots, x_n$  is

$$\hat{\mathbb{P}}(A) = \frac{1}{n} (\text{how many items there are } \in A)$$

and the *empirical cumulative distribution function* for a numerical dataset is

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

The zen of data science is in seeing datasets and random variables as two sides of the same coin. Empirical distributions *are* probability distributions.

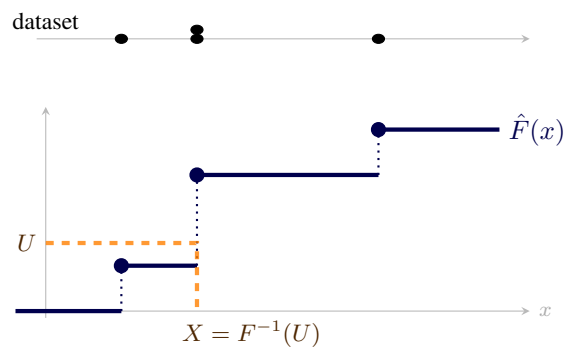
- When the true distribution is unknown, we can use the dataset's empirical distribution instead. There's no need to approximate the dataset by fitting a standard random variable, when we can just *resample* from the dataset.
- When the true distribution is intractable, we can approximate it by the empirical distribution of a random sample. Instead of getting bogged down with integrals, we just use Monte Carlo integration.

### RESAMPLING

Suppose we're given a dataset, and we want to find a distribution that it might have come from. In sections 1.4 and 1.5 we took the approach “Consider a family of distribution functions  $F_\theta(x)$  with some parameter  $\theta$ , and pick a specific parameter value  $\hat{\theta}$  using maximum likelihood; this should make  $\hat{F}(x) \approx F_{\hat{\theta}}(x)$ .”

When all we have is a dataset, how do we choose which family of distributions 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 when a *perfect* fit is staring us in the face, namely the empirical distribution itself! This is a perfectly valid cumulative distribution function (it starts at 0, and ends at 1, and is increasing) and it fits the data perfectly.

We can sample from the empirical cumulative distribution function using the inversion method (section 1.5)—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. This is called *resampling*.



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. We'll see many more examples in section 3.
- 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  $2.5 \times 10^9$  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.

#### Example 2.7 (Exact resampling).

Given a dataset  $x_1, \dots, x_n$ , let  $X^*$  be a random variable drawn from the empirical distribution of the dataset. Show that

$$\mathbb{E} X^* = \frac{1}{n} \sum_{i=1}^n x_i \quad \text{Var } X^* = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2 \text{ where } \bar{x} = \mathbb{E} X^*.$$

*This is one of those questions where it's hard to tell if the answer is so obvious it doesn't need any justification, or so subtle it needs careful justification. Let's be careful.*

As noted above,  $X^*$  can be generated by picking a value at random from the dataset, each item equally likely, i.e. we can generate it by  $X^* = x_I$  where  $I$  is a uniform random variable on  $\{1, \dots, n\}$ . By the law of the unconscious statistician, with the function  $h(i) = x_i$ ,

Law of the unconscious  
statistician: equation (7)  
page 17

$$\mathbb{E} X^* = \sum_{i=1}^n h(i) \mathbb{P}(I = i) = \frac{1}{n} \sum_{i=1}^n x_i$$

and  $\text{Var } X^* = \mathbb{E}(X^* - \mathbb{E} X^*)^2$  can be calculated similarly.

## MONTE CARLO APPROXIMATION

Monte Carlo integration is a method for approximating an expectation,

$$\mathbb{E} h(X) \approx \frac{1}{n} \sum_{i=1}^n h(X_i)$$

where  $X_1, \dots, X_n$  is a random sample drawn from distribution  $X$ . Another way to write this (for a discrete random variable  $X$ ) is

$$\sum_x \Pr_X(x) h(x) \approx \sum_x \hat{P}(x) h(x)$$

where  $\hat{P}$  is the probability mass function for the empirical distribution.

This way of writing it emphasizes a functional programming view of Monte Carlo integration:  $\mathbb{E} h(X)$  can be thought of as an operation which takes as input a probability distribution and returns a value,

$$\Psi : P \mapsto \sum_x P(x) h(x).$$

Assuming the empirical distribution  $\hat{P}$  is close to the true distribution  $\Pr_X$ , we expect  $\Psi(\Pr_X) \approx \Psi(\hat{P})$ .

In other words, if our job is to perform some operation on a probability distribution but this operation is too hard, we can take a random sample, get its empirical distribution, and perform the operation on the empirical distribution rather than on the true distribution. This is the basis for all sorts of approximation algorithms throughout machine learning and data science. We'll apply it to inference problems in section 3.2, and we'll see an application to robotics and computer vision in section 6.4.

## EMPIRICAL DISTRIBUTION $\approx$ TRUE DISTRIBUTION

In sections 1.4 and 1.5 we looked at the problem of fitting a probability distribution to a dataset. The unstated assumption in section 1.5 was that the true cumulative distribution function should be close to

the empirical version. Now, having learnt about convergence theorems in section 2.3, let's investigate this assumption.

**Exercise 2.8.** Consider a random sample  $X_1, \dots, X_n$ , independent random variables with common distribution  $X$ , and let  $F(x) = \mathbb{P}(X \leq x)$ . Let  $\hat{F}_n(x)$  be the empirical distribution function. Find a confidence interval for  $\hat{F}_n(x)$  in terms of  $F(x)$ .

The definition of  $\hat{F}_n(x)$  is

$$\hat{F}_n(x) = \frac{1_{X_1 \leq x} + \dots + 1_{X_n \leq x}}{n}$$

which is crying out for the central limit theorem to be applied. The result is

$$\hat{F}_n(x) \approx \text{Normal}\left(\mu, \frac{\sigma^2}{n}\right) \quad \text{where } \mu = \mathbb{E} 1_{X \leq x}, \sigma^2 = \text{Var} 1_{X \leq x}.$$

We've worked out both of these quantities in section 2.1:

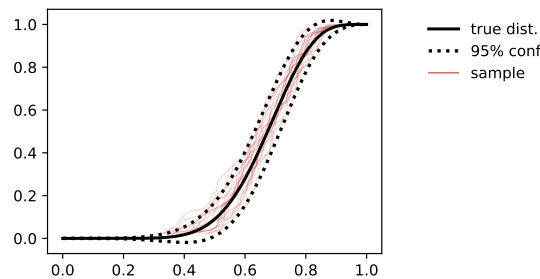
$$\mu = \mathbb{E} 1_{X \leq x} = \mathbb{P}(X \leq x) = F(x) \quad \text{from equation (3) page 17}$$

$$\sigma^2 = \text{Var} 1_{X \leq x} = F(x)(1 - F(x)) \quad \text{from exercise 2.2 page 18.}$$

Thus 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 empirical distribution is a random function, the first example we've seen in this course of an exotic random variable. Section 6 is all about another type of exotic random variable, random infinite sequences. Such random variables lend themselves to a functional programming style with closures, for example

```
1 def rF(n, rng):
2     xs = rng(n) # generate a dataset of size n from the specified r.n.g.
3     def F(x):
4         return sum(xi <= x for xi in xs) / n
5     return F
6 # rF is a function-valued random variable, i.e. every time you call rF() you get a different function.
7
8 myrng = lambda n : numpy.random.exponential(size=n)
9 # For this distribution, the true c.d.f. is F(x) = 1 - e^-x, so F(3) ≈ 0.950213
10
11 F = rF(1000, myrng)
12 F(3) # 0.942 (for example)
13 F(3) # 0.942 (same value as last time)
14 F(3) # 0.942
15
16 F = rF(1000, myrng)
17 F(3) # 0.938
```

Beta distribution:  
appendix page 104

Exp random variable:  
appendix page 102



### 3. Inference

Inference means reaching conclusions on the basis of data and reasoning. In this section of the course, you will learn how to ask useful questions about uncertainty in data.

**Example 3.1.** I have a coin, which might be biased. I toss it 10 times and get 9 heads.

What's the chance that the next toss is heads? Is it  $\frac{1}{2}$  because that's how coins work? Do we conclude from the data that this coin is biased? Are we confident enough to make a bet on heads on odds of 9 to 1? How much more confident would we be if we saw these same frequencies from a hundred coin tosses?

This course is about probabilistic modelling, so the starting point is to write down a model. Section 4 will go into how we come up with good models.

We learnt in section 1 about maximum likelihood estimation as a technique for estimating unknown parameters. We can simply go ahead and use this estimate—in this example, we might simply say “The probability of heads is 0.9, and the probability of seeing two heads in a row is 0.81.” This is called *plug-in estimation*. Or we can look for ways to compute how confident we can be in this estimate. We'll look at a variety of tools in the rest of section 3. All of the tools revolve in one way or another around the likelihood function.

*Let's illustrate what the likelihood function tells us, for this example of coin tosses. The obvious probabilistic model is*

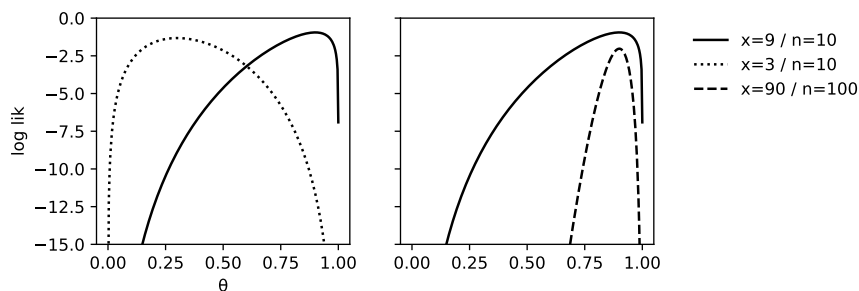
$$X \sim \text{Binom}(n, \theta) \quad \text{i.e.} \quad \Pr_X(x | \theta) = \binom{n}{x} \theta^x (1 - \theta)^{n-x}, \quad x \in \{0, \dots, n\}$$

the binomial distribution,  
page 102 in the appendix

where  $X$  is the number of heads and  $\theta$  is an unknown parameter, the probability of heads. We want to learn about  $\theta$  using the data. The likelihood function is

$$\log \text{lik}(\theta | X = x) = \log \Pr_X(x | \theta) = \log \binom{n}{x} + x \log \theta + (n - x) \log(1 - \theta)$$

and section 1.1 claimed that it measures how much evidence there is for a particular parameter value. Here are some plots.



The maximum likelihood estimator is the peak of the likelihood plot,  $\hat{\theta} = x/n$ . For both 9/10 heads and for 90/100 heads it is at 0.9, and for 90/100 heads the drop-off away from  $\theta = 0.9$  is steeper. Intuitively, for 9/10 heads we'll guess  $\theta = 0.9$  but we're not sure, but for 90/100 heads we're more certain.

```
1 def loglik(theta, x, n):
2     # code carefully to avoid overflow for large n
3     def log_factorial(k): return scipy.special.gammaln(k+1)
4     def log_binom(n, k): return log_factorial(n) - log_factorial(k) - log_factorial(n-k)
5     return log_binom(n, x) + x*numpy.log(theta) + (n-x)*numpy.log(1-theta)
6
7 theta = numpy.linspace(.0001, .9999, 300)
8 with plt.rc_context({'figure.figsize': (6, 2.5)}):
9     fig, (ax1, ax2) = plt.subplots(1, 2, sharex=True, sharey=True)
10    ax1.plot(theta, loglik(theta, 9, 10), label='x=9 / n=10', color='k')
11    ax1.plot(theta, loglik(theta, 3, 10), label='x=3 / n=10', color='k', linestyle='dotted')
```

```
12 ax2.plot( $\theta$ , loglik( $\theta$ , 9, 10), label='x=9 / n=10', color='k')
13 ax2.plot( $\theta$ , loglik( $\theta$ , 90, 100), label='x=90 / n=100', color='k', linestyle='dashed')
14 ax1.set_ylim(-15,0)
15 ax1.set_ylabel('log lik')
16 ax1.set_xlabel('θ')
17 h1,l1 = ax1.get_legend_handles_labels()
18 h2,l2 = ax2.get_legend_handles_labels()
19 plt.legend(h1+h2[1:], l1+l2[1:], loc=2, bbox_to_anchor=(1.1,1), frameon=False)
```

### 3.1. Bayesianism

**tl;dr.** Suppose we have a probabilistic model  $\Pr_X(x|\theta)$  where  $X$  is a random variable representing possible outcomes of an experiment and  $\theta$  is the unknown parameters in the model. Bayesianism says we should represent our uncertainty about unknown parameters by using a probability distribution. We start with a *prior distribution* with density  $\Pr_\Theta(\theta)$  for  $\theta$ , then observe data  $x$ , then calculate a *posterior distribution* with density  $\Pr_\Theta(\theta | X = x)$ , by using Bayes's rule:

$$\Pr_\Theta(\theta | X = x) = \frac{\Pr_\Theta(\theta) \Pr_X(x | \theta)}{\int_\phi \Pr_\Theta(\phi) \Pr_X(x | \phi) d\phi}$$

(with the integral replaced by a sum, if  $\Theta$  is a discrete random variable). It's common to write this as

$$\Pr_\Theta(\theta | X = x) \propto \Pr_\Theta(\theta) \Pr_X(x | \theta)$$

where the constant of proportionality is whatever is needed to make the left hand side be a density function, i.e. to integrate to one.

Any conclusions we want to draw about the unknown parameters should simply be written out as probabilities about  $(\Theta | X = x)$ . Standard readouts are *posterior confidence intervals*, *posterior point estimates*, and *posterior predictive probabilities*.

Bayesian inference makes heavy use of conditioning. Review conditional random variables on page 2, and the rules of conditional distributions on page 6 especially Bayes' rule for continuous random variables.

#### Example 3.2.

I have a coin, which might be biased. I toss it  $n = 10$  times and get  $x = 9$  heads. Is the coin biased? Use the probability model

$$X \sim \text{Binom}(n, \theta) \quad \text{i.e.} \quad \Pr_X(x) = \binom{n}{x} \theta^x (1 - \theta)^{n-x}, \quad x \in \{0, \dots, n\}$$

where  $X$  is the number of heads and  $\theta$  is the unknown parameter, the probability of heads.

#### 3.1.1. FINDING THE POSTERIOR DISTRIBUTION

**Prior distribution.** Bayesianism requires us to set down a prior belief about  $\theta$ . If we don't have a prior belief, Bayesianism says, then there are no grounds for us to draw conclusions. Let's invent out of thin air a prior distribution

$$\Theta \sim \text{Uniform}[0, 1] \quad \text{i.e.} \quad \Pr_\Theta(\theta) = 1, \quad \theta \in [0, 1].$$

There are certain pragmatic choices for prior distributions, which you'll gain experience of as you work through Bayesian exercises. These are cases where the posterior distribution ends up belonging to the same family as the prior distribution, just with different parameters; they are called *conjugate priors*.

**Bayes update.** Applying Bayes's rule, the posterior distribution has density

$$\begin{aligned} \Pr_\Theta(\theta | X = x) &= \frac{1}{\kappa} \Pr_\Theta(\theta) \Pr_X(x | \theta) \\ &= \frac{1}{\kappa} \binom{n}{x} \theta^x (1 - \theta)^{n-x} \\ &= \frac{1}{\kappa'} \theta^x (1 - \theta)^{n-x} \end{aligned} \tag{14}$$

where the normalizing constant is chosen so that the posterior density integrates to one,

$$\kappa' = \int_{\phi=0}^1 \phi^x (1 - \phi)^{n-x} d\phi.$$

Equation (14) is the density function of the  $\text{Beta}(x + 1, n - x + 1)$  distribution. We don't even need

Beta distribution:  
appendix page 104

to work out  $\kappa'$ , we just need to recognize the terms involving  $\theta$ , spot that they're the same as for the Beta distribution, and look up on Wikipedia to see  $\kappa' = x!(n-x)!/n!$ .

**Monte Carlo computational methods.** In nearly every Bayesian calculation outside introductory textbooks, the posterior density is a formula that we can't do anything with analytically—we can't even integrate it to find the normalizing constant. This doesn't actually matter! The lesson from section 2.6 is that we can just as well work with the empirical distribution, all we need is a sample.

So, how do we sample from the posterior distribution? This is actually a huge field of study. There is one method in particular called the Gibbs sampler, which you will learn about in Part II *Machine Learning and Bayesian Inference*. It is based on Markov chains, the topic of section 6.

### 3.1.2. READOUTS FROM POSTERIOR DISTRIBUTIONS

The question tells us  $x = 9$  and asks “Is the coin biased?”, which we might interpret as

$$\mathbb{P}(\Theta \neq 1/2 \mid X = 9)$$

but this is zero because we took the parameter to have a continuous distribution, and for any continuous distributions the probability of observing a single exact value is always zero. We need to pose a better question, e.g.

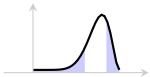
$$\mathbb{P}(\Theta > 1/2 \mid X = 9)$$

which in Python is

```
1 n,x = (10,9)
2 1 - scipy.stats.beta.cdf(0.5, a=x+1, b=n-x+1) # 0.994
```

Alternatively, the computationally-minded Bayesian could answer this by taking a random sample from the posterior distribution then using Monte Carlo integration:

```
3 theta_sample = numpy.random.beta(a=x+1, b=n-x+1, size=10000)
4 numpy.mean(theta_sample > 0.5) # 0.994
```



**Posterior confidence interval.** We could also report a confidence interval for  $(\Theta \mid X = x)$ , i.e. find values  $lo$  and  $hi$  such that

$$\mathbb{P}(\Theta \in [lo, hi] \mid X = x) = 95\%.$$

An obvious way to do this is to pick  $lo$  and  $hi$  so that  $\mathbb{P}(\Theta \leq lo \mid X = x) = 0.025$  and  $\mathbb{P}(\Theta \leq hi \mid X = x) = 0.975$ . In Python,

```
5 lo,hi = scipy.stats.beta.ppf([0.025, 0.975], a=x+1, b=n-x+1) # (0.587, 0.977)
```

Or the Monte Carlo approach:

```
6 lo,hi = numpy.quantile(theta_sample, [.025, 0.975]) # (0.595, 0.977)
```

**Posterior point estimates.** You might be asked “Given the observed data, what is  $\theta$ ?” You should sanctimoniously reply “As a Bayesian, I don’t believe that is a meaningful question; all I will tell you is the posterior distribution of  $(\Theta \mid X = x)$ .” You might then be told “Give me an estimate or you’re fired”. In this situation, the Bayesian has a choice.

- Report the value of  $\theta$  that maximizes  $P_{\Theta}(\theta \mid X = x)$ , i.e. the mode of the posterior distribution. This is called the *maximum a posteriori* (MAP) estimate—a fancy name to give this estimate the facade of rigour. It’s easy to check that if the prior distribution is uniform, then the MAP estimate is exactly the same as the maximum likelihood estimate. In this case, the posterior distribution is  $\text{Beta}(x+1, n-x+1)$  and the mode is  $x/n$ , which is 0.9 in our example.
- Report some straightforward summary of the posterior distribution, such as the mean or median. In this case, the posterior mean is  $(x+1)/(n+2) \approx 0.833$  and the median is `scipy.stats.beta.median(a=x+1,b=n-x+1) ≈ 0.852`.
- Maybe there is a natural *loss function*  $L(\phi, \theta)$ , which measures the price you pay if you report the estimate  $\phi$  and the true value is  $\theta$ . Then you should report  $\hat{\theta}$  to minimize the *expected posterior loss*,

$$\hat{\theta} = \arg \min_{\phi} \mathbb{E} L(\phi, \Theta).$$

where the expectation is taken over the random variable  $\Theta$ . With some reasonably simple calculus, one can show that for  $L(\phi, \theta) = (\phi - \theta)^2$  the solution is to report the posterior mean, and for  $L(\phi, \theta) = |\phi - \theta|$  the solution is to report the posterior median.

**Posterior predictive probability.** Given the observed data, what is the probability that the next coin will be heads? One answer is “The probability is a random variable  $\Theta$ , and I can tell you its posterior distribution.” Another way to interpret the question is that it’s asking about the probability of a well-defined event which happens to depend on some random variables, namely the next coin toss itself and also  $\Theta$ . To calculate probabilities based on random variables, we can use the law of total probability. Let  $X'$  be the outcome of the next coin toss; then

law of total probability:  
see section 1.3 page 6

$$\begin{aligned}\mathbb{P}(X' = \text{heads} \mid X = x) &= \int_{\theta} \Pr_{\Theta}(\theta \mid X = x) \mathbb{P}(X' = \text{heads} \mid \Theta = \theta, X = x) dx \\ &= \int_{\theta} \Pr_{\Theta}(\theta \mid X = x) \theta dx \\ &= \mathbb{E}(\Theta \mid X = x) = \frac{x+1}{n+2} \approx 0.833 \quad (\text{the posterior mean}).\end{aligned}$$

Or the Monte Carlo approach:

```
7 numpy.mean(theta_sample) # 0.834
```

This answer is called the *posterior predictive probability*. It averages over all the uncertain parameter estimates using their posterior distributions. We can use this method to answer any question about future outcomes, e.g. “what is the probability that the next  $m$  coins are heads?” It’s appealing because we can communicate the answer purely in terms of the observed data, not in terms of parameters that are only in the mind of the modeller.

#### Example 3.3 (Nuisance parameters).

Given a random sample from distribution  $X \sim \text{Uniform}[\theta, \theta + \phi]$ , and using the prior distributions  $\Theta \sim \text{Exp}(\lambda_0)$  and  $\Phi \sim \text{Exp}(\mu_0)$ , find the posterior density of  $\phi$ .

In this question,  $\theta$  is called a *nuisance parameter*. It’s part of the probability model, but we’re not interested in its value. In such cases we must first work out the posterior distribution for *all* the unknown parameters, because that’s what Bayes’ rule tells us to do. From this joint distribution we can extract the information we want, for example by marginalizing out the nuisance parameters.

First, write out the priors and the density from the model:

For working with joint distributions, see section 1.3

$$\begin{aligned}\Pr_{\Theta}(\theta) &= \lambda_0 e^{-\lambda_0 \theta} \\ \Pr_{\Phi}(\phi) &= \mu_0 e^{-\mu_0 \phi} \\ \Pr(x_1, \dots, x_n \mid \theta, \phi) &= \prod_{i=1}^n \left( \frac{1}{\phi} 1_{x_i \in [\theta, \theta + \phi]} \right) = \frac{1}{\phi^n} 1_{m \geq \theta} 1_{M \leq \theta + \phi}\end{aligned}$$

where  $m = \min_i x_i$  and  $M = \max_i x_i$ . (In this problem the endpoints of the  $X$  distribution are themselves parameters, so it’s a good idea to write out the density in full using indicator functions.) So the posterior is

$$\Pr_{\Theta, \Phi}(\theta, \phi \mid \text{data}) = \frac{1}{\kappa} \lambda_0 e^{-\lambda_0 \theta} \mu_0 e^{-\mu_0 \phi} \frac{1}{\phi^n} 1_{\theta \leq m} 1_{\theta + \phi \geq M}.$$

The posterior density for  $\phi$  can be found by marginalizing out the nuisance parameter:

$$\begin{aligned}\Pr_{\Phi}(\phi \mid \text{data}) &= \int_{\theta} \Pr_{\Theta, \Phi}(\theta, \phi \mid \text{data}) d\theta \\ &= \frac{1}{\kappa} \frac{\mu_0 e^{-\mu_0 \phi}}{\phi^n} \int_{\theta} \lambda_0 e^{-\lambda_0 \theta} 1_{\theta \leq m} 1_{\theta \geq M - \phi} d\theta \\ &= \frac{1}{\kappa} \frac{\mu_0 e^{-\mu_0 \phi}}{\phi^n} \left( e^{-\lambda_0 \max(M - \phi, 0)} - e^{-\lambda_0 m} \right) 1_{\phi \geq M - m}.\end{aligned}$$

This doesn’t look like any standard distribution, so we’ll just stop here and not bother working out the normalizing constant.

**Exercise 3.4.** Repeat the analysis of example 3.2 but using as prior  $\Theta \sim \text{Beta}(\alpha, \beta)$  for  $\alpha > 0$  and  $\beta > 0$ . Show that the posterior distribution is

$$(\Theta \mid X = x) \sim \text{Beta}(\alpha + x, \beta + n - x).$$

\* \* \*

It's worth distinguishing between Bayes's rule and Bayesianism. Bayes's rule is a theorem about conditional probability,

$$\mathbb{P}(A \mid B) = \frac{\mathbb{P}(A) \mathbb{P}(B \mid A)}{\mathbb{P}(B)}.$$

Bayesianism is the doctrine that one should represent uncertainty about unknown parameters by describing them as random variables. If we accept Bayesianism, then Bayes's rule crops up naturally.

Bayesianism insists that we set down a prior belief for all unknown parameters before we even think about incorporating data. But where are we meant to get the prior from, if not data? See the discussion in section 3.3.3.

In well-chosen models, it shouldn't matter too much what the prior is. For example, in exercise 3.4, if  $n$  is very large then  $\alpha$  and  $\beta$  have little influence. If we have too little data then the prior distribution will have a big impact on our answer. Bayesianism makes it easy to crank a handle and get out answers—but you should always stop and reflect whether your answers really reflect the data or whether they just reflect the assumptions you put in.

## 3.2. Frequentism

**tl;dr.** Suppose we have a probabilistic model  $\Pr_X(x|\theta)$  where  $X$  is a random variable representing possible outcomes of an experiment and  $\theta$  is an unknown parameter. Frequentism says that  $\theta$  is fixed but unknown, and we should base our thinking on all the ways that the experiment might have turned out but didn't.

The frequentist is interested in *output procedures* such as

```
1 def confint(x):
2     lo = ... #some function of x
3     hi = ... #some function of x
4     print("θ is in [{}, {}]" .format(lo, hi))
```

Running `confint(X)` will print a true statement or a false statement, depending on the random variable  $X$ , and the frequentist is interested in designing output procedures that prints a false statement with no more than 5% probability, whatever the value of  $\theta$ . This is referred to as bounding the *error probability* of the procedure.

There is a general-purpose computational method called *bootstrap resampling* for estimating the error probability of an output procedure.

The frequentist imagines a “counterfactual multiverse” of all the ways that the experiment might have turned out. We happen to be in one of these universes, but we don't know which, so we'll only take actions that we know are safe across most of the multiverse.

### Example 3.5.

I have a coin, which might be biased. I toss it  $n = 10$  times and get  $x = 9$  heads. Is the coin biased? Use the probability model

$$X \sim \text{Binom}(n, \theta) \quad \text{i.e.} \quad \Pr_X(x) = \binom{n}{x} \theta^x (1 - \theta)^{n-x}, \quad x \in \{0, \dots, n\}$$

where  $X$  is the number of heads and  $\theta$  is an unknown parameter, the probability of heads.

### 3.2.1. CLASSIC ANALYSIS

**An output procedure for confidence intervals.** Let's invent a totally arbitrary output procedure, which returns a confidence interval for  $\theta$ .

```
1 n, δ = 10, 0.2
2 def confint(x):
3     mle = x / n
4     print("θ is in [{}, {}]" .format(mle-δ, mle+δ))
```

Why this interval? The variable `mle` is actually the maximum likelihood estimator for  $\theta$  and so it's reasonable to guess they should be close. And  $\delta$  is just a tuneable parameter to control the error probability.

The maximum likelihood calculation for Binom is in section 1.1

**Bounding the error probability.** To bound the error probability of `confint(X)`, let's start by rewriting it as a straightforward statement about  $X$ :

$$\begin{aligned} \mathbb{P}(\text{confint}(X) \text{ true}) &= \mathbb{P}(\theta \in [\text{mle} - \delta, \text{mle} + \delta]) \\ &= \mathbb{P}(\theta \geq \text{mle} - \delta \text{ and } \theta \leq \text{mle} + \delta) \\ &= \mathbb{P}(\text{mle} \leq \theta + \delta \text{ and } \text{mle} \geq \theta - \delta) \\ &= \mathbb{P}(\theta - \delta \leq \text{mle} \leq \theta + \delta) \\ &= \mathbb{P}(n(\theta - \delta) \leq X \leq n(\theta + \delta)) \\ &= \text{cdf}(n(\theta + \delta)) - \text{cdf}(n(\theta - \delta)) + \text{pmf}(n(\theta - \delta)) \end{aligned} \tag{15}$$

where  $\text{cdf}(x)$  is  $\mathbb{P}(X \leq x)$  and  $\text{pmf}(x)$  is  $\mathbb{P}(X = x)$ . We can compute a lower bound for this numerically using `scipy.optimize.fmin`. This library routine requires a starting guess, and it's wise to plot the error function to make sure the starting guess avoids local minima.

```

5 def probtrue(theta):
6     def cdf(x): return scipy.stats.binom.cdf(x, n=n, p=theta)
7     def pmf(x): return scipy.stats.binom.pmf(x, n=n, p=theta)
8     return cdf(n*(theta+delta)) - cdf(n*(theta-delta)) + pmf(n*(theta-delta))
9 scipy.optimize.fmin(probtrue, .5)
10 # Optimization terminated successfully.
11 # Current function value: 0.773474
12 # Iterations: 12
13 # Function evaluations: 24
14
15 # As a sanity check: run the optimizer multiple times with different starting points
16 res = [scipy.optimize.fmin(probtrue, random.random(), full_output=1, disp=False)
17        for _ in range(20)]
18 min(p for _, p, *_ in res) # 0.77345

```

Thus  $\mathbb{P}(\text{confint}(X) \text{ true}) \geq 77.3\%$ .

**Interpretation.** We were told  $x = 9$ , so we call `confint(9)` and it prints out " $\theta$  is in  $[0.7, 1.1]$ ". Also, we know that `confint` prints a true statement at least 77.3% of the time. Can we conclude

$$\mathbb{P}(\theta \in [0.7, 1.1]) \geq 77.3\% ?$$

No. The parameter  $\theta$  is fixed and unknown, and it may be inside the range or it may be outside, so the probability is either 0 or 1 and we don't know which. What we should really say is

$$\mathbb{P}(\text{confint}(X) \text{ true}) \geq 77.3\%, \quad \text{confint}(x) \text{ says } \theta \in [0.7, 1.1].$$

The probability statement on the left is about a procedure that we could run on any hypothetical dataset, and the statement on the right is based on the dataset we actually saw. This pair of statements is commonly abbreviated

"a 77.3% confidence interval for  $\theta$  is  $[0.7, 1.1]$ ."

### 3.2.2. RESAMPLING

The heart of the frequentist's analysis is "If this trial were run again, what result might I see?" Modern frequentist analysis asks the same question but in a data-driven and computer-driven way:

*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?*

There are two general strategies. We've seen them before, for the simple case of sampling from a random distribution:

- The parametric approach is (i) specify a probability model with unknown parameters, (ii) fit the parameters using maximum likelihood, (iii) sample from the fitted distribution (section 1.4–1.5)
- The non-parametric approach is simply to sample from the empirical distribution, which is equivalent to picking a value at random from the observed dataset (section 2.6)

These two strategies apply to richer problems, as illustrated in the next two examples.

#### Example 3.6 (Non-parametric 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  $\bar{X}_n^*$  because there are so many possible values that  $\bar{X}_n^*$  might take—but it's straightforward to use Monte Carlo integration instead, to find e.g.  $\mathbb{P}(\bar{X}_n^* \geq x)$  or to draw a histogram.*



```

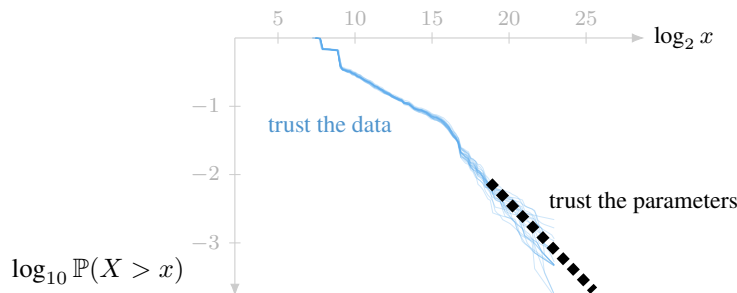
1 xs = [13, 5, 2, ...] # the dataset
2 def sim_mean():
3     n = len(xs)
4     X = random.choices(xs, k=n) # k = number of samples to draw
5     return sum(X) / n
6
7 mc_samples = [sim_mean() for i in range(100000)]
8 matplotlib.pyplot.hist(mc_samples)

```

#### Example 3.7 (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 roughly straight line on a log-plot of the empirical distribution, as we saw in the weblog dataset in section 1.5. 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  $> m$ .



#### 3.2.3. THE BOOTSTRAP

*This is the least rigorous maths I've ever come across.*<sup>7</sup>

Here is a general-purpose computational method, which removes any need for clever maths or exhaustive optimization for bounding the error probability.

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 resampled versions. 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.

Let's apply it to the problem at hand, computing  $\mathbb{P}(\text{confint}(X) \text{ true})$  for the confint function defined in section 3.2.1. As in (15) we want to compute

$$\mathbb{P}(\theta \in [\text{mle} - \delta, \text{mle} + \delta]), \quad \text{where } \text{mle} = X/n. \quad (16)$$

This has a random variable inside, namely  $X$ , so it's a genuine probability as required by step 1.

Step 2 is to replace terms. The maximum likelihood estimator (given the data) is  $\hat{\theta} = x/n$ , so replace  $\theta$  by  $x/n$  in (16). And  $\text{mle}$  is a random variable,  $\text{mle} = X/n$ , so replace it by a resampled

<sup>7</sup>Anonymous student feedback from 2017/2018

version. A reasonable resampling approach in this case is to let  $X^*$  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  $X^* \sim \text{Binom}(n, x/n)$ . Putting all this together, we have obtained the expression

$$\mathbb{P}(x/n \in [X^*/n - \delta, X^*/n + \delta]) \quad \text{where } X^* \sim \text{Binom}(n, x/n).$$

Step 3 is to use the Monte Carlo method to estimate this probability:

```

1  n, x, delta = 10, 9, 0.2
2  Xstar = numpy.random.binomial(n, x/n, size=10000)
3  numpy.mean((x/n >= Xstar/n - delta) & (x/n <= Xstar/n + delta))
4  # returns: 0.931

```

**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.

One particular type of resampling is especially important in machine learning: cross validation. We’ll see it in section 3.3.2, and discuss guidelines for making sure it works robustly.

**Exercise 3.8.** The classic analysis concluded that  $\mathbb{P}(\text{confint}(X) \text{ true}) \geq 77.3\%$  for any value of  $\theta$ . The bootstrap resampling method concluded that this probability is  $\approx 93.1\%$ . Does this indicate that the bootstrap resampling method is a bad approximation?

Hint: Plot  $\text{probtrue}(\theta)$ . What is the value at  $\theta = 0.9$ ?

### 3.3. Model selection

tl;dr. When there are several models that we could fit to the data, how do we choose one? This is a profoundly subtle question, and different disciplines have taken different approaches.

*All models are false, but some are useful.*<sup>8</sup>

This epigram, by the statistician George Box, begs the question: what are we going to use the models for? We'll work through three standard approaches (Bayesian, hypothesis testing, and cross validation), using a toy dataset.

Einstein said “Everything should be made as simple as possible, but not simpler.”<sup>9</sup> Science sees a model as an *explanation* for the data, and scientists have a gut instinct that simple models are likely to be closer to the truth and to generalize better to new scenarios. Engineers on the other hand see a model as a piece of machinery that will be deployed, often with the job of making *predictions* about new data; the best model for them is the one that makes the best predictions.

Shouldn't the best explanation lead to the best predictions? And if a model makes better predictions doesn't that mean it's the better explanation? No, and this photograph<sup>10</sup> of goats in a tree illustrates why.



Photographer's caption:  
“These are the trees that grow the argane nuts that are ground into argane oil. They climb the trees to eat the nuts.”

AI caption by Microsoft Azure: “A group of giraffes standing next to a tree”.

This AI likely never saw goats in trees in its training dataset (though allegedly it saw an overabundance of giraffes), and so it's perfectly reasonable for it to infer the rule “if there's an animal in a tree it's not a goat”. This rule, or something like it, could well improve its prediction scores *on the training dataset*. But the rule isn't a good explanation, because it's not simple enough to generalize to new scenarios like argane trees.

In this section we'll examine explanation versus prediction for a toy example. It's taken from “To explain or predict?” by Galit Shmueli<sup>11</sup>, who explains at length how different academic disciplines see model selection. See also the discussion by Brian Ripley<sup>12</sup>.

**Example 3.9.** As a data scientist, you have been given a dataset `shmueli`<sup>13</sup> with three columns,  $y$ ,  $x_1$ , and  $x_2$ . You believe the underlying model is either

$$\text{Model A : } Y_i \sim \text{Normal}(\alpha + \beta_1 x_{1,i} + \beta_2 x_{2,i}, \sigma^2)$$

<sup>8</sup>G. 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>

<sup>9</sup>Actual not-so-simple quotation: “It can scarcely be denied that the supreme goal of all theory is to make the irreducible basic elements as simple and as few as possible without having to surrender the adequate representation of a single datum of experience.” Albert Einstein. “On the Method of Theoretical Physics”. In: *Philosophy of Science* 1.2 (1934), pp. 163–169. URL: <http://www.jstor.org/stable/184387>

<sup>10</sup>Photo by Fred Dunn, [flickr.com/photos/gratapictures/17208409348/](https://www.flickr.com/photos/gratapictures/17208409348/), CC BY-NC. This example is taken from [aiweirdness.com](http://aiweirdness.com), a blog by Janelle Shane.

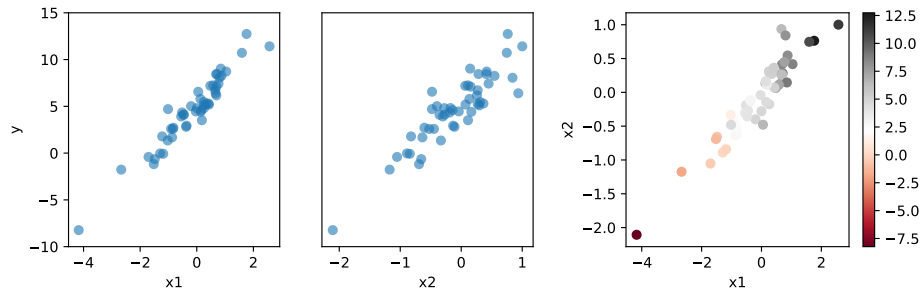
<sup>11</sup>Galit Shmueli. “To Explain or to Predict?” In: *Statistical Science* (2010). URL: <https://doi.org/10.1214/10-STS330>.

<sup>12</sup>Brian Ripley. *Selecting amongst large classes of models*. Lecture for a symposium in honour of John Nelder's 80th birthday, Imperial College. Mar. 2004. URL: <http://www.stats.ox.ac.uk/~ripley/Nelder80.pdf>.

or

$$\text{Model B : } Y_i \sim \text{Normal}(\alpha + \beta x_{1,i}, \sigma^2)$$

Which of these should you choose?



A logician might say “Model B is a special case of Model A, so Model A is obviously better.” A scientist might say “Use Model B unless the data is clearly in favour of A.” An engineer might say “Try them both and use whichever makes better predictions.”

We’ll work through three approaches, hypothesis testing (scientist), cross validation (engineer), and Bayesian (Zen guru). All of them use same basic ingredient—the likelihood function—but they each give it a different spin.

It’s worth noting that the computer code for each approach is short, just a few lines of data science plus some lines of bookkeeping. It’s a hallmark of good data science style to have lots of deep thinking and explanation, then concise code.

<sup>13</sup>Available at [https://teachingfiles.blob.core.windows.net/founds/model\\_selection\\_sample.csv](https://teachingfiles.blob.core.windows.net/founds/model_selection_sample.csv). I created this dataset from model A, with parameters  $Y \sim \text{Normal}(5 + 3x_1 + 0.1x_2, 1)$ .

## 3.3.1. HYPOTHESIS TESTING AND P-VALUES

In science and policy making, it is often useful to frame questions the following way. “My default model is  $H_0$ , which I’ll stick with unless the evidence says otherwise. I’m planning a data-gathering exercise, and based on the data  $Y$  that I gather I might stick with  $H_0$  or I might reject it.” *Hypothesis* is another word for model, and *null hypothesis* means default model. In programming terms, the data scientist has to define a model-testing output procedure `testH0(y)` which prints either “reject  $H_0$ ” or “don’t reject  $H_0$ ”. This is how a frequentist sees the world—it’s all about defining output procedures, then finding their error probability. Here is a common procedure for implementing `testH0`.

1. Define a function `test_statistic(y)`. It can be whatever function of the data we like. Just make sure it’s a function *only* of the data—don’t let it use any unknown parameters, because they’re unknown! We aim for a function that is likely to be small if the null hypothesis is true, and large if it’s false.
2. Assuming  $H_0$  is true<sup>14</sup>, find the distribution of  $T = \text{test\_statistic}(Y)$ . A good way to do this is with resampling.
3. From the actual data we saw, compute  $t = \text{test\_statistic}(y)$ . Mark  $t$  on the histogram of  $T$ , and measure  $p = \mathbb{P}(T \geq t)$ .
4. If  $p \leq 5\%$ , print “reject  $H_0$ ” otherwise print “don’t reject  $H_0$ ”. (The magic number 5% was given as an illustration<sup>15</sup> in 1925, and it stuck ever since.)

In this procedure,  $p$  is called the *p-value* or *significance level*, and it measures the probability of seeing results as extreme as we actually saw, assuming that the null hypothesis is true. With some simple but careful thinking, we can see that this procedure’s error probability is  $\leq 5\%$ , i.e. in less than one case in twenty it will report “reject  $H_0$ ” when  $H_0$  is actually true.

*In the dataset for example 3.9, perhaps  $x_2$  is the dosage of a drug and  $Y$  is the patient response. A reasonable default is “the drug is ineffective”—we don’t want to start prescribing it, with cost and side effects, unless the data tells us it is effective. So we’ll take model B to be the null hypothesis.*

*Step 1 is to invent a test statistic. Intuitively, assuming model B is true, then if we try fitting model A we’ll likely see a small value for the maximum likelihood estimator  $\hat{\beta}_2$ . On the other hand, if model B is false, we’d likely see a larger value. So let’s use this as a test statistic.*

```
1 def test_statistic(df):
2     _, _, beta2, _ = mle_A(df.y, df.x1, df.x2) # max.lik estimators from model A
3     return numpy.abs(beta2)
```

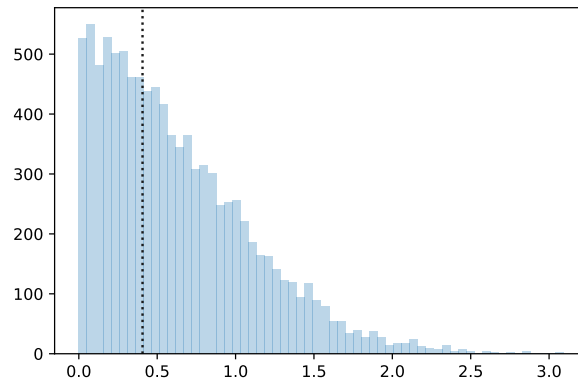
*For step 2 we’ll use parametric resampling, with parameters obtained from the assumption that  $H_0$  is true.*

```
1 alpha, beta, sigma = mle_B(shmueli.y, shmueli.x1, shmueli.x2)
2 def parametric_resample():
3     i = numpy.random.randint(low=0, high=len(shmueli), size=len(shmueli))
4     res = shmueli.loc[i, ['x1', 'x2']]
5     res['y'] = numpy.random.normal(loc=alpha + beta*res.x1, scale=sigma)
6     return res
7
8 Tstar = [test_statistic(parametric_resample()) for _ in range(10000)]
9 t = test_statistic(shmueli) # 0.406
10 p = sum(Tstar >= t)/len(Tstar) # 0.602
11
12 plt.hist(Tstar, bins=60, alpha=.3)
13 plt.axvline(x=test_statistic(shmueli), linestyle='dotted', color='.3')
```

see section 3.2.2 for  
parametric resampling  
`randint(low,high,size)`  
returns a list of size  
elements taken randomly,  
with replacement, from  
{low, ..., high - 1}

<sup>14</sup>If  $H_0$  includes unknown parameters, then the distribution of  $T$  depends on unknown parameters, so we can’t actually find its distribution. The technically correct thing to do is to define  $p = \max_{\theta \in H_0} \mathbb{P}(T \geq t | \theta)$ , i.e. take the worst case over all possible parameters in the model, as we did in section 3.2.1. Resampling ignores this issue.

<sup>15</sup>R. A. Fisher. *Statistical methods for research workers*. 1925.



The  $p$ -value is 60.2%, so we print "don't reject  $H_0$ ". The histogram of  $T_{star}$  shows us that the test statistic is entirely in line with what we'd expect to see if the null hypothesis (model B) is true.

**The alternative hypothesis.** It's bizarre that there's nothing in the frequentist testing procedure that depends on the alternative hypothesis. If someone asks you "Pick one of these hypotheses: either a glass of wine a day is good for you, or the moon is made of green cheese" surely you'd want very compelling evidence for rejecting the wine hypothesis! It's because of qualms about this that we make the test procedure print out "reject  $H_0$ " rather than "accept the alternative".

There is one exception. If you're trying to decide between two models and you don't have any idea about what test\_statistic should be, a good choice is

$$\text{test\_statistic}(Y) = \frac{\text{lik}(\text{model} = A \mid Y = y)}{\text{lik}(\text{model} = B \mid Y = y)}$$

where the likelihood of a model is found by maximizing over all parameters in the model,

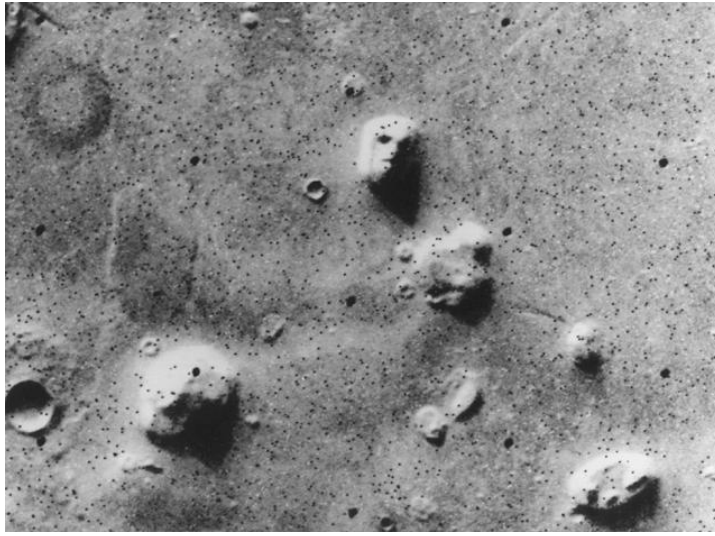
$$\text{lik}(\text{model} = m \mid Y = y) = \max_{\theta} \text{lik}(\theta \mid Y = y, \text{model} = m).$$

If model B is false then the denominator is likely to be small so the test statistic will be large. The model-testing procedure using this statistic is called the *likelihood ratio test*.

**Multiple testing.** If your testing procedure is "Keep inventing different tests until you find one that prints "reject  $H_0$ " then report only this test", then your error probability is 100% and you will be an academic disgrace once you're found out<sup>16</sup>—remember that the  $p < 0.05$  criterion has a one in twenty error probability. The inimitable Randall Munroe puts it best (page 46).

<sup>16</sup>Beth Mole. *Big nutrition research scandal sees 6 more retractions, purging popular diet tips*. Ed. by arstechnica.com. [Online; posted 20-September-2018]. Sept. 2018. URL: <https://arstechnica.com/science/2018/09/six-new-retractions-for-now-disgraced-researcher-purges-common-diet-tips/>.

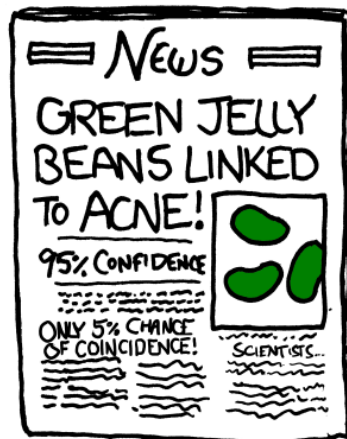
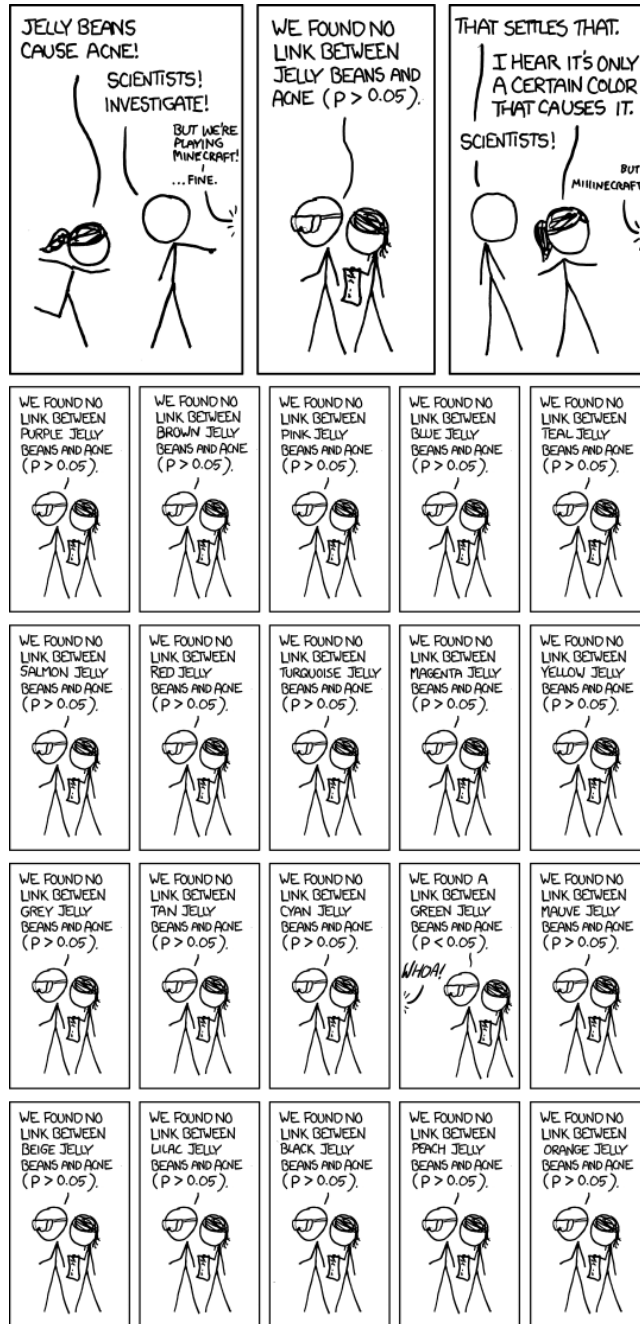
**Exercise 3.10.** The Viking 1 Orbiter spacecraft took this photograph<sup>17</sup> of the surface of Mars on 25 July 1976. NASA scientists attribute the appearance of a face to purely natural processes. Others say that the probability of producing such a face-like artefact purely by chance is infinitesimal, like monkeys at typewriters producing Shakespeare, and that we should therefore reject the null hypothesis “There is no intelligent life on Mars”. Is this correct frequentist reasoning?



<sup>17</sup>NASA/JPL. Catalog number PIA01141, <https://photojournal.jpl.nasa.gov/catalog/pia01141>



xkcd by Randall Munroe, <https://xkcd.com/882/>





## 3.3.2. CROSS VALIDATION AND PERPLEXITY \*

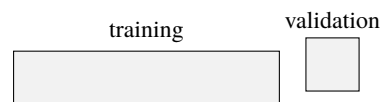
In engineering we just want to build systems that will work well when they're deployed, so we want models that work well on new data. Suppose we've fitted the two models A and B (estimated their parameters, e.g. using maximum likelihood estimation), and now we want to decide which of the two models to deploy. A reasonable rule is to pick whichever model has the higher log likelihood when applied to new data,

$$\text{score} = \mathbb{E} \log \text{lik}(\text{fitted model} \mid \text{new obs} = Y') = \mathbb{E} \log \text{Pr}_Y(Y' \mid \text{fitted model})$$

where  $Y'$  is a hypothetical new observation. (This is frequentism, averaging over a multiverse of what might be.) If we have a way to obtain a sample  $y'_1, \dots, y'_m$  of new data, we can approximate the score using Monte Carlo integration,

$$\text{score} \approx \frac{1}{m} \sum_{i=1}^m \log \text{lik}(\text{model} \mid \text{new obs} = y'_i).$$

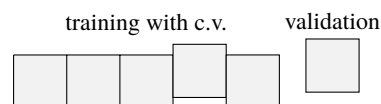
If all we have is the dataset we're given, we can split it into two pieces, a *training subset* and a *validation subset*, then use the former to fit parameters and the latter to evaluate the score. This is called *holdout cross-validation*.



A better approach is to split the dataset into  $K$  parts. For each  $k \in \{1, \dots, K\}$  take out part  $k$  and fit the model on the rest of the data, then compute the score on part  $k$ ; and report the average score across all  $k$ . This smooths out problems from unlucky choice of validation subset. It is called  *$K$ -fold cross-validation*.



Sometimes the model has *hyperparameters*, i.e. parameters that can't be estimated within the model itself, such as the prior distribution in a Bayesian model. Training now involves both fitting parameters and choosing hyperparameters—and choosing a hyperparameter is the same type of job as selecting a model, so we can do it with cross validation. Remember to use a separate holdout validation subset, to evaluate the score of the overall training procedure.



Here is a cross validation comparison of the two models in exercise 3.9, using 5-fold<sup>18</sup> cross validation. We'll fit the models using straightforward maximum likelihood estimation, so there are no hyperparameters to worry about.

```
1 def loglik( $\alpha$ ,  $\beta_1$ ,  $\beta_2$ ,  $\sigma$ , df):
2     # same function works for both models; just set  $\beta_2 = 0$  for model B
3     n = len(df) # df is a dataframe of validation records
4     ey =  $\alpha + \beta_1 \cdot \text{df.x1} + \beta_2 \cdot \text{df.x2}$ 
5     return -n * numpy.log( $\sigma$ ) - sum((df.y - ey)**2) / (2 *  $\sigma$ **2)
6
7 parts = sklearn.model_selection.KFold(n_splits=5)
8 scoresA, scoresB = [], []
9 for itrain, itest in parts.split(numpy.arange(len(shmueli))):
```

<sup>18</sup>Why 5? You often see recommendations for  $K = 5$  or  $K = 10$ . The choice is a compromise between having enough data in the training set to get good parameter estimates (for which we want  $K$  large) and having enough data in the validation set to get a good estimate of the score (for which we want  $K$  small), all of this constrained by the overall size of the dataset. See Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. 2nd. Springer, 2009. URL: <https://web.stanford.edu/~hastie/Papers/ESLII.pdf> section 7.10 for a discussion.

```

10 # itrain and itest are integer vectors — indexes into the dataset
11 (α, β1, β2, σ) = mle_A(shmueli.iloc[itrain])
12 scoresA.append(lik(α,β1,β2,σ, shmueli.iloc[itest]))
13 (α, β, σ) = mle_B(shmueli.iloc[itrain])
14 scoresB.append(lik(α,β,0,σ, shmueli.iloc[itest]))
15 numpy.mean(scoresA), numpy.mean(scoresB) # (-5.9940, -5.7491)
16
17 # We see model B is has higher score. Fit it to the entire dataset, then deploy!
18 (α, β, σ) = mle_B(shmueli)

```

**Perplexity and other scores.** The quantity  $e^{-\text{score}}$  is called *perplexity*, by machine learning researchers in the field of natural language processing. The typical setting is that we want a random variable for generating words: it should make the likelihood high for words that the data shows are common, and low for words that are uncommon. (It's more interesting to measure perplexity for sentences than for independent words, but this has to wait until after we've studied Markov chains in section 6.)

Why does the score function use  $\log \text{lik}$  rather than just  $\text{lik}$ ? In natural language processing, this is just a convention—supported by suggestive links to information theory and data compression. In regression modelling, where the data consists of predictor variables and response variables, many machine learning systems don't even use likelihood at all. Other choices are

see section 1.6 for the terminology: regression, predictor, and response

$$\begin{array}{ll} \text{for numerical responses :} & \text{score} = \mathbb{E}(Y' - \text{predicted value}(X'))^2 \\ \text{for classification problems :} & \text{score} = 1_{Y' = \text{most likely label}(X')} \end{array}$$

where  $Y'$  and  $X'$  are the response variable and predictor variables for a new observation. If there is a natural loss function to use in your application, use it! In these notes I have emphasized the use of likelihood because it ties together regression models and perplexity scores, and because it is a unifying idea behind all of inference.

\* \* \*

**Data hygiene rules.** We want models that work well on new data, so we defined the score function as an expectation over a hypothetical new observation, and approximated it using a validation subset. The golden rule is

*The validation set should reflect the type of new data we want our model to do well on.*

Here is elaboration of the golden rule.

- It's vital to keep the training subset and the validation subset separate. Otherwise the fitting procedure can sneakily learn characteristics of the validation subset, and the score we compute won't tell us what to expect for genuinely new data.
- Pay attention to the *unit of prediction*. For example, suppose we're trying to predict the trips that a person will make, and we have a dataset of trips. Do we want to predict "new trips by the current userbase"? If so, put some trips by every user in both the training set and the validation set. Or do we want to predict "new trips by new users"? If so, make sure that a given person's trips are either all in the training subset or all in the validation subset—otherwise the fitting procedure might sneakily learn to identify a person based on side characteristics, and use this to deduce likely trips.
- If we want our model to do well uniformly across population subgroups, make sure those subgroups are evenly represented in the validation data. For example, here's a simple model for predicting someone's sexuality from their Facebook likes: "always say they are straight". This model has roughly 90% accuracy on the overall population! If we want a model that worked well on people of all sexualities, we should represent them equally in the validation set. This is called *stratification*. (A scientist might say: if we built a proper model in the first place, it would automatically generalize well to new population mixes, and this sort of stratification is a poor substitute.)
- Watch out for trends over time (called *secular trends*). Cross validation uses each part of the dataset as a stand-in for a hypothetical new observation; but in time-ordered datasets maybe we shouldn't use a slice of the past as a drop-in replacement for the future. This is also an issue when you combine datasets from different sources.

## 3.3.3. BAYESIAN MODEL WEIGHTING \*

For a true Bayesian, “which model is correct?” is just another uncertainty, and it should be treated exactly the same as any other uncertain parameter.

Let  $M$  be a discrete random variable taking values in  $\{A, B\}$  indicating which model is correct, with prior distribution  $\Pr_M(m)$ . Write  $\theta$  for vector of unknown parameters (concatenating the parameters from A and B), and assume we have set down prior distributions for them. Applying Bayesian update, the posterior distribution on all the parameters is

$$\Pr(m, \theta | Y = y) \propto \Pr_M(m) \Pr_\Theta(\theta) \Pr_Y(y | \theta, \text{model}=m)$$

where  $y$  denotes the entire column in the dataset. Perhaps all we care about is the posterior probability of each of the models, in which case we’ll treat  $\theta$  as a nuisance variable and integrate it out:

$$\begin{aligned} \Pr_M(m | Y = y) &\propto \Pr_M(m) \int_{\theta} \Pr_\Theta(\theta) \Pr_Y(y | \theta, \text{model}=m) d\theta \\ &= \Pr_M(m) \text{ev}(m | Y = y). \end{aligned}$$

where  $\text{ev}(m | Y = y)$  is defined by this integral, and referred to as the *evidence* for model  $m$ . In words, we start with a prior belief  $\Pr_M(m)$  about which model is true, and we update that belief in the light of the data, according to the evidence for each model.

The true Bayesian would never say “Use Model A rather than Model B”, they would only say “In the light of the data, and given prior beliefs, here are the updated weights to use for each of the models.” Their predictions about new observations would be based on posterior predictive probability, averaging over all unknown parameters including  $M$ . This is called *Bayesian model averaging*.

see page 35 for Bayesian posterior predictive probability

To compute evidence in example 3.9, we’ll start by inventing out of thin air the prior belief that  $1/\sigma^2 \sim \Gamma(0.1, 0.1)$  and that all other parameters are  $\text{Normal}(0, 5^2)$ . Engineers like to use  $1/\sigma^2$  rather than  $\sigma$  or  $\sigma^2$ ; they call it the ‘precision’. Let  $\theta$  be a tuple consisting of all these parameters. The evidence is an integral over  $\theta$ . Let’s write it as an expectation, and then use Monte Carlo integration to approximate it. The evidence is

$$\text{ev}(m | Y = y) = \int_{\theta} \Pr_\Theta(\theta) h_m(\theta) = \mathbb{E} h_m(\Theta)$$

where  $h_m$  is the probability density for the model,

$$h_A(\theta) = \Pr_Y(y | \theta, \text{model}=A) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y_i - e_i)^2 / 2\sigma^2}, \quad \text{where } e_i = \alpha + \beta_1 x_{1,i} + \beta_2 x_{2,i}$$

and similarly for model B.

```

1 # Generate random values from the prior distribution
2 N = 100000
3 αs = numpy.random.normal(loc=0, scale=5, size=N)
4 β1s = numpy.random.normal(loc=0, scale=5, size=N)
5 β2s = numpy.random.normal(loc=0, scale=5, size=N)
6 σs = 1/numpy.sqrt(numpy.random.gamma(shape=0.1, scale=1/0.1, size=N))
7
8 # Compute P_Y for each sampled parameters, then average to get the evidence
9 def P(α, β1, β2, σ):
10     n = len(shmueli)
11     e = α + β1*shmueli.x1 + β2*shmueli.x2
12     ll = -n*numpy.log(σ) - sum((y-e)**2)/(2*(σ**2))
13     return numpy.exp(ll)
14 pA = numpy.array([P(α, β1, β2, σ) for α,β1,β2,σ in zip(αs,β1s,β2s,σs)])
15 pB = numpy.array([P(α, β, 0, σ) for α,β,σ in zip(αs,β1s,σs)])
16 evA, evB = numpy.mean(pA), numpy.mean(pB)
17
18 # It's only evidence ratios that matter, so we might as well normalize:
19 evA, evB = evA/(evA+evB), evB/(evA+evB)
20 evA, evB # (0.00503, 0.99496)

```

Model selection includes choosing which prior to use. (Where did the 5 and 0.1 come from in the priors used in the example above?) Bayesian methods can't help: strict Bayesianism insists that we invent a prior before we even look at the data. We can weasel out by declaring the parameters of the prior distribution to be *hyperparameters*, which is a fancy way of saying “parameter that I have no prior for”, and use non-Bayesian model selection such as cross-validation to pick values for them.

Brian Ripley, an eminent data scientist, says “I think Bayesians are rarely Bayesian in their model choices”<sup>19</sup>.

cross-validation: see  
page 47

---

<sup>19</sup>Brian Ripley. *Selecting amongst large classes of models*. Lecture for a symposium in honour of John Nelder's 80th birthday, Imperial College, Mar. 2004. URL: <http://www.stats.ox.ac.uk/~ripley/Nelder80.pdf>.

## 4. Crafting a model

### 4.1. Quantifying a question

*tl;dr.* The unknown parameters in a probabilistic model have two roles. First, they make the model expressive: we should put in enough parameters so that, by tuning them, we can make the model produce the full range of output patterns that we think are plausible. Second, when we fit the model and inspect the estimated parameters, we learn which of those patterns are actually present in the dataset.

When we invent a model, how should we choose its parameters? Here are two important design guidelines.

- Use parameters that correspond to the questions we want to ask and the quantities we want to measure. Be mindful of whether it's even possible to measure these quantities from our data. This comes under the heading of *identifiability*.
- Usually we'll run maximum likelihood estimation, using a numerical optimization library. Choose a parameterization that makes it easy for the library routines to succeed. This comes under the heading of *natural parameters*.

A model with well-crafted parameters is like a beautiful science equation, easier to understand and more subtle than endless tabulations. We also have tools, both Bayesian and frequentist, for reasoning about how confident we should be in our parameter estimates. Model parameterization is one of the best tools we have for asking questions about a dataset.

Don't think of a model as a claim about what's *true*—any interesting dataset almost certainly has so much richness that any simple parametric model we invent is wrong—but a wrong model can still be useful. There are nevertheless *bad* models! If we pick useless parameters, we'll get useless answers.

Bayesian confidence intervals page 34.  
Frequentist confidence intervals page 37.

#### Example 4.1.

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

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

Is there a racial bias in police decisions to stop-and-search? If so, is it direct racial bias, or might it be indirect bias induced by gender bias?

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

If the police decision to stop-and-search someone is based purely on signs of criminality, then the

probability of finding criminality should be equal in different ethnic groups. But %find is lower for Asian suspects, i.e. the police are stopping relatively more non-criminal Asian suspects, which suggests bias.

But it's 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? The gender breakdown over all police stop-and-searches is different in different ethnic groups:

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

What if there are cultural differences that lead to different %Male on the streets; and what if police are relatively more likely to stop Male suspects than Female, regardless of ethnicity and chance of finding something? Then those ethnicities with higher %Male on the streets would end up with lower %find. In other words, might the lower %find among Asian suspects be indirect racial bias attributable to direct gender bias, rather than direct racial bias?

```

1  if os.path.exists('stop-and-search.csv'):
2      print("file already downloaded")
3  else:
4      !wget "https://teachingfiles.blob.core.windows.net/founds/stop-and-search.csv"
5      police = pandas.read_csv('stop-and-search.csv')
6
7      # Count number of stop records by ethnicity
8      police.groupby('Officer-defined ethnicity').apply(len)
9
10     # Define outcome, and cross-tabulate ethnicity vs outcome
11     police['outcome'] = numpy.where(police['Outcome']=='Nothing found – no further action', \
12                                     'nothing', 'find')
13     x = police.groupby(['Officer-defined ethnicity', 'outcome']).apply(len)
14     x.unstack()
15
16     # Convert to percentages
17     (x / x.sum(level='Officer-defined ethnicity') * 100).unstack()
18
19     # Cross-tabulate gender vs ethnicity, among all stop-and-search records
20     x = police.groupby(['Officer-defined ethnicity', 'Gender']).apply(len)
21     (x / x.sum(level='Officer-defined ethnicity')*100).unstack(fill_value=0)
22
23     # %find, broken down by ethnicity and gender
24     df = police.groupby(['Officer-defined ethnicity', 'Gender', 'outcome']) \
25           .apply(len).unstack(fill_value=0)
26     x = df.find / (df.find + df.nothing)
27     (x * 100).unstack()

```

## 4.1.1. REVIEW OF MAXIMUM LIKELIHOOD ESTIMATION

Before we bring in gender, let's make sure we can solve maximum likelihood estimation for the simplest possible model of ethnic bias,


$$\mathbb{P}(Y_i = \text{find}) = \beta_{e_i} \quad (17)$$

where  $e_i$  is the ethnicity covariate in row  $i$  of the dataset, and  $\beta$  is a vector of probabilities, one per ethnic group. To estimate  $\beta$ , write out the log likelihood

$$\log \text{lik}(\beta | y) = \sum_i \begin{cases} \log \beta_{e_i} & \text{if } y_i = \text{find} \\ \log(1 - \beta_{e_i}) & \text{if } y_i = \text{nothing} \end{cases}$$

and maximize it. Such a simple model is easy enough to solve with algebra and calculus, but it's more useful to be able to solve it by computer. Reassuringly it matches the table above.

```

1  # Some entries have ethnicity numpyp.nan, meaning the value is missing in the dataset.
2  # We'll exclude these records, otherwise the definition of e would fail.
3  ok = ~ pandas.isnull(police['Officer-defined ethnicity'])
4
5  # Prepare vectors of y (boolean) and e (integer index)
6  y = police.loc[ok, 'outcome'] == 'find'
7  ethnicity_levels = ['Asian', 'Black', 'Mixed', 'Other', 'White']
8  ethnicity_code = {k:i for i,k in enumerate(ethnicity_levels)}
9  e = numpy.array([ethnicity_code[v] for v in police.loc[ok, 'Officer-defined ethnicity'])
10
11 def loglik(beta, y, e):
12     xi = beta[e] # get a vector [beta_{e1}, beta_{e2}, ...]
13     return numpy.sum(numpy.log(numpy.where(y, xi, 1-xi)))
14
15 initial_guess = numpy.array([0.5, 0.5, 0.5, 0.5, 0.5])
16 mle = scipy.optimize.fmin(lambda beta: -loglik(beta,y,e), initial_guess)
17  RuntimeWarning: invalid value encountered in log
18
19 pandas.Series(mle, index=ethnicity_levels)

```

Asian	Black	Mixed	Other	White
0.300	0.318	0.606	0.331	0.327

The warning is a sign of a real problem, which we'll address in section 4.1.3.

## 4.1.2. EXPRESSIVITY, IDENTIFIABILITY, AND CONTRASTS

To make the model more expressive, let's make it take account of both ethnicity and gender simultaneously, by

$$\mathbb{P}(Y_i = \text{find}) = \beta_{e_i} + \gamma_{g_i} \quad (18)$$

where  $e_i$  is the ethnicity covariate in row  $i$  of the dataset and  $g_i$  is the gender covariate. This model can accommodate multiple explanations for what's going on in the data, via different parameter choices:

- 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  $e$ .
- If there is no gender bias, and the difference in  $\mathbb{P}(\text{find})$  is down to racial bias, then the model can accommodate this via  $\gamma_g = \text{const}$  for all  $g$ , and we can read off the racial bias in  $\beta$ .

```

1 ok = ...          # only keep rows where both ethnicity and gender are available
2 y, e, g = ...     # similar to before
3 def loglik(theta, y, e, g):
4     beta, gamma = theta[:5], theta[5:]
5     xi = beta[e] + gamma[g]
6     return numpy.sum(numpy.log(numpy.where(y, xi, 1-xi)))
7 initial_guess = numpy.array([0.5, 0.5, 0.5, 0.5, 0.5, 0.2, -0.2, 0])
8 mle = scipy.optimize.fmin(lambda theta: -loglik(theta, y, e, g), initial_guess)
9 ⚠ Warning: Maximum number of function evaluations has been exceeded.
10 beta, gamma = mle[:5], mle[5:]
11
12 # Print out a table of  $\mathbb{P}(\text{find} | e, g)$  by adding  $\beta_e$  and  $\gamma_g$ 
13 x = beta[numpy.newaxis, :] + gamma[:, numpy.newaxis]
14 pandas.DataFrame(numpy.round(x, 2), index=gender_levels, columns=ethnicity_levels)
```

	Asian	Black	Mixed	Other	White
Female	0.301	0.318	0.606	0.332	0.329
Male	0.303	0.320	0.608	0.334	0.330
Other	0.225	0.242	0.530	0.257	0.253

You might or might not see the warning, depending on how the optimizer runs. Or you might find that the answer you get is sensitive to `initial_guess`. Or you might find, when you try to compute a confidence interval for one of the parameters, that it is surprisingly wide. These are all symptoms of *non-identifiability* of the parameters. (You might also see the warning from section 4.1.1, but that's a separate issue.)

The problem is that if we have one maximum likelihood solution  $(\beta, \gamma)$ , then any other solution  $\tilde{\beta} = \beta + \delta$ ,  $\tilde{\gamma} = \gamma - \delta$  will give exactly the same fitted model,  $\mathbb{P}(\text{find}) = \beta + \gamma = \tilde{\beta} + \tilde{\gamma}$ . Is non-identifiability a problem? It certainly is if it stops the optimizer from working! Some optimizers get confused when there's a continuum of optima: they are unable to converge on a single answer, so they keep searching, hence the warning message.

But the real issue with non-identifiability is that we have to be careful which questions we ask. We can't ask for example "What is  $\beta_{\text{Asian}}$ ?" since the answer we get depends on  $\delta$ , which is arbitrary. It is nonetheless meaningful to ask e.g. "What is  $\beta_{\text{Asian}} - \beta_{\text{White}}$ ?", since the difference doesn't depend on  $\delta$ . Differences of this sort are called *contrasts*. Non-identifiability also manifests itself in confidence intervals: we'll get a wide confidence interval for  $\beta_{\text{Asian}}$  no matter how much data there is, and a narrower interval for  $\beta_{\text{Asian}} - \beta_{\text{White}}$ .

A common trick is to rewrite the model with 'reduced' parameters that don't suffer from non-identifiability, for example

$$\Pr_Y(\text{find} | e, g) = \alpha' + \beta'_e + \gamma'_g, \quad \text{where we'll require } \beta'_{\text{Asian}} = \gamma'_{\text{Female}} = 0. \quad (19)$$

It doesn't make any difference which reference levels we choose to set to 0; here I chose them alphabetically. When we unwrap this model,

$$\begin{aligned}
 \Pr_Y(\text{find} | e=\text{Asian } g=\text{Female}) &= \alpha' \\
 \Pr_Y(\text{find} | e=\text{Asian } g=\text{Male}) &= \alpha' + \gamma'_{\text{Male}} \\
 \Pr_Y(\text{find} | e=\text{Black } g=\text{Female}) &= \alpha' + \beta'_{\text{Black}} \\
 \Pr_Y(\text{find} | e=\text{Black } g=\text{Male}) &= \alpha' + \beta'_{\text{Black}} + \gamma'_{\text{male}} \\
 &\dots
 \end{aligned}$$



This is exactly as expressive as our original model,  $\Pr_Y(\text{find}) = \beta_e + \gamma_g$ , since we can get exactly the same values for  $\Pr_Y(\text{find})$  by using the reduced parameters

$$\begin{aligned}\alpha' &= \beta_{\text{Asian}} + \gamma_{\text{Female}} \\ \gamma'_{\text{Male}} &= \gamma_{\text{Male}} - \gamma_{\text{Female}} \\ \beta'_{\text{Black}} &= \beta_{\text{Black}} - \beta_{\text{Asian}} \\ \beta'_{\text{Mixed}} &= \beta_{\text{Mixed}} - \beta_{\text{Asian}} \\ &\dots\end{aligned}$$

This rewriting also makes it clear that the parameters of the reduced model tell us about differences between groups, i.e. about contrasts. For example,  $\beta'_{\text{Black}}$  measures the difference in  $\Pr_Y$  between Black and Asian ethnicities.

\* \* \*

We'll learn more about non-identifiability through studying linear algebra in section 5.3–5.4.

#### Exercise 4.2 (Constructs).

A *construct* is a concept constructed in the mind of the data scientist, for example ‘skill level’<sup>20</sup>. This is as opposed to a quantity like location which, even if it isn’t known, could conceivably be measured directly.

Three chess players play each other. In a tournament,  $A$  won 7 matches against  $B$  and lost 3,  $A$  won 9 matches against  $C$  and lost 1, and  $B$  won 6 matches against  $C$  and lost 4. We wish to ascribe a skill level to each player, such that the higher the skill difference the more likely it is that the higher-skilled player wins a match. Let  $\mu_A$ ,  $\mu_B$ , and  $\mu_C$  be skill levels, and consider this model: if match  $i$  is between players  $p1(i)$  and  $p2(i)$  then the probability that  $p1(i)$  wins is  $e^{\xi_i} / (1 + e^{\xi_i})$  where  $\xi_i = \mu_{p1(i)} - \mu_{p2(i)}$ .

- Find the log likelihood of  $(\mu_A, \mu_B, \mu_C)$
- Show that these parameters are not identifiable, and give an equivalent ‘reduced’ parameterization that is identifiable.
- Compute the maximum likelihood estimators numerically.

<sup>20</sup>Microsoft’s Xbox Live uses an invented construct called TrueSkill for ‘skill of a gamer’. It is documented at <https://www.microsoft.com/en-us/research/project/trueskill-ranking-system> and is 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>.

## 4.1.3. NATURAL PARAMETERS

The model on page 53,  $\mathbb{P}(Y_i = \text{find}) = \beta_{e_i}$ , has a problem. When we ran the optimizer it warned us

 **RuntimeWarning: invalid value encountered in log**

and if we insert a `print( $\beta$ )` statement in the `loglik` function, we see that the optimizer is trying  $\beta$  values that are  $< 0$  or  $> 1$ . This isn't surprising, since there's nowhere in the code that we told it to restrict itself to valid probabilities.

When we compute maximum likelihood, we had better make sure that the model makes sense—that it only looks for probabilities in the range  $[0, 1]$ . We can ensure this using what I call *natural parameters*: tweak the model so that it takes arbitrary real-valued parameters and transforms them to legitimate probabilities. Instead of trying to fit a model like (19),

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

let's fit

$$\mathbb{P}(Y_i = \text{find}) = \frac{e^{\xi_i}}{1 + e^{\xi_i}} \quad \text{where } \xi_i = \alpha + \beta_{e_i} + \gamma_{g_i}. \quad (20)$$

This is just an algebraic gimmick that maps any real number  $\xi \in (-\infty, \infty)$  to a value  $e^{\xi}/(1 + e^{\xi})$  in the range  $[0, 1]$ . The two models (19) and (20) are *not* equivalent, i.e. there's no way to rewrite one to obtain the other, as there was for (18) and (19). But what we're really after here is to disentangle the effects of ethnicity and of gender—and our new model lets us do this, so it's a useful.

When we unwrap (20),

$$\begin{aligned} \Pr_Y(\text{find} \mid e=\text{Asian } g=\text{Female}) &= e^{\alpha} / (1 + e^{\alpha}) \\ \Pr_Y(\text{find} \mid e=\text{Asian } g=\text{Male}) &= e^{\alpha+\gamma_{\text{male}}} / (1 + e^{\alpha+\gamma_{\text{male}}}) \\ \Pr_Y(\text{find} \mid e=\text{Black } g=\text{Female}) &= e^{\alpha+\beta_{\text{Black}}} / (1 + e^{\alpha+\beta_{\text{Black}}}) \\ \Pr_Y(\text{find} \mid e=\text{Black } g=\text{Male}) &= e^{\alpha+\beta_{\text{Black}}+\gamma_{\text{Male}}} / (1 + e^{\alpha+\beta_{\text{Black}}+\gamma_{\text{Male}}}) \\ &\dots \end{aligned}$$

The particular algebraic gimmick we used here is known as *logit* or *softmax*, and we saw it before in section 1.6. It lets us use a general-purpose optimization routine, and it won't stray into disallowed parts of the parameter space, because there are none. Generally speaking, optimization routines are happiest with (i) unconstrained problems, i.e. where there are no bounds on the allowable parameter values, (ii) differentiable functions, (iii) functions that have non-zero gradient everywhere except at optima—so don't try a model like  $\Pr_Y(\text{find}) = \max(0, \min(1, \alpha + \beta_e + \gamma_g))$ , which is flat over many parts of the parameter space.

\* \* \*

**Intersectionality.** We often see medical reports like “a Mediterranean diet halves your risk of heart attack”. These usually have a natural-parameter model behind them, for example

$$\mathbb{P}(\text{heart attack}) = \frac{e^{\xi+\mu d}}{1 + e^{\xi+\mu d}}$$

where  $\mu$  is a parameter for the effect of “on Mediterranean diet”, the covariate  $d$  is 1 if you follow that diet and 0 otherwise, and  $\xi$  is made up of parameters relating to other features such as age and gender and weight. Writing it out in more detail,

$$\mathbb{P}(\text{heart attack} \mid \text{no diet}) = \frac{e^{\xi}}{1 + e^{\xi}}, \quad \mathbb{P}(\text{heart attack} \mid \text{Med. diet}) = \frac{e^{\xi+\mu}}{1 + e^{\xi+\mu}}.$$

This sort of study is usually done in populations where the risk of heart attack is fairly small, so the numerators are  $\approx 0$  thus the denominators are  $\approx 1$ , so

$$\mathbb{P}(\text{heart attack} \mid \text{no diet}) \approx e^{\xi}, \quad \mathbb{P}(\text{heart attack} \mid \text{Med. diet}) \approx e^{\mu} e^{\xi}.$$

We deduce from the headline that the study found the maximum likelihood estimator to be  $\mu = \log 1/2$ . The medical report won't say what the risk of heart attack was cut *from* or what it was cut *to*, since those numbers depend on  $\xi$  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”.

What about intersectionality: what if the effect of a Mediterranean diet is different in different populations? What if police have a gender bias but it's different in different ethnic groups? In section 5.2.1 we'll see examples of how parametric models can be used to ask this sort of question.

## 4.1.4. LOGISTIC REGRESSION

Our final model,

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

is called a *logistic regression*. It's logistic because it uses the logit transform for parameters, and it's a regression because it has a response variable ( $Y_i$ ) predicted by covariates ( $e_i, g_i$ ).

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  $\xi$ , 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.

```

1  # Only keep records where both ethnicity and gender are available
2  ok = ~ (pandas.isnull(police['Officer-defined ethnicity']) | pandas.isnull(police['Gender']))
3
4  # Prepare vectors of y (boolean) and e (integer index) and g (integer index)
5  y = police.loc[ok, 'outcome'] == 'find'
6  ethnicity_levels = ['Asian', 'Black', 'Mixed', 'Other', 'White']
7  ethnicity_code = {k:i for i,k in enumerate(ethnicity_levels)}
8  e = numpy.array([ethnicity_code[v] for v in police.loc[ok, 'Officer-defined ethnicity'])]
9  gender_levels = ['Female', 'Male', 'Other']
10 gender_code = {k:i for i,k in enumerate(gender_levels)}
11 g = numpy.array([gender_code[v] for v in police.loc[ok, 'Gender']])
12
13 def unwrap_pars(theta):
14     alpha = theta[0]
15     beta = {'Asian': 0, 'Black': theta[1], 'Mixed': theta[2], 'Other': theta[3], 'White': theta[4]}
16     beta = numpy.array([beta[k] for k in ethnicity_levels])
17     gamma = {'Female': 0, 'Male': theta[5], 'Other': theta[6]}
18     gamma = numpy.array([gamma[k] for k in gender_levels])
19     return (alpha, beta, gamma)
20
21 def loglik(theta, y, e, g):
22     alpha, beta, gamma = unwrap_pars(theta)
23     xi = alpha + beta[e] + gamma[g]
24     # log lik = sum_i (xi_i 1_{y_i} - log(1 + e^{xi_i}))
25     return numpy.sum(xi[y]) - numpy.sum(numpy.log(1 + numpy.exp(xi)))
26
27 initial_guess = numpy.array([0, 0,0,0,0, 0,0])
28 mle = scipy.optimize.fmin(lambda x: -loglik(x,y,e,g), x0=initial_guess, maxiter21=5000)
29
30 # Print out a table of P(find | e, g)
31 alpha, beta, gamma = unwrap_pars(mle)
32 xi = alpha + beta[numpy.newaxis,:] + gamma[:,numpy.newaxis]
33 x = numpy.exp(xi) / (1 + numpy.exp(xi))
34 pandas.DataFrame(numpy.round(x,3), index=gender_levels, columns=ethnicity_levels)

```

	Asian	Black	Mixed	Other	White
Female	0.297	0.315	0.460	0.332	0.326
Male	0.302	0.320	0.465	0.337	0.331
Other	0.239	0.254	0.387	0.269	0.264

<sup>21</sup>I cheated here by setting maxiter=5000. Without it, the code produced

 **Warning: Maximum number of function evaluations has been exceeded.**

We saw this warning before and said it was due to non-identifiability of parameters. This time it arises because of a 'hard-to-identify' parameter, namely  $\beta_{\text{Mixed}}$ . There are so few cases of  $e=\text{Mixed}$  in the dataset that the log likelihood function is fairly flat as  $\beta_{\text{Mixed}}$  varies. See the discussion in section 5.4.1.

It's a fact of machine learning life that model training often requires babysitting, one part of which is helping the machine to realize when it's found a good enough estimate, so that it doesn't hunt around fruitlessly optimizing something that makes little difference.

## 4.2. Accounting for uncertainty \*

Donald Rumsfeld, the former US Secretary of Defense, famously said<sup>22</sup>

*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’. Pragmatically, in data science, there are many sources of uncertainty, and it’s useful to be able to mix them. And Rumsfeld’s unknown unknowns? That’s when you have the wrong models.

### Example 4.3 (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.

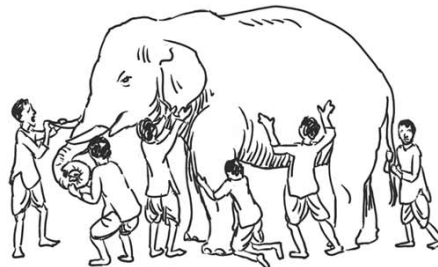
Sometimes there is prior data, e.g. someone has conducted a study of “typical bias in coins used in data science textbook illustrations”. A Bayesian data scientist might translate those observed frequencies directly into a prior distribution.

### Example 4.4 (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  $t_i \in \{\text{active}, \text{placebo}\}$ , and let  $X_{i,j} \sim \text{Normal}(\Theta_i, \sigma^2)$  be independent conditional on  $\Theta_i$ . (The wellness score is a ‘construct’ in the sense of example 4.2 on page 55.) This model allows an individual subjects’s measurements to be clustered tightly together (if  $\sigma$  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,  $\Theta_i$  is a parameter for  $X_{i,j}$ , and we are treating  $\Theta_i$  as a random variable, which is what Bayesians do. But we can at the same time use maximum likelihood estimation and bootstrap resampling for  $\mu$  and  $\rho$  and  $\sigma$ , like a frequentist. This is called *mixed effects modelling*. The  $\Theta_i$  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 machine<sup>23</sup>. 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

<sup>22</sup>U.S. Department of Defense news briefing, 12 February 2002, about the failure to find weapons of mass destruction in Iraq

<sup>23</sup>I.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>.

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. The next example shows the general technique but applied instead to ecology.

**Example 4.5 (Empirical Bayesianism).**

I am catching butterflies. Each butterfly species  $i$  has frequency  $\theta_i$ , 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  $\theta_i$ , and we knew the total number of species  $n$ , then it would be easy to work out the probability of interest:

$$\mathbb{P}\left(\begin{array}{c} \text{next butterfly} \\ \text{is new species} \end{array}\right) = \frac{\sum_{i=1}^n \theta_i 1_{X_i=0}}{\sum_{i=1}^n \theta_i}. \quad (21)$$

But if we don’t know the  $\theta_i$  and we don’t know  $n$ , what can we do?

Let’s adopt a Bayesian approach and treat the  $\theta_i$  as random variables drawn independently from some common distribution, say with density function  $g(\theta)$ , and let  $\Theta$  be a typical value,  $\Pr(\Theta = \theta) = g(\theta)$ , and let  $X \sim \text{Poisson}(\Theta)$  be a typical count. Then the numerator of (21) is

$$\begin{aligned} \mathbb{E}\left(\sum_{i=1}^n \theta_i 1_{X_i=0}\right) &= n \mathbb{E}(\Theta 1_{X=0}) \\ &= n \mathbb{E}\left[\mathbb{E}(\Theta 1_{X=0} \mid \Theta)\right] \quad \text{law of total expectation, (8) page 18} \\ &= n \mathbb{E}(\Theta e^{-\Theta}) = n \int_{\theta=0}^{\infty} \theta e^{-\theta} g(\theta) d\theta. \end{aligned}$$

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 (21) 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 (21) by the total number of samples we’ve seen,  $\sum_i X_i$ . Therefore,

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

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’.

Butterfly counting, and Turing and Good’s work, are examples of *empirical Bayesianism*. Extensions of the 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). 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 Hastie<sup>24</sup>. They say

<sup>24</sup>Bradley Efron and Trevor Hastie. *Computer age statistical inference: algorithms, evidence, and data science*. CUP, 2016.

*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. [...] 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 [...] Empirical Bayes removes the Bayes scaffolding. In place of a reassuring prior, 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.*

## 5. Feature spaces

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.

Section 5.1 starts with a very simple linear model example, to show how to implement them in practice, and to flesh out the uselessly abstract definitions above. In the following sections we'll look at extensions:

- Linear models are expressive, and we can use them to ask all sorts of questions about a dataset by choosing appropriate features. We'll look at examples in section 5.2. Linear models should be your go-to models for all sorts of data science and machine learning problems, the second thing you try (after simple tabulations) to get a sense of the data you're working with.
- A simple way to estimate the parameters is using *least squares estimation*. There are fast algorithms for doing this, which come from the mathematics of linear algebra. The mathematics also gives insight into how linear models work, especially questions of parameter identifiability. Section 5.3 contains a review of the relevant linear algebra.
- There is a probabilistic interpretation of least squares estimation: for a regression model with Normally distributed response variables, least squares estimation is the same as maximum likelihood estimation. This means we can use all the inference tools developed in section 3 to compute confidence intervals, compare models, etc. Section 5.4 describes this link.
- A wider class of probabilistic models, including logistic regression as seen in section 4.1.4, can be seen as *generalized linear models*. Section 5.5 discusses this. Linear models are the building block for many other machine learning techniques, some of which you'll study in Part II *Machine Learning and Bayesian Inference*: support vector machines, perceptrons, and deep neural networks.

## 5.1. Fitting a linear model

tl;dr. A *linear model* can be written as

$$y = \beta_1 e_1 + \cdots + \beta_K e_K + \varepsilon$$

where  $y = (y_1, y_2, \dots)$  is the vector of responses with  $y_i$  the value for record  $i$  in the dataset,  $e_1, \dots, e_K$  are feature vectors with  $e_k = (e_{k,1}, e_{k,2}, \dots)$  where  $e_{k,i}$  is the value of the  $k$ th feature for record  $i$ ,  $\beta_k$  is the parameter that weights the  $k$ th feature, and  $\varepsilon = (\varepsilon_1, \varepsilon_2, \dots)$  is a vector of *residuals*, also called error or noise.

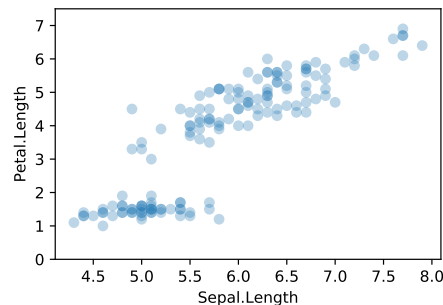
*Least squares estimation* means picking the parameters  $\beta$  to minimize the *mean square error*  $\sum_i \varepsilon_i^2$ . Use `sklearn.linear_model.LinearRegression()` to do this.

### Example 5.1.

The Iris dataset was collected by the botanist Edgar Anderson and popularized<sup>25</sup> 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
...				

Let’s investigate how petal length depends on sepal length. Here is a plot:



It suggests a curve. Let’s fit a quadratic curve, using the linear model

$$\text{Petal.Length} \approx \alpha + \beta \text{Sepal.Length} + \gamma (\text{Sepal.Length})^2. \quad (22)$$

Linear does NOT mean ‘straight line’. It refers to linear algebra—adding vectors, and multiplying vectors by scalars. In vector form, the model says

$$\begin{bmatrix} \text{Petal.Length}_1 \\ \text{Petal.Length}_2 \\ \vdots \end{bmatrix} \approx \alpha \begin{bmatrix} 1 \\ 1 \\ \vdots \end{bmatrix} + \beta \begin{bmatrix} \text{Sepal.Length}_1 \\ \text{Sepal.Length}_2 \\ \vdots \end{bmatrix} + \gamma \begin{bmatrix} (\text{Sepal.Length}_1)^2 \\ (\text{Sepal.Length}_2)^2 \\ \vdots \end{bmatrix}.$$

<sup>25</sup>It’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>



In scientific computing, the coding style is also in terms of vectors:

```
1 one, x, y = numpy.ones(len(iris)), iris['Sepal.Length'], iris['Petal.Length']
2 model = sklearn.linear_model.LinearRegression(fit_intercept=False)
3 # Specify the three feature vectors [one, x, x**2] and the response vector y
4 model.fit(numpy.column_stack([one, x, x**2]), y[:, numpy.newaxis])
5 ((alpha, beta, gamma),) = model.coef_ #unpack a 1 x 3 array of parameters

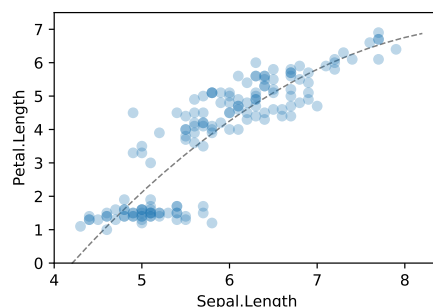
(-17.447, 5.392, -0.296)
```

In fact the sklearn model fitting function always includes a one vector, unless we explicitly tell it otherwise with `fit_intercept=False`. Another way to write this code is

```
6 model2 = sklearn.linear_model.LinearRegression()
7 model2.fit(numpy.column_stack([x, x**2]), y[:, numpy.newaxis])
8 (alpha,), ((beta, gamma),) = model2.intercept_, model2.coef_
```

What does this fit look like? We could explicitly evaluate  $\alpha + \beta x + \gamma x^2$  for a range of  $x$  values and plot. Or use `model.predict()`, to relieve us from re-typing the model formula.

```
9 newx = numpy.linspace(4.2, 8.2, 20)
10 xi = model2.predict(numpy.column_stack([newx, newx**2]))
11
12 fig, ax = plt.subplots()
13 ax.plot(newx, xi, color='0.5', zorder=-1, linewidth=1, linestyle='dashed')
14 ax.scatter(iris['Sepal.Length'], iris['Petal.Length'], alpha=.3)
15 ax.set_ylim(0, 7.5)
16 ax.set_ylabel('Petal.Length')
17 ax.set_xlabel('Sepal.Length')
```



**Terminology.** We'd describe model (22) as having two features, `Sepal.Length`, and  $(\text{Sepal.Length})^2$ . The rows in this dataset have other attributes, and they can be transformed to create an infinite 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 *response* or *label* in 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  $\alpha$ ,  $\beta$ , and  $\gamma$ , 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 one with parameter  $\alpha$ . 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  $\alpha$ ,  $\beta$  and  $\gamma$  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 just easier to work with, so they're a better place to start.

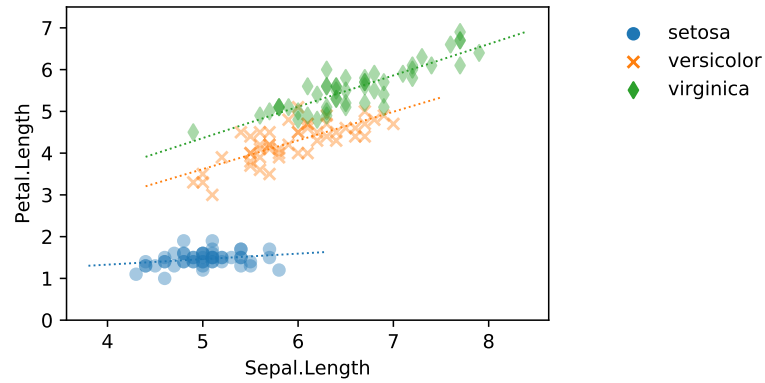
## 5.2. Features

Here is a gallery of cunning ways to use features to ask questions about a dataset.

### 5.2.1. ONE-HOT CODING

One-hot coding is used to turn an enum feature (also called *categorical* or *factor*) into a collection of binary features, so it can be used in a linear model. Here's an example.

The Iris data is made up of three species. Maybe there's a straight-line fit between petal length and sepal length, but with different slopes and intercepts for each species.



One way to write this is

$$\text{Petal.Length} \approx \alpha_{\text{species}} + \beta_{\text{species}} \text{Sepal.Length}.$$

Here's the same equation, but written as a linear model:

$$\begin{aligned} \text{Petal.Length} \approx & \alpha_1 s_1 + \alpha_2 s_2 + \alpha_3 s_3 \\ & + \beta_1 (s_1 \otimes \text{Sepal.Length}) + \beta_2 (s_2 \otimes \text{Sepal.Length}) + \beta_3 (s_3 \otimes \text{Sepal.Length}) \end{aligned}$$

In this equation, each  $s_k$  is a binary vector marking out which rows belong to the  $k$ th species, for example  $s_3 = 1[\text{Species}=\text{virginica}]$ . This is called *one-hot coding* of the Species vector. Also,  $\otimes$  means elementwise multiplication.

$1_x$  also written  $1[x]$  is the indicator function,  $1_{\text{true}} = 1$  and  $1_{\text{false}} = 0$

```
1 species_levels = numpy.unique(iris['Species'])
2 x, y = iris['Sepal.Length'], iris['Petal.Length']
3 s1,s2,s3 = (iris['Species']==s for s in species_levels)
4 model = sklearn.linear_model.LinearRegression(fit_intercept=False)
5 model.fit(numpy.column_stack([s1,s2,s3,s1*x,s2*x,s3*x]), y[:,numpy.newaxis])
```

We've seen one-hot coding before. In section 2.1 we used it to encode the event  $\{X \in A\}$  as a numerical random variable  $1_{X \in A}$ , which allowed us to compute the expectation  $\mathbb{E} 1_{X \in A} = \mathbb{P}(X \in A)$ .

### 5.2.2. PERIODIC AND SECULAR TRENDS

#### Example 5.2.

The UK Met Office makes available historic data<sup>26</sup> 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.

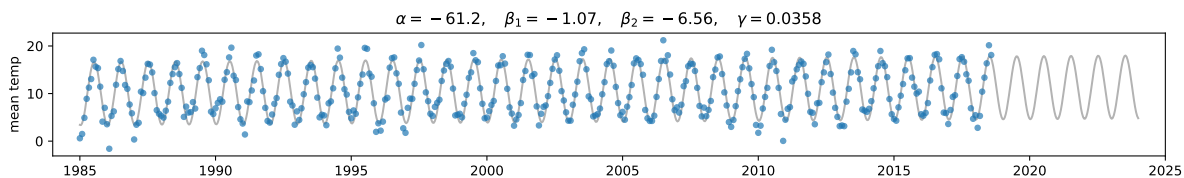
month	tmax	tmin	af	rain	sun	station	lat	lng	alt_m
1963 Sep	14.7	5.9	0	126.4	127.7	Eskdalemuir	55.311	-3.206	242
1955 Aug	—	—	—	35.1	194.7	Shawbury	52.794	-2.663	72
1937 May	15.3	8.4	0	59.8	184.8	Lowestoft	52.483	1.727	18
2007 Aug	20.6	11.8	0	40.3	204.6	Waddington	53.175	-0.522	68
1925 July	21.8	12.6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

Are temperatures increasing? It's tricky to read this directly off a raw data 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

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

where  $t$  is the date in years, and  $\alpha$ ,  $\beta$ ,  $\gamma$ , and  $\theta$  are unknown parameters. Here is the data and the fitted model for Cambridge station (measured at the National Institute of Agricultural Botany, between Churchill and Girton colleges). The plot shows the mean temperature  $\text{temp} = (\text{tmin} + \text{tmax})/2$ .



The model is linear in  $\alpha$  and  $\beta$  and  $\gamma$  and not in  $\theta$ —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

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

```

1 climate = pandas.read_csv('https://teachingfiles.blob.core.windows.net/founds/climate.csv')
2 df = climate.loc[(climate.station=='Cambridge') & (climate.yyyy>=1985)]
3 t = df.yyyy + (df.mm-1)/12
4 temp = (df.tmin + df.tmax)/2
5 model = sklearn.linear_model.LinearRegression()
6 X = numpy.column_stack([numpy.sin(2*numpy.pi*t), numpy.cos(2*numpy.pi*t), t])
7 model.fit(X, temp[:, numpy.newaxis])
8 (alpha,), ((beta1,beta2,gamma),) = (model.intercept_, model.coef_)

```

**Intercepts.** Why is  $\alpha$  so extreme? It is the temperature in the year 1 BC (there was no year 0 AD), based on linearly extrapolating the rate  $\gamma$ . It's daft to trust that the model will predict well for such a wild extrapolation! If we rewrite the model in the equivalent form

$$\text{temp} \approx \alpha + \beta_1 \sin(2\pi t) + \beta_2 \cos(2\pi t) + \gamma(t-2000)$$

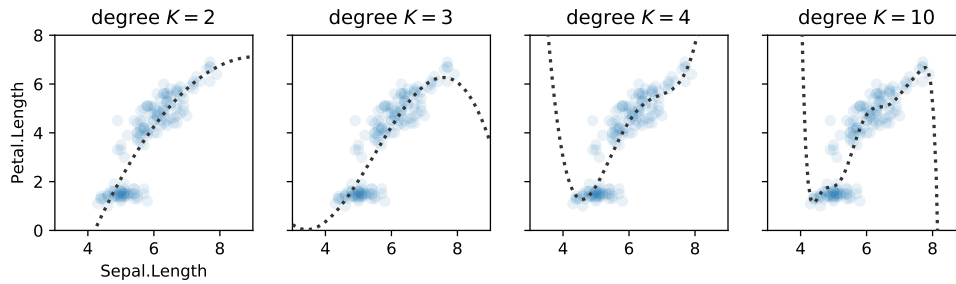
then  $\alpha$  will report the temperature for year 2000.

### 5.2.3. NON-LINEAR RESPONSE

We've already seen that we can use polynomial features to capture smooth curves. Higher degree polynomials have more parameters to estimate, so they're more expressive and can fit the data better, but it's unwise to rely on them especially outside the range where we have data. In the iris dataset from page 5.1,

$$\text{Petal.Length} \approx \alpha + \beta_1 \text{Sepal.Length} + \beta_2 (\text{Sepal.Length})^2 + \dots + \beta_K (\text{Sepal.Length})^K$$

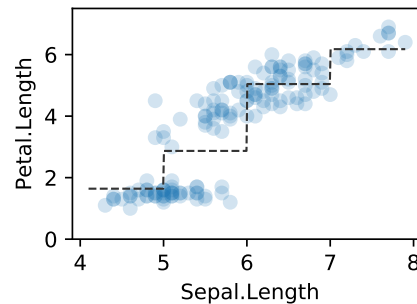
<sup>26</sup><https://www.metoffice.gov.uk/public/weather/climate-historic>



A different approach is to use parameters for anchor points in an arbitrary curve. In this next model the arbitrary curve is a step function with fixed  $x$ -axis breaks, and least squares estimation finds the height at each step.

$\lfloor x \rfloor$  is  $x$  rounded down to the nearest integer

$$\text{Petal.Length} \approx \beta_4 1[\lfloor \text{Sepal.Length} \rfloor == 4] + \dots + \beta_7 1[\lfloor \text{Sepal.Length} \rfloor == 7].$$



This model is more honest because it is upfront about being an arbitrary fit to the data, incapable of extrapolating outside the data range. This example isn't interesting (we could just as well have fitted each integer bin separately), but it's very useful when combined with other features. More guidance on curve fitting on page 69.

**Exercise 5.3.** For the climate data in section 5.2.2 we proposed the model

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

in which the  $+\gamma t$  term asserts that temperature is increasing at a constant rate. To test this, create a non-numerical feature out of  $t$  by

```
1 decade = 'decade_' + str(numpy.floor(t/10).astype(numpy.int)) + '0s'
```

(which has values like 'decade\_1980s', 'decade\_1990s' etc.) and fit the model

$$\text{temp} \approx \alpha + \beta \sin(2\pi t + \theta) + \gamma_{\text{decade}}.$$

Write this as a linear model. How might we use it to investigate whether temperatures are indeed increasing at a constant rate? What are the advantages and disadvantages of this model, as opposed to fitting

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

separately for each decade?

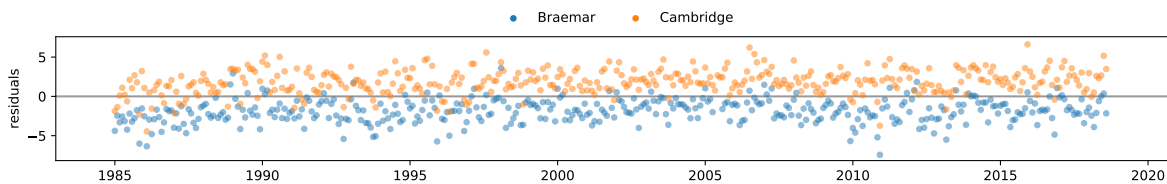
#### 5.2.4. DISCOVERING FEATURES

the residuals are the error term in a linear model

It's often illuminating to plot the residual vector, to find out if we have missed any features worth encoding. Here's an example. Suppose we take the climate dataset for two weather stations, Cambridge and Braemar in the Scottish highlands, and for both stations together we fit

$$\text{temp} = \alpha + \beta \sin(2\pi t + \theta) + \varepsilon$$

where  $\varepsilon$  is the error term. Here is a scatter plot showing  $\varepsilon$  as a function of  $t$  and station. We can see immediately that there's a systematic difference between the two stations (suggesting we should add a  $+\delta_{\text{station}}$  term to the model) and also a hint of a general trend over time (suggesting we do indeed need the  $+\gamma t$  term).



```

1 climate = pandas.read_csv('https://teachingfiles.blob.core.windows.net/founds/climate.csv')
2 ok = (climate.station.isin(['Cambridge', 'Braemar'])) & (climate.yyyy>=1985) \
3       & ~pandas.isnull(climate.tmin+climate.tmax)
4 df = climate.loc[ok].copy()
5
6 df.t = df.yyyy + (df.mm-1)/12
7 df.temp = (df.tmin + df.tmax)/2
8 model = sklearn.linear_model.LinearRegression()
9 X = numpy.column_stack([numpy.sin(2*numpy.pi*df.t), numpy.cos(2*numpy.pi*df.t)])
10 model.fit(X, df.temp)
11 df.pred = model.predict(X) # This would raise SettingWithCopy warning, without copy()
above
12
13 with plt.rc_context({'figure.figsize': (15,1.7)}):
14     for station in numpy.unique(df.station):
15         i = df.station == station
16         plt.scatter(df.t.loc[i], (df.temp - df.pred).loc[i], label=station, s=15, alpha=0.5)
17     plt.axhline(0, color='0.6', zorder=-1)
18     plt.xlim(1983, 2021)
19     leg = plt.legend(ncol=2, frameon=False, bbox_to_anchor=(0.5, 1), loc='lower center')
20     for lh in leg.legendHandles:
21         lh.set_alpha(1)
22     plt.ylabel('residuals')

```

\* \* \*

We design features for several purposes:

- Features to extract a particular summary from the data, e.g. the linear trend in the climate data
- ‘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 the polynomial fits in section 5.2.3). 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.

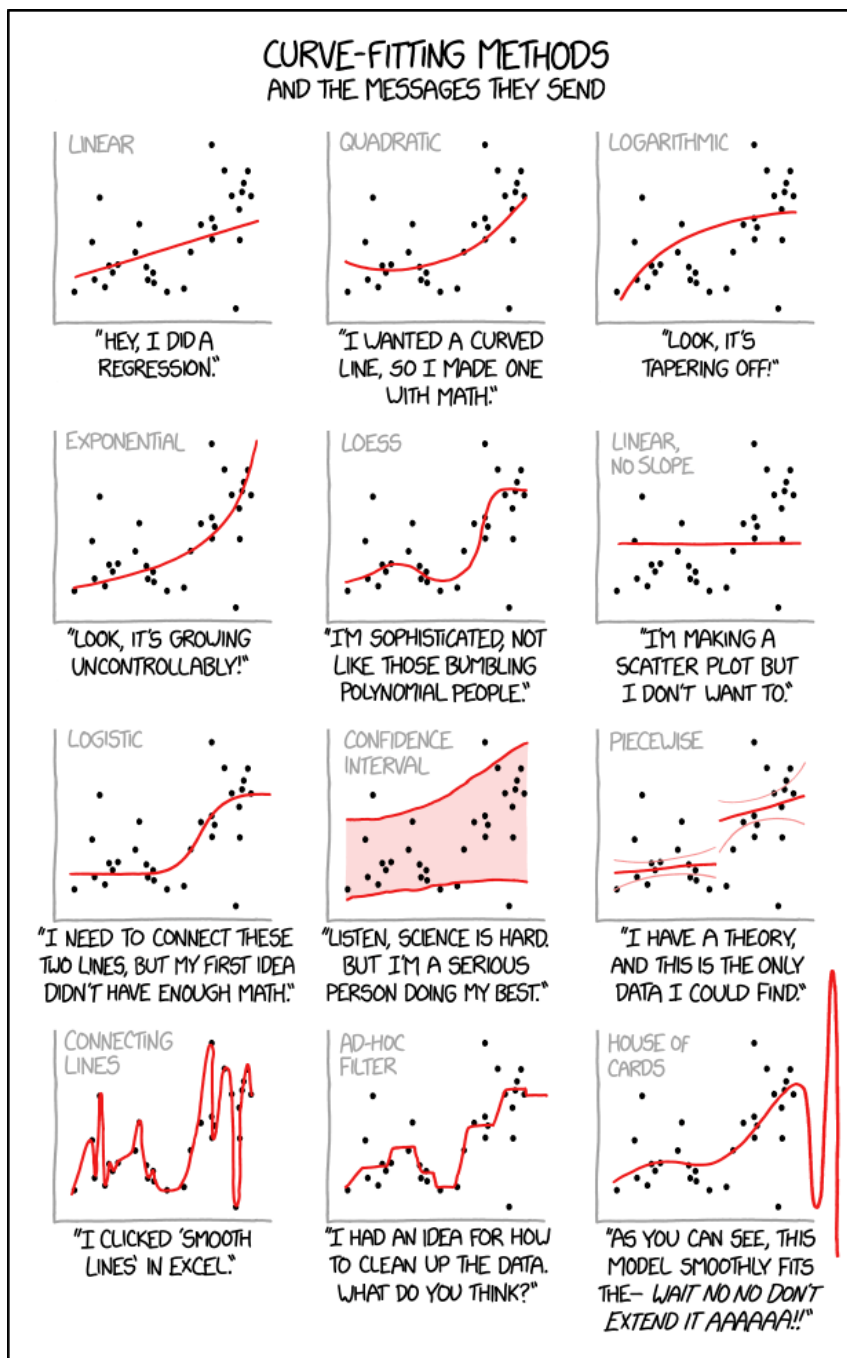
**Dimension reduction.** Start with a long list of possible features  $\{e_1, \dots, e_K\}$ . Construct two new features  $f_1$  and  $f_2$ , each of them a linear combination of the raw features. The goal is to construct them so that

$$Y \approx \alpha + \beta_1 f_1 + \beta_2 f_2$$

has errors that are as small as possible. For this it helps to have geometrical intuition about feature spaces, which needs linear algebra—see section 5.4. This procedure gives us two features  $f_1$  and  $f_2$  that capture as much information as they can about  $Y$ , i.e. it has discovered a *two-dimensional embedding* of the dataset’s  $K$  dimensions. We can show the data on a scatter plot of  $f_1$  against  $f_2$ , and it’s likely to reveal useful clusters. The general term for finding lower-dimensional representations of data is *dimension reduction*.

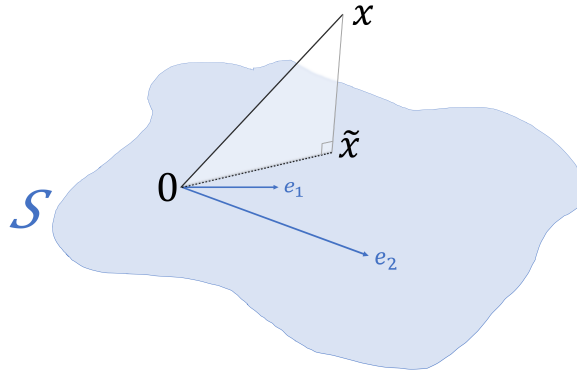
**Feature selection.** Start with a long list of possible features. Pick  $m$ , a number of features to use, and find the best fitting model subject to the constraint that it's only allowed to use  $m$  of the possible features. This is called *feature selection*.

xkcd by Randall Munroe, <https://xkcd.com/2048/>



### 5.3. Linear mathematics

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



- Linearly independent basis vectors  $e_1$  and  $e_2$
- The set of linear combinations of those vectors, also known as the subspace spanned by those vectors, is  $S = \{\lambda_1 e_1 + \lambda_2 e_2 : -\infty < \lambda_1, \lambda_2 < \infty\}$
- 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

For linear models in data science all we need is vectors in simple Euclidean space,  $\mathbb{R}^n$  where  $n$  is the number of records in the dataset. A linear model like the Iris model on page 62 is a vector equation

$$\begin{bmatrix} \text{Petal.Length}_1 \\ \text{Petal.Length}_2 \\ \vdots \end{bmatrix} \approx \alpha \begin{bmatrix} 1 \\ 1 \\ \vdots \end{bmatrix} + \beta \begin{bmatrix} \text{Sepal.Length}_1 \\ \text{Sepal.Length}_2 \\ \vdots \end{bmatrix} + \gamma \begin{bmatrix} (\text{Sepal.Length}_1)^2 \\ (\text{Sepal.Length}_2)^2 \\ \vdots \end{bmatrix}.$$

But it's good for the soul to define linear algebra abstractly and mathematically 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, where vectors represent functions) and *Quantum Computing* (where vectors represent quantum states).

- Let  $V$  be a set whose elements are called *vectors*, denoted by Roman letters<sup>27</sup>  $u, v, w$ , etc.
- Let  $F$  be a field whose elements are called *scalars*, denoted by Greek letters  $\lambda, \mu$ , 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  $\lambda v$ .
- Let there be a binary operation  $V \times V \rightarrow F$ , called *inner product*, written  $v \cdot w$ .

<sup>27</sup>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  $0$  is a vector and  $1$  is a scalar.



## 5.3.1. DEFINITIONS AND USEFUL PROPERTIES \*

**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  $\lambda$  and vectors  $v, w$ .
6.  $(\lambda + \mu)v = \lambda v + \mu v$  and  $(\lambda\mu)v = \lambda(\mu v)$  for all scalars  $\lambda, \mu$  and vector  $v$ .
7.  $1v = v$  for every vector  $v$ , where  $1$  is the unit scalar (i.e.  $1\lambda = \lambda$  for every scalar  $\lambda$ ).

**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  $V$ .

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  $\lambda, \mu$ .
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  $\lambda$ . 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 every pair of vectors in the set is orthogonal and in addition none of them is equal to  $0$ .

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.

**Useful properties.** Here are some useful properties that can be proved from the abstract definitions. They 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 just careful definition-pushing, but it's reassuring to know that it can be done.

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  $\lambda$ .
13.  $(\lambda v) \cdot w = \lambda(v \cdot w)$ , for all scalars  $\lambda$  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*.

**Exercise 5.4. 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 5.5. 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 5.6. Prove useful property 13**

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

## 5.3.2. ORTHOGONAL PROJECTION AND LEAST SQUARES

**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 solves<sup>28</sup>

$$\min_{x' \in S} \|x - x'\|^2.$$

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

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

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

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  $e_i$ .

**Example 5.7 (Closest point via calculus).**

Let  $e_1 = [1, 1, 0]$ , let  $e_2 = [1, 0, -1]$ , and let  $x = [1, 2, 3]$ . Find the closest point to  $x$  in the span of  $\{e_1, e_2\}$ . Show that the residual is orthogonal to  $S$ .

*Just write out the optimization problem we want to solve:*

$$\min_{\lambda_1, \lambda_2} \|x - (\lambda_1 e_1 + \lambda_2 e_2)\|^2.$$

*We can compute the solution numerically:*

```
1 e1, e2, x = np.array([1, 1, 0]), np.array([1, 0, -1]), np.array([1, 2, 3])
2 λ1, λ2 = scipy.optimize.fmin(lambda λ: np.linalg.norm(x - λ[0]*e1 - λ[1]*e2), [0, 0])
3 λ1*e1 + λ2*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  $\lambda_1$  and  $\lambda_2$  and setting the derivatives equal to 0,*

$$\begin{aligned} \frac{\partial}{\partial \lambda_1} = 0 : \quad & -2x \cdot e_1 + 2\lambda_1 e_1 \cdot e_1 + 2\lambda_2 e_1 \cdot e_2 = 0 \\ \frac{\partial}{\partial \lambda_2} = 0 : \quad & -2x \cdot e_2 + 2\lambda_1 e_1 \cdot e_2 + 2\lambda_2 e_2 \cdot e_2 = 0 \end{aligned} \tag{23}$$

*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:*

```
1 e1 = numpy.array([1, 1, 0])
2 e2 = numpy.array([1, 0, -1])
3 x = numpy.array([1, 2, 3])
4 λ1, λ2 = numpy.linalg.solve([[e1@e1, e1@e2], [e1@e2, e2@e2]], [x@e1, x@e2])
5 λ1*e1 + λ2*e2 # array([ 0.33333333, 2.66666667, 2.33333333])
```

<sup>28</sup>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.

For geometrical insight, rearrange equations (23) 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  $e_2$ .

**Example 5.8 (Closest point via explicit projection).**

Let  $x = [1, 2, 3]$ , and let  $\tilde{x}$  be the projection onto the subspace spanned by  $e_1 = [1, 1, 0]$  and  $e_2 = [1, 0, -1]$ . Create an orthonormal basis out of  $\{e_1, e_2\}$ , and thence find the coordinates of  $\tilde{x}$  with respect to the basis  $\{e_1, e_2\}$ .

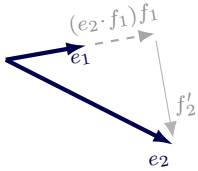
Hint: first use Useful Property 16 on page 72 to get the coordinates of  $\tilde{x}$  in the orthonormal basis.

First create the orthonormal basis. 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  $\text{span}\{e_1, e_2\} = \text{span}\{f_1, f_2\} = S$ . Thus,  $\{f_1, f_2\}$  is an orthonormal basis for  $S$ .

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

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

which with some algebra can be rewritten in terms of  $e_1$  and  $e_2$ . In numpy,

```
6 f1 = e1 / numpy.linalg.norm(e1)
7 f2' = e2 - (e2@f1) * f1
8 f2 = f2' / numpy.linalg.norm(f2')
9
10 # x in original coordinate system
11 (x@f1)*f1 + (x@f2)*f2 # array([ 0.33333333, 2.66666667, 2.33333333])
12
13 # x in terms of e1 and e2
14 g1 = numpy.array([1,0]) / numpy.linalg.norm(e1)
15 g2 = numpy.array([(e2@f1)/numpy.linalg.norm(e1), 1]) / numpy.linalg.norm(f2')
16 (lambda1, lambda2) = (x@f1)*g1 + (x@f2)*g2
17 lambda1*e1 + lambda2*e2 # array([ 0.33333333, 2.66666667, 2.33333333])
```

\* \* \*

**Colinearity and matrix rank.** In Euclidean space, if we have a collection of vectors and we stack them to form a matrix, then the *rank* of the matrix is the dimension of the space spanned by those vectors. In Python, use `numpy.linalg.matrix_rank(numpy.column_stack([e1,e2]))`.

In this example, we projected onto basis vectors  $e_1$  and  $e_2$  that were linearly independent. What happens if we 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 (23) would still be correct, but they would have multiple solutions for  $\lambda_1$  and  $\lambda_2$ .

## 5.3.3. 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. Before returning to data science, and 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

$$f = \sum_{i=1}^{\infty} (f \cdot e_i) e_i \quad (24)$$

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 (24) 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.

## 5.4. Linear regression and least squares

tl;dr. A *linear regression* is a probabilistic model of the form

$$Y \sim \text{Normal}(\beta_1 e_1 + \dots + \beta_K e_K, \sigma^2) \quad (25)$$

where  $e_1, \dots, e_K$  are covariates,  $Y$  is the random response, and  $\sigma$  and  $\beta_1, \dots, \beta_K$  are unknown parameters. Maximum likelihood estimation for this model is equivalent to least squares estimation for the corresponding linear model.

In a linear regression model, we can use all the inference tools from section 3: find confidence intervals for the parameters, test hypotheses, etc. (For inference based on resampling, use parametric resampling: first estimate the unknown parameters, then generate new observations from (25).)

To demonstrate the link between linear regression and linear models, it's easier to work through an illustration rather than to write out abstract equations.

For the Iris dataset on page 62, we investigated how petal length depends on sepal length. Consider the linear regression model

$$\text{Petal.Length}_i \sim \text{Normal}\left(\alpha + \beta \text{Sepal.Length}_i + \gamma (\text{Sepal.Length}_i)^2, \sigma^2\right) \quad (26)$$

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

Let's find the 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

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

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

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

In this equation we have switched to vector notation, and  $1$  means the vector  $[1, 1, \dots, 1]$ . This is nothing other than least squares estimation for the linear model

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

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

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

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

$$\hat{\sigma} = \sqrt{\frac{1}{n} \|y - (\hat{\alpha} 1 + \hat{\beta} e + \hat{\gamma} f)\|^2}. \quad (27)$$

## 5.4.1. NON-IDENTIFIABILITY AND CONFOUNDED FEATURES \*

tl;dr. The linear subspace spanned by the feature vectors is referred to as the *feature space*. Let  $y$  be the response vector, and let  $\tilde{y}$  be the projection of  $y$  onto the feature space. Maximum likelihood estimation is equivalent to expressing  $\tilde{y}$  as a linear combination of feature vectors.

- If the feature vectors are linearly independent, then the Projection Theorem says the coordinates of  $\tilde{y}$  in feature space are unique, thus there is a unique solution for the maximum likelihood estimators.
- If the feature vectors are linearly independent, then there are multiple ways to write  $\tilde{y}$  as a linear combination of the feature vectors, thus the maximum likelihood estimators are not unique. We say the parameters are *non-identifiable*.

Even when feature vectors are linearly independent, if they are closely correlated then the parameters will be hard to identify—for example, confidence intervals will be wide.

Features that are linearly dependent or closely correlated are said to be *confounded*.

**Example 5.9 (Detecting non-identifiability).**

For the police dataset on page 51, consider the model

$$1[\text{outcome}=\text{find}] \approx \beta_e + \gamma_g.$$

where  $e$  is the ethnicity of the suspect and  $g$  is the gender. Are the parameters identifiable?

(Previously, in section 4.1, we proposed a proper probabilistic model based on the same parameters,  $\Pr(\text{output}=\text{find}) = \beta_e + \gamma_g$ , whereas now we're treating  $1[\text{output}=\text{find}]$  as a numerical response. It's a hack to treat a binary response as a real number with an implied Normal distribution. However, (i) the question only asks about parameter identifiability, which is a question about the feature vectors not the response, and (ii) the hack can still give us interesting answers about the distribution of response.)

Write the model as a linear model using one-hot coding:

$$1[\text{outcome}=\text{find}] \approx \sum_{i \in \text{ethnicities}} \beta_i 1[e = i] + \sum_{j \in \text{genders}} \gamma_j 1[g = j].$$

Consider the matrix whose columns are made up of these eight feature vectors (five ethnicity levels, three gender levels). If the columns are linearly independent, then the rank of the matrix will be 8, otherwise it will be  $< 8$ .

one-hot coding:  
section 5.2.1 page 64

matrix rank, page 74

```
1 # Only keep rows where ethnicity and gender aren't missing
2 ethnicity_levels = ['Asian', 'Black', 'Mixed', 'Other', 'White']
3 gender_levels = ['Male', 'Female', 'Other']
4 df = police.loc[police['Officer-defined ethnicity'].isin(ethnicity_levels) \
5                & police['Gender'].isin(gender_levels)]
6
7 e = [df['Officer-defined ethnicity']==i for i in ethnicity_levels]
8 g = [df['Gender']==j for j in gender_levels]
9 X = numpy.column_stack(e + g).astype(int)
10 numpy.linalg.matrix_rank(X) # returns: 7
```

The features are linearly dependent, therefore the parameters are non-identifiable.

Data science is all about noise and uncertainty, whereas linear independence is a strict clean mathematical definition, so we shouldn't pay too much attention to strict linear independence. Here is an example with features that are linearly independent but closely correlated, which makes it hard to identify the parameters.

**Example 5.10 (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  $\text{Normal}(0.5e_i + 1.5f_i, 1)$ ,  $i = 1, \dots, 20$ . How well can we recover the coefficients 0.5 and 1.5, given that  $e$  and  $f$  are nearly identical?

*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  $\sigma$ . (When we maximize out nuisance parameters, what's left is called the profile log likelihood.)*

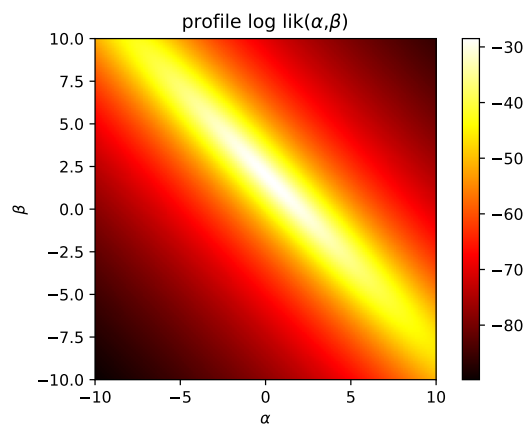
$$\text{profile log lik}(\alpha, \beta | y) = \max_{\sigma > 0} \text{log lik}(\alpha, \beta, \sigma | y)$$

*and the solution for  $\sigma$  is similar to equation (27) on page 76. Here is a numerical experiment.*

```

1 # Simulate data with these confounded features
2 e = numpy.ones(20)
3 f = numpy.ones(20)
4 f[0] = 0
5 y = numpy.random.normal(0.5*e + 1.5*f, 1)
6
7 # What are the parameter estimates?
8 model = sklearn.linear_model.LinearRegression(fit_intercept=False)
9 model.fit(numpy.column_stack([e, f]), y[:, numpy.newaxis])
10 ((alpha, beta),) = model.coef_ # on this random trial, (0.197, 1.782)
11
12 def profile_loglik(alpha, beta):
13     n = len(y)
14     s = numpy.linalg.norm(alpha*e + beta*f - y) # this gives sqrt(sum epsilon_i^2) where epsilon = residuals
15     sigma = s / numpy.sqrt(n)
16     return -n*numpy.log(sigma) - n/2*numpy.log(2*numpy.pi) - s**2/(2*sigma**2)
17
18 # Create 2d arrays, one with each alpha value, one with each beta value, one with profile log likelihood
19 alpha, beta = numpy.linspace(-10, 10, 200), numpy.linspace(-10, 10, 200)
20 A, B = numpy.meshgrid(alpha, beta)
21 pll = numpy.vectorize(profile_loglik)(A, B)
22 plt.imshow(pll, extent=(alpha[0], alpha[-1], beta[0], beta[-1]), origin='lower', cmap=plt.get_cmap('hot'))
23 plt.colorbar()

```



*The plot shows a peak at the point of maximum likelihood, and it also shows a long ‘ridge’ of high likelihood where  $\alpha + \beta$  is roughly constant. Likelihood measures how much evidence there is for an estimate; the plot is telling us that  $\alpha$  and  $\beta$  are likely to be on the ridge, but there isn't much evidence to distinguish one point on the ridge from another. In other words we can be very confident about the value of  $\alpha + \beta$  (ground truth 2, our estimate 1.979, our error 0.021), but not very confident about either of them individually (our  $\alpha$  error was 0.302, our  $\beta$  error was 0.282). If we compute confidence intervals, we'll see the same story.*



## 5.4.2. GAUSS'S INVENTION OF LEAST SQUARES \*



There is a link between linear regression and least squares estimation, but it's not just "Oh, how nice, after we've done least squares estimation we can express our answer as a probability model." Arguably, the probability model has primacy. (i) In many situations, random quantities can be approximated by a Normal distribution. (ii) Likelihood is a fundamental measure of evidence for all sorts of inference procedures. (iii) Maximum likelihood estimation for Normal random variables is equivalent to least squares estimation. (iv) Therefore, least squares estimation is a reasonable thing to do, and not just a totally heuristic kludge.

Least squares estimation was invented by Carl Friedrich Gauss, the 'prince of mathematicians', who also invented the Gaussian distribution—referred to in these notes as the Normal distribution. Here is Gauss's account<sup>29</sup> of how the idea of least squares came to him. Up to that time,

*... in every case in which it was necessary to deduce the orbits of heavenly bodies from observations, there existed advantages not to be despised, suggesting, or at any rate permitting, the application of special methods; of which advantages the chief one was, that by means of hypothetical assumptions an approximate knowledge of some elements could be obtained before the computation of the elliptic elements was commenced. Notwithstanding this, it seems somewhat strange that the general problem—To determine the orbit of a heavenly body, without any hypothetical assumption, from observations not embracing a great period of time, and not allowing the selection with a view to the application of special methods,—was almost wholly neglected up to the beginning of the present century; or at least, not treated by any one in a manner worthy its importance; since it assuredly commended itself to mathematicians by its difficulty and elegance, even if its great utility in practice were not apparent. An opinion had universally prevailed that a complete determination from observations embracing a short interval of time was impossible—an ill-founded opinion—for it is now clearly shown that the orbit of a heavenly body may be determined quite nearly from good observations embracing only a few days; and this without any hypothetical assumption.*

*Some idea occurred to me in the month of September of the year 1801, engaged at the time on a very different subject, which seemed to point to the solution of the great problem of which I have spoken. Under such circumstances we not unfrequently, for fear of being too much led away by an attractive investigation, suffer the associations of ideas, which more attentively considered, might have proved most fruitful in results, to be lost from neglect. And the same fate might have befallen these conceptions, had they not happily occurred at the most propitious moment for their preservation and encouragement that could have been selected. For just about this time the report of the new planet, discovered on the first day of January of that year with the telescope at Palermo, was the subject of universal conversation; and soon afterwards the observations made by the distinguished astronomer Piazzi from the above date to the eleventh of February were published. Nowhere in the annals of astronomy do we meet with so great an opportunity, and a greater one could hardly be imagined, for showing most strikingly, the value of this problem, than in this crisis and urgent necessity, when all hope of discovering in the heavens this planetary atom, among innumerable small stars after the lapse of nearly a year, rested solely upon a sufficiently approximate knowledge of its orbit to be based upon these very few observations. Could I ever have found a more seasonable opportunity to test the practical value of my conceptions, than now in employing them for the determination of the orbit of the planet Ceres, which during the forty-one days had described a geocentric arc of only three*

Normal approximation:  
section 2.1 page 17  
Inference based on  
likelihood: all of  
section 3

<sup>29</sup>Carl Friedrich Gauss. *Theoria motus corporum coelestium in sectionibus conicis solem ambientum*. 1809. English translation: Charles Henry Davis. *Theory of the motion of the heavenly bodies moving about the sun in conic sections*. 1857. URL: <https://quod.lib.umich.edu/m/moa/AGG8895.0001.001/15?rgn=full+text;view=image>.

*degrees, and after the lapse of a year must be looked for in a region of the heavens very remote from that in which it was last seen? This first application of the method was made in the month of October, 1801, and the first clear night, when the planet was sought for (by de Zach, December 7, 1801) as directed by the numbers deduced from it, restored the fugitive to observation. Three other new planets, subsequently discovered, furnished new opportunities for examining and verifying the efficiency and generality of the method.*

*Several astronomers wished me to publish the methods employed in these calculations immediately after the second discovery of Ceres; but many things—other occupations, the desire of treating the subject more fully at some subsequent period, and, especially, the hope that a further prosecution of this investigation would raise various parts of the solution to a greater degree of generality, simplicity, and elegance,—prevented my complying at the time with these friendly solicitations. I was not disappointed in this expectation, and I have no cause to regret the delay. For the methods first employed have undergone so many and such great changes, that scarcely any trace of resemblance remain between the method in which the orbit of Ceres was first computed, and the form given in this work. Although it would be foreign to my purpose, to narrate in detail all the steps by which these investigations have been gradually perfected, still, in several instances, particularly when the problem was one of more importance than usual, I have thought that the earlier methods ought not to be wholly suppressed. But in this work, besides the solution of the principal problems, I have given many things which, during the long time I have been engaged upon the motions of the heavenly bodies in conic sections, struck me as worthy of attention, either on account of their analytical elegance, or more especially on account of their practical utility.*

## 5.5. Generalized linear models \*

tl;dr. Consider a probabilistic regression model of the form

$$\Pr_Y(y \mid e_1, \dots, e_K) = g(y, \xi) \quad \text{where } \xi = \beta_1 e_1 + \dots + \beta_K e_K$$

where  $Y$  is the response,  $e_1, \dots, e_K$  are covariates, and  $\beta_1, \dots, \beta_K$  are unknown parameters. In other words, the response depends on the covariates only via a linear combination of parameters, weighted by covariates. This is called a *generalized linear model*.

For such models, we can reason about identifiability and confounding with the same tools as for linear regression. For many such models, there are efficient algorithms for maximum likelihood estimation.

Here's an example, the police dataset on page 51. In section 4.1.4 we studied the *logistic regression*

$$\Pr_Y(\text{find} \mid e, g) = \frac{e^\xi}{1 + e^\xi} \quad \text{i.e.} \quad \Pr_Y(y \mid e, g) = \frac{e^{\xi 1_{y=\text{find}}}}{1 + e^\xi}, \quad \xi = \alpha + \beta_e + \gamma_g$$

where  $y \in \{\text{find}, \text{nothing}\}$  is the response variable,  $e$  is the ethnicity covariate, and  $g$  is the gender covariate. The formula for  $\xi$  can be written as a linear model using one-hot coding, and the model is a generalized linear model because  $\Pr_Y$  depends on the parameters only via  $\xi$ . On page 77 we saw how to reason about identifiability of the parameters. It matches our ad hoc reasoning in section 4.1.2.

Fast algorithms for estimation in such models is a topic for a more specialized course.



## 6. Random processes

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 *random* or *stochastic process*. In Part II, you will come across random 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 numerically computing posterior distributions
- in *Data Science Principles and Practice* they are extended to become recurrent neural networks, useful for language modelling
- in *Information Theory* they are used to describe noisy communications channels, and also the data streams sent over such channels

### Example 6.1.

The Russian mathematician Andrei Markov (1856–1922) invented a new type of random process, 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])] # lookup based on the last two elements
7     nextchar = random.choice(alphabet, weights=p)
8     c.append(nextchar)
```

In this code, `next_char_prob` is a dictionary where each value  $p = \text{next\_char\_prob}[\dots]$  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 pleasurely. 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.*

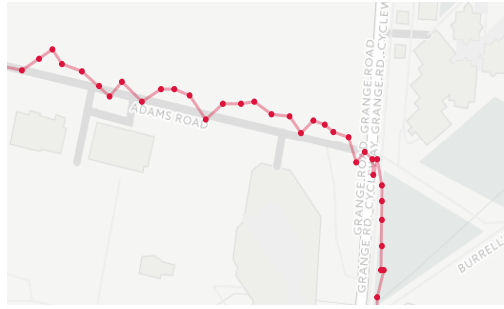
We'll develop Markov's model in sections 6.1–6.3 & 6.5. The goal in these sections is to understand the behaviour of the random process, given its parameters.

**Exercise 6.2.** Technically, a random variable is a function that can give different answers. In Markov's example,  $C$  is a random variable taking values in the space of infinite sequences. Define a function `rc` which produces an infinite sequence, evaluated on demand. It should work like this:

```
1 c = rc()
2 c[:10] # returns e.g. "once. sen "
3 c[:15] # returns "once. sen thery", i.e. from the same sequence as before
4 c = rc()
5 c[:10] # returns a new sequence
```

#### Example 6.3 (De-noising GPS).

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



A typical machine learning task is to estimate the user's true path and mode of transport, given these readings. The true path is very likely smoother than the readings—at one sample per second, the user is probably cycling, which gives us an idea of how smooth the true path is. We might treat this as a random process with two layers of randomness: the true location evolves as a random process (using randomness to model the user's changes in direction and speed and so on), and the observed data is random (using randomness to model GPS noise). Given more data we might also want to learn the user's typical parameters—acceleration profile, etc.

We'll look at a simpler version of this problem in section 6.4. The goal in that section is to make inferences about unknown quantities given data.

## 6.1. Markov chains

**tl;dr.** A *Markov chain* is a sequence  $(X_0, X_1, X_2, \dots)$  where each  $X_{n+1}$  is a discrete random variable, generated randomly based on only on  $X_n$ . The *state space* is the set of values from which the  $X_n$  are drawn.

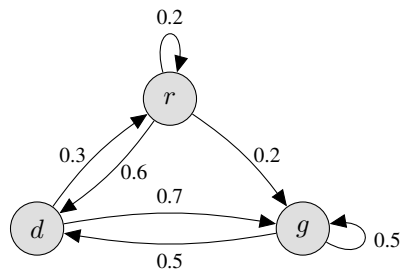
If the probability distribution does not depend on  $n$ , i.e. if there is some matrix  $P$  such that

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

we say it is a *time-homogeneous Markov chain* with *transition matrix*  $P$ .

**Example 6.4** (Markov model for Cambridge weather).

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 a time-homogenous Markov chain. We can show it either with the transition matrix, or with the corresponding *state space diagram*.



$$P = \begin{matrix} & \begin{matrix} r & d & g \end{matrix} \\ \begin{matrix} r \\ d \\ g \end{matrix} & \begin{pmatrix} 0.2 & 0.6 & 0.2 \\ 0.3 & 0 & 0.7 \\ 0 & 0.5 & 0.5 \end{pmatrix} \end{matrix}$$

When you write out a Markov state space diagram or transition matrix, double-check that every row sums to 1, i.e. that the total probability of all edges out of a node is equal to 1.

**Formal mathematical definition.** For the purposes of calculating with Markov chains, it's helpful to turn the waffly phrase “generated randomly based only on  $X_n$ ” into a formal mathematical equation,

$$\mathbb{P}(X_{n+1} = x_{n+1} \mid X_0 = x_0, X_1 = x_1, \dots, X_n = x_n) = \mathbb{P}(X_{n+1} = x_{n+1} \mid X_n = x_n) \quad \text{for all } x_0, \dots, x_{n+1}. \quad (28)$$

(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.) This equation says “Whatever the past history, all that matters for the purposes of deciding the next state is the current state.” Or “The next state is independent of the past, conditional on the current state.” This is referred to as *memorylessness*.

conditional independence is defined on page 6

### MEMORY LENGTH

In the Shakespeare example on page 83, the next character was chosen based on the previous *two* characters, which at first glance looks like it doesn't satisfy equation (28). 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  $X$ .

The rule “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 frequencies<sup>30</sup>.

*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.*

\* \* \*

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.

<sup>30</sup>Andrej 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?’ ”



## 6.2. Calculations based on memorylessness

tl;dr. Very many probabilistic models with dependencies (including Markov chains) can be represented by a *causal diagram*, a directed acyclic graph whose nodes are random variables and where the edges show which variables are used to generate which other variables. A Markov chain has a very simple causal diagram:

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

Suppose we want to calculate a probability such as  $\mathbb{P}(X_3 = x \mid X_0 = y)$ . The general strategy for problems like this is:

1. Draw the causal diagram.
2. In the probability expression that we want to calculate, identify the random variables whose distributions we want to derive (in this case  $X_3$ ). Pick out some useful ancestor variables in the causal diagram, somewhere between the variables we're given and the variables we want to calculate; these are called *latent variables* (in this case we might choose  $X_2$  and  $X_1$ , and it's an art to pick out the most useful).
3. Incorporate the latent variables into the probability expression, typically by conditioning on them using the conditional form of the law of total probability.
4. Rewrite any probabilities to have the form 'probability of a child node, conditional on its parents'.
5. Solve!

Markov calculations make heavy use of conditioning. Review the rules of conditional distributions on page 6, especially the conditional forms of the laws.

**Example 6.5 (Multi-step transition probabilities).**

Consider the Markov model for Cambridge weather on page 85. If it's grey today, what's the chance it will be grey three days from today?

The question is asking us to calculate

$$\mathbb{P}(X_3 = g \mid X_0 = g). \quad (29)$$

Step 1 says to draw a causal diagram. The underlying mechanism of a Markov chain is 'choose the next state based on the current state', so the causal diagram is

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

For step 2: We want to calculate something about the distribution of  $X_3$ , and we're given  $X_0$ , so  $X_1$  and  $X_2$  are the latent variables we might use. Let's use both of them. Step 3 says to put these values into (29) by conditioning, giving

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

Step 4 says to rewrite this with terms that have a child node on the left and its parents on the right. For the first term, the Markov property tells us that it simplifies to the form we want:

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

For the second term, shift the random variables around using the definition of conditional probability (conditional form), to put the child node on the left:

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

Another application of the Markov property gives us what we want:

$$\text{expression (29)} = \sum_{x_1, x_2} \mathbb{P}(X_3 = g \mid X_2 = x_2) \mathbb{P}(X_2 = x_2 \mid X_1 = x_1) \mathbb{P}(X_1 = x_1 \mid X_0 = g).$$

Now we can solve it. Rewriting it in terms of the transition matrix, it is

$$\begin{aligned} &= \sum_{x_1, x_2} P_{gx_1} P_{x_1 x_2} P_{x_2 g} \\ &= [P^3]_{gg} \quad \text{when written in matrix form.} \end{aligned}$$

```

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

```

**Example 6.6 (Extended Markov property).**

Let  $X$  be a Markov chain. The Markov property, equation (28), 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  $(x_0, \dots, x_{n+m})$ ,

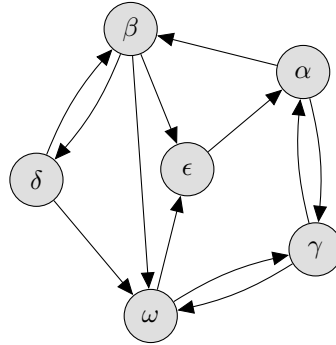
$$\begin{aligned} \mathbb{P}(X_{n+m} = x_{n+m}, \dots, X_{n+1} = x_{n+1} \mid X_n = x_n, \dots, X_0 = x_0) \\ = \mathbb{P}(X_{n+m} = x_{n+m}, \dots, X_{n+1} = x_{n+1} \mid X_n = x_n). \end{aligned}$$

Prove also that if the chain is time-homogenous then this is equal to

$$\mathbb{P}(X_m = x_{n+m}, \dots, X_1 = x_{n+1} \mid X_0 = x_n).$$

**Example 6.7 (Hitting probabilities).**

A web surfer starts at page  $\alpha$ , and from each page picks an outgoing link at random from that page. What is the chance they hit  $\omega$  before returning to  $\alpha$ ?



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

$$\mathbb{P}\left(X \text{ hits } \omega \text{ at some } n \geq 1 \mid X_0 = \alpha\right). \quad (30)$$

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 6.5. Instead, let's condition just on  $X_1$ . Using the law of total probability (conditional form),

$$(30) = \sum_{x_1} \mathbb{P}\left(X \text{ hits } \omega \text{ at some } n \geq 1 \mid X_1 = x_1, X_0 = \alpha\right) \mathbb{P}(X_1 = x_1 \mid X_0 = \alpha).$$

The second part is just  $P_{\alpha x_1}$ . For the first part, the extended Markov property (example 6.6) says that conditional on  $X_1$  the future is independent of  $X_0$ , thus

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

The final trick is to ‘reset the clock’. It doesn’t make any difference whether we start measuring time from  $n = 0$  or from  $n = 1$  thus, as in example 6.6,

$$\mathbb{P}\left(\begin{array}{l} X \text{ hits } \omega \text{ at some } n \geq 1 \\ \text{before hitting } \alpha \end{array} \middle| X_1 = x_1\right) = \mathbb{P}\left(\begin{array}{l} X \text{ hits } \omega \text{ at some } n \geq 0 \\ \text{before hitting } \alpha \end{array} \middle| X_0 = x_0\right).$$

To streamline the notation, let’s define  $\pi_x$  to be this hitting probability,

$$\pi_x = \mathbb{P}\left(\begin{array}{l} X \text{ hits } \omega \text{ at some } n \geq 0 \\ \text{before hitting } \alpha \end{array} \middle| X_0 = x\right).$$

We’ve shown that (30), the probability we want to calculate, is equal to  $\sum_x P_{\alpha x} \pi_x$ . We still need to find  $\pi_x$ . By repeating the entire conditioning argument we’ve just been through, it’s easy to show

$$\pi_x = \sum_y P_{xy} \pi_y \text{ for } x \notin \{\alpha, \omega\}, \quad \text{and} \quad \pi_\alpha = 0, \quad \pi_\omega = 1.$$

In Python we’ll let  $\pi$  be a 6-dimensional vector with elements in  $[0, 1]$ , solve the equations with matrices, then extract  $\sum_x P_{\alpha x} \pi_x = [P\pi]_\alpha$ .

```

1 import numpy as np
2 # States are in the order  $[\alpha, \beta, \gamma, \delta, \epsilon, \omega]$ 
3 # Set up an adjacency matrix for the graph, and scale it so rows sum to 1
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
9 # We want to solve  $P\pi = \pi$  i.e.  $(P - I)\pi = 0$ , except for  $\pi_\alpha$  and  $\pi_\omega$ 
10 # Bundle all the equations together in a matrix, and solve with np.linalg.lstsq
11 A = P - np.eye(6)
12 A[0, :] = [1, 0, 0, 0, 0, 0]
13 A[5, :] = [0, 0, 0, 0, 0, 1]
14 b = np.zeros(6)
15 b[5] = 1
16  $\pi = \text{np.linalg.lstsq}(A, b)[0]$  # [0, 0.333, 0.5, 0.667, 0, 1]
17
18 # Return the hitting probability we wanted to calculate
19 (P @  $\pi$ )[0] # 0.417

```

There is a general theorem behind this example. You can find this and other results about hitting times etc. in standard textbooks on Markov chains<sup>31</sup>—though often it’s just as much work to translate your problem into a theorem-ready version as it is to just solve it from first principles. Also, there will be many applications, especially in machine learning applied to Markov chains, where there aren’t ready-made theorems.

**Theorem (hitting probability).** Let  $A$  be a subset of a Markov chain’s state space. The hitting probability from  $x$  is

$$\mathbb{P}(\text{ever hit } A \mid \text{start at } x).$$

The hitting probabilities solve

$$\pi_x = \sum_y P_{xy} \pi_y \text{ for all } x \notin A, \quad \pi_x = 1 \text{ for all } x \in A.$$

Sometimes this system of equations has multiple solutions. In that case, the hitting probability from  $x$  is the minimum of  $\pi_x$  over all solutions  $\pi \geq 0$ .

For example 6.7, first delete the edges out of  $\alpha$  and put a single link from  $\alpha$  to itself. Let the original transition matrix be  $P$ , and call this modified matrix  $P'$ . Applying the theorem to  $P'$  with  $A = \{\omega\}$ , we get a solution  $\pi'_x = \mathbb{P}(\text{ever hit } \omega \mid \text{start at } x)$ . This is the same as the probability of hitting  $\omega$  before  $\alpha$  in the original Markov chain. The rest of the solution is as before.

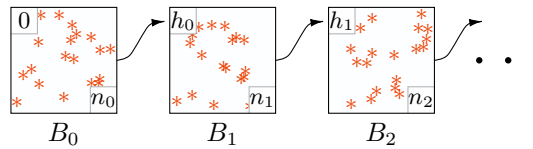
<sup>31</sup>J. R. Norris. *Markov Chains*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1997. URL: <http://www.statslab.cam.ac.uk/~james/Markov/>.

### 6.3. Application: double-spend in bitcoin \*

tl;dr. Satoshi Nakamoto's bitcoin paper contains a calculation about double-spend attacks. It's a nice illustration of how one finds a Markov chain embedded in a practical problem, and uses memorylessness for the calculations.

Bitcoin is a decentralized electronic cash system, introduced by Satoshi Nakamoto in 2008<sup>32</sup>. It has been a wild success, arguably because of the ingenious way it balances incentives<sup>33</sup>. 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.

Bitcoin is a system for storing and verify transaction records, which are depicted as \* in the diagram. A transaction record might be e.g.  $Tx_1$  = "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.



$n_0$ , a nonce that solves  $\text{hash}(0, B_0.\text{records}, n_0) < \text{threshold}$ ,

$h_0 = \text{hash}(0, B_0.\text{records})$ ,

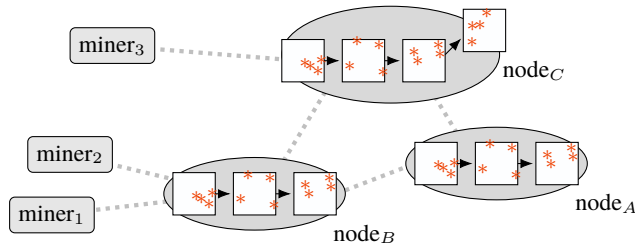
$n_1$ , a nonce that solves  $\text{hash}(h_0, B_1.\text{records}, n_1) < \text{threshold}$ ,

$h_1 = \text{hash}(h_0, B_1.\text{records})$ ,

$\dots$

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 verifies 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 10,000 nodes<sup>34</sup>, 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 roughly 5.1 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  $3.1 \times 10^{22}$  hashes must be tested to find a block, and each block contains around 1500 transactions. When a block has been mined, the nodes broadcast it to each other, and it takes roughly 580ms to reach 50% of the nodes.



<sup>32</sup>Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2008. URL: <https://bitcoin.org/bitcoin.pdf>.

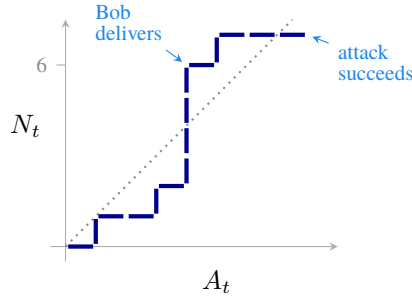
<sup>33</sup>Simon Barber et al. "Bitter to Better : How to Make Bitcoin a Better Currency". In: *Financial Cryptography—FC 2012*. Vol. 7397. Lecture Notes in Computer Science. 2012, pp. 399–414. URL: <http://www.cs.stanford.edu/~xb/fc12/>.

<sup>34</sup>Bitcoin statistics are from September 2018. Current statistics can be found at <https://dsn.tm.kit.edu/bitcoin/>, <https://bitcoinwisdom.com/bitcoin/difficulty>, [blockchain.info/charts/n-transactions-per-block](https://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 start<sup>35</sup> 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  $1 - p$ . When Bob sees  $N_t = 6$  he delivers the widget, and from then it’s a race between Alice and the rest of the network: if  $A_s > N_s$  at any subsequent time  $s$ , then Alice’s chain will be accepted and her attack succeeds. What is the probability of this?



We want to calculate the probability of the event {Alice double-spends}. A general strategy for calculating complicated probabilities is to break them down into pieces by conditioning on how the event might have happened, using the law of total probability. In this case there’s a shift at the instant Bob delivers—that’s when the race starts—so we’ll condition on the state at this instant.

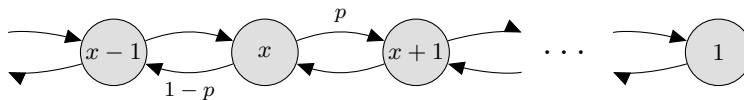
law of total probability:  
page 6

$$\mathbb{P}\left(\begin{array}{c} \text{Alice} \\ \text{double-} \\ \text{spends} \end{array}\right) = \sum_a \mathbb{P}\left(\begin{array}{c} A_t = a \text{ when} \\ \text{Bob delivers} \end{array}\right) \mathbb{P}\left(\begin{array}{c} \text{Alice} \\ \text{double-} \\ \text{spends} \end{array} \middle| \begin{array}{c} A_t = a \text{ when} \\ \text{Bob delivers} \end{array}\right). \quad (31)$$

The first term is easy: it comes from the sort of combinatorial probability you studied in IA *Maths* for *NST*. There are  $\binom{a+5}{a}$  ways to arrange  $a$  of Alice’s blocks and 6 other blocks and terminate with a non-Alice block, and so

$$\mathbb{P}\left(\begin{array}{c} A_t = a \text{ when} \\ \text{Bob delivers} \end{array}\right) = \binom{a+5}{a} p^a (1-p)^6.$$

The second term is harder. It’s a race between Alice and the rest of the network. If  $a > 6$  then she wins the race straight away; otherwise her lead starts at  $a - 6 \leq 0$  and it may go up or down, and we want to know if it ever hits 1. Every time a block is mined, either it’s Alice’s block and her lead increases by 1, and this happens with probability  $p$ ; or otherwise her lead decreases by 1, and this happens with probability  $1 - p$ . So her lead evolves like a Markov chain—formally speaking, we’ll let  $X_n$  be her lead when the  $n$ th block has been mined after Bob delivered the widget. The initial state is  $X_0 = a - 6$  and the state space diagram is



<sup>35</sup>What 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 Sompolsky and Aviv Zohar. “Bitcoin’s Security Model Revisited”. In: *CoRR* (2016). URL: <http://arxiv.org/abs/1605.09193>.

In terms of this Markov chain, the probability we want to calculate—the probability that Alice double-spends given that  $A_t = a$  when Bob delivered—is

$$\mathbb{P}(X_n \text{ hits } 1 \mid X_0 = a - 6).$$

Again we'll break this down into simpler events using the law of total probability. Let's condition on  $X_1$  as we did in example 6.7. For every  $x < 1$ ,

$$\begin{aligned} \mathbb{P}(X_n \text{ hits } 1 \mid X_0 = x) &= \sum_{x_1} \mathbb{P}(X_1 = x_1 \mid X_0 = x) \mathbb{P}(X_n \text{ hits } 1 \mid X_1 = x_1, X_0 = x) \end{aligned} \quad (32)$$

$$\begin{aligned} &= p \mathbb{P}(X_n \text{ hits } 1 \mid X_1 = x + 1, X_0 = x) + (1 - p) \mathbb{P}(X_n \text{ hits } 1 \mid X_1 = x - 1, X_0 = x) \\ &= p \mathbb{P}(X_n \text{ hits } 1 \mid X_1 = x + 1) + (1 - p) \mathbb{P}(X_n \text{ hits } 1 \mid X_1 = x - 1) \\ &= p \mathbb{P}(X_n \text{ hits } 1 \mid X_0 = x + 1) + (1 - p) \mathbb{P}(X_n \text{ hits } 1 \mid X_0 = x - 1) \end{aligned} \quad (33)$$

Equation (32) is the law of total probability, and the next equation simply fills in the only two possibilities for  $x_1$ . Equation (33) uses memorylessness, to say that conditional on  $X_1$  the future is independent of  $X_0$ . The final equation uses time homogeneity, to say that we can ‘reset the clock’ and it doesn't make any difference whether we start measuring time from  $n = 0$  or  $n = 1$ . Let's write this out as a cleaner equation:

$$\pi_x = p \pi_{x+1} + (1 - p) \pi_{x-1} \text{ for } x < 1, \quad \pi_x = 1 \text{ for } x \geq 1$$

where  $\pi_x = \mathbb{P}(X_n \text{ hits } 1 \mid X_0 = x)$ . Now we're down to a pure maths recurrence equation, with solution<sup>36</sup>

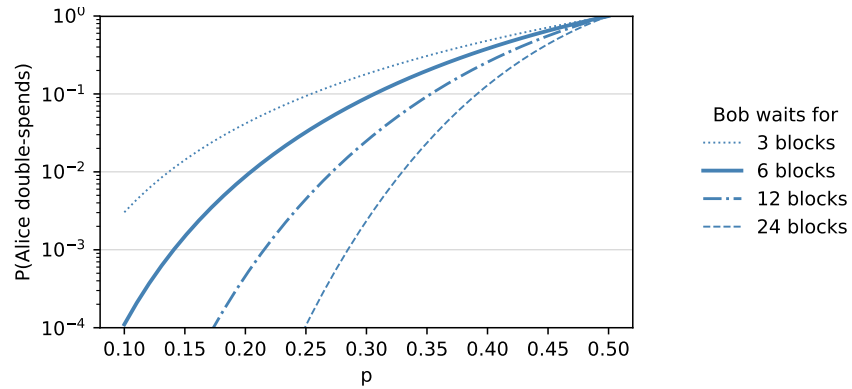
$$\pi_x = \begin{cases} \left(\frac{p}{1-p}\right)^{1-x} & \text{for } x < 1 \\ 1 & \text{for } x \geq 1 \end{cases} \quad \text{if } p < 1/2,$$

$$\pi_x = 1 \quad \text{if } p \geq 1/2.$$

Putting everything together,

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

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”.



\* \* \*

<sup>36</sup>The techniques for solving recurrence relations from scratch are very close to those for solving ODEs, which you learned about in *IA Maths for NST*. For now, you should satisfy yourself that the equations given do indeed solve the recurrence equation. The difference between the  $p < 1/2$  and  $p \geq 1/2$  cases is down to boundary conditions.

**Conditioning.** To find  $\mathbb{P}(\text{event})$ , we can condition on how the event happens. It takes skill to spot good ways of conditioning. We applied it here in two ways. Equation (31) conditioned on the state of the system at the instant that Bob delivered the widget, on the grounds that the system enters a new ‘race’ phase at that instant. And, in the race phase, equation (32) conditioned on who mined the next block. For Markov chains, you often see conditioning on either the previous state or the subsequent state, because this is a good fit for the underlying causal diagram.

causal diagram: see  
section 6.2

**Memorylessness.** The most important step in the calculation was using the memoryless property, equation (33). 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  $A - N = x + 1$ , it’s immaterial whether we reached there from  $A - N = x$  or from  $A - N = x + 2$ , the future looks exactly the same either way.

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 “ $A - N$  used to be  $x$  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 underlying system  $(A_t, N_t)$  evolves in real (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”.

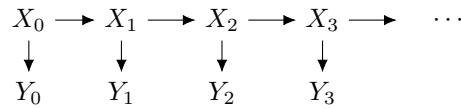
## 6.4. Inference with Markov chains \*

tl;dr. Calculations based on memorylessness, of the sort we've seen in sections 6.2–6.3, can be applied to machine learning for systems with hidden state. The new ingredient is Bayes' rule.

It's easier to work through an illustration than to write out abstract equations. Here is a simplified model of de-noising GPS readings.

**Example 6.8 (Bayesian analysis of Hidden Markov model).**

Consider a moving object. Let  $X_n$  be its location at timestep  $n$ , and assume that  $X_n$  is a time-homogenous Markov chain. Let  $Y_n$  be a noisy observation at timestep  $n$ , and assume that conditional on  $X_n$  it is independent of everything else. The causal diagram is



Write  $P$  for the transition matrix and  $E$  for the noise matrix,

$$P_{xx'} = \mathbb{P}(X_{n+1} = x' \mid X_n = x), \quad E_{xy} = \mathbb{P}(Y_n = y \mid X_n = x).$$

Given observations  $(y_0, \dots, y_n)$ , and assuming we know the distribution of the initial state, find the distribution of  $X_n$ .

Let's approach this inductively. We want to find the distribution of  $(X_n \mid y_0, \dots, y_n)$ , so let's start at  $n = 0$  and build up. It'll help to have some compact notation: let

$$\pi_n(x) = \Pr_{X_n}(x \mid y_0, \dots, y_n).$$

Finding  $\pi_0$  is a straightforward application of Bayes' rule:

$$\begin{aligned} \pi_0(x_0) &= \Pr_{X_0}(x_0 \mid Y_0 = y_0) \\ &\propto \Pr_{X_0}(x_0) \Pr_{Y_0}(y_0 \mid X_0 = x_0) \\ &= \Pr_{X_0}(x_0) E_{x_0 y_0}. \end{aligned}$$

Next for  $n = 1$ . We want to find  $\Pr_{X_1}(x_1 \mid Y_0 = y_0, Y_1 = y_1)$ . Following the general strategy laid out in section 6.2, we want to rewrite the probability to have the form 'probability of a child node conditional on its parents', i.e. we want  $Y_1$  on the left and  $X_1$  on the right. This invites another application of Bayes' rule:

$$\begin{aligned} \pi_1(x_1) &= \Pr_{X_1}(x_1 \mid Y_0 = y_0, Y_1 = y_1) \\ &\propto \Pr_{X_1}(x_1 \mid Y_0 = y_0) \Pr_{Y_1}(y_1 \mid X_1 = x_1, Y_0 = y_0) \\ &= \Pr_{X_1}(x_1 \mid Y_0 = y_0) E_{x_1 y_1}. \end{aligned} \tag{34}$$

For the second term we'll again aim for 'probability of a child node conditional on its parents', i.e. we want to introduce  $X_0$ , which we can achieve by the law of total probability:

$$\begin{aligned} \Pr_{X_1}(x_1 \mid Y_0 = y_0) &= \sum_{x_0} \Pr_{X_0}(x_0 \mid Y_0 = y_0) \Pr_{X_1}(x_1 \mid X_0 = x_0, Y_0 = y_0) \\ &= \sum_{x_0} \pi_0(x_0) P_{x_0 x_1} \quad \text{by memorylessness.} \end{aligned} \tag{35}$$

Putting (34) and (35) together,

$$\pi_1(x_1) \propto \sum_{x_0} \pi_0(x_0) P_{x_0 x_1} E_{x_1 y_1}.$$

The same reasoning works for any  $n$ , giving

$$\pi_{n+1}(x) \propto \sum_{x'} \pi_n(x') P_{x' x} E_{xy}. \tag{36}$$

We can't go any further analytically. We've reduced the problem to something that has exactly the same form as a regular Bayesian calculation, and the next step is to find a computational approximation.

This is the conditional form of Bayes' rule, page 6

This is the conditional form of the law of total probability, page 6



## PARTICLE FILTERS \*

Particle filters are a numerical tool for solving a Bayesian update equation like (36). We could in principle store  $\pi_n(x)$  with a fine mesh over  $x$  values and use numerical integration to get  $\pi_{n+1}(x)$ , but that is  $O(n^2)$  where  $n$  is the number of points in the mesh. The alternative, inspired by Monte Carlo integration and importance sampling, is to take a sample of values (‘particles’) drawn from the distribution  $\pi_n$  and use the empirical distribution as a stand-in for the true distribution. Each particle follows a random trajectory generated by  $P$ , and each particle is given a weight which is updated according to  $E$ . This takes  $O(n)$ .

Empirical distributions and Monte Carlo approximation: see page 28. Importance sampling: see page 26.

Here is an illustration<sup>37</sup>. 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 observations are gyroscope and compass and accelerometer, and the state includes position and velocity. Integrating multiple types of observation in this way is called *sensor fusion*.



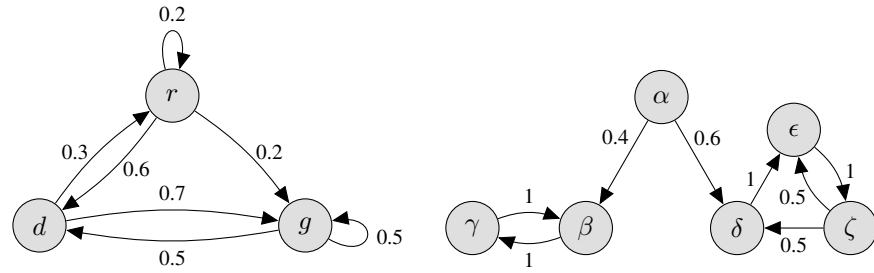
<sup>37</sup>Julian Straub. “Pedestrian Indoor Localization And Tracking Using A Particle Filter Combined with a Learning Accessibility Map”. Bachelor thesis. Technische Universität München, 2010. URL: <http://people.csail.mit.edu/jstraub/pub/Pedestrian-Indoor-Localization-and-Tracking-using-a-Particle-Filter-combined-with-a-learning-Accessibility-Map/>. The author now works at Oculus.

## 6.5. Limit theorems and equilibrium

tl;dr. When analysing Markov chains, it's often useful to be able to ask about their long-run average behaviour. We'll study

- how to classify a Markov chain: irreducible, and aperiodic
- the stationary distribution, and how to calculate it both directly and using detailed balance
- three theorems: the existence of a unique stationary distribution, ergodicity, and limiting behaviour

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 2.3 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 page 85, and a pathological case.



### 6.5.1. STATIONARY BEHAVIOUR AND IRREDUCIBILITY

**Definition.** A Markov chain is said to be *stationary* if its distribution does not change over time, i.e. if there is a vector  $\pi$  such that  $\mathbb{P}(X_n = x) = \pi_x$  for all  $n$ . Conversely, if  $\pi$  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  $\pi$  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  $\pi$  is a stationary distribution, and we pick the Markov chain’s initial state  $X_0$  randomly according to  $\pi$ , then  $X_1$  will have distribution  $\pi$  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  $\pi$ —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 6.5

**Finding a stationary distribution.** We can find a stationary distribution using the same sort of calculations based on memorylessness that we used in section 6.2. 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  $\pi$  must satisfy

$$\pi_x = \sum_y \pi_y P_{yx} \quad \text{for all } x \tag{37}$$

where  $P$  is the transition matrix.

**Exercise 6.9 (Computing the stationary distribution).**  
Find the stationary distribution of Cambridge weather.

Writing out in longhand the equations from (37),

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

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  $\pi$  is a solution then so is  $\kappa\pi$  for any constant  $\kappa$ . To pin  $\pi$  down we need an extra equation, an equation that comes from the fact that  $\pi$  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 (37) 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  $\pi$  is a one-dimensional array then it can be used either as a row vector or a column vector. In  $\pi P$  it is treated as a row vector, and in  $(P - I)^\top \pi$  it is treated as a column vector.

**Exercise 6.10 (Computing the stationary distributions).**  
Show that the pathological Markov chain has multiple stationary distributions. Find them all.

We can compute a stationary distribution for the pathological Markov chain using exactly the same method, but there is a problem: equation (37) 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

$$[\pi_\alpha, \pi_\beta, \pi_\gamma, \pi_\delta, \pi_\epsilon, \pi_\zeta] = a [0, 1/2, 1/2, 0, 0, 0] + (1 - a) [0, 0, 0, 1/5, 2/5, 2/5] \quad (38)$$

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 (38) 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  $\pi$  to

$$\pi = \pi P, \quad \pi^\top 1 = 1. \quad (39)$$

**Example 6.11.** 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  $\beta$  to  $\alpha$ .

### 6.5.2. 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  $\pi$  such that

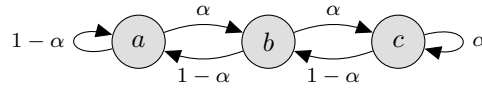
$$\pi_x P_{xy} = \pi_y P_{yx} \quad \text{for all states } x \text{ and } y \quad (40)$$

then  $\pi$  solves  $\pi = \pi P$ . Equation (40) is called the *detailed balance* condition. This theorem is trivial to prove: just write out (37) and substitute in (40).

If the chain is irreducible, then the theorem of Section 6.5.1 tells us that there is a unique stationary distribution. If we have found a distribution  $\pi$  that solves the detailed balance condition, then  $\pi$  must be that unique stationary distribution.

**Exercise 6.12 (Stationary distribution using detailed balance).**

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 6.5.1 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 (39). In this case, the detailed balance equations are*

$$\text{for } (a, b) \text{ and } (b, a): \quad \pi_a \alpha = \pi_b (1 - \alpha)$$

$$\text{for } (a, c) \text{ and } (c, a): \quad \pi_a 0 = \pi_c 0$$

$$\text{for } (b, c) \text{ and } (c, b): \quad \pi_b \alpha = \pi_c (1 - \alpha)$$

$$\text{for } (a, a) \text{ etc.:} \quad \pi_a (1 - \alpha) = \pi_a (1 - \alpha) \text{ etc.}$$

*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 6.13** (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 (39), 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.

**6.5.3. ERGODIC THEOREM**

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

$$\mathbb{E}\left(\frac{1}{n} \sum_{i=1}^n 1_{X_i=x}\right) \rightarrow \pi_x \quad \text{as } n \rightarrow \infty, \text{ for all states } x. \quad (41)$$

If the Markov chain's initial state  $X_0$  were chosen from distribution  $\pi$ , then we know from Section 6.5.1 that  $X_n$  would have distribution  $\pi$  for every  $n$ , thus  $\mathbb{E} 1_{X_i=x} = \mathbb{P}(X_i = x) = \pi_x$  for all  $i$ , and so (41) 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 6.14.** 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  $\delta$ , 40% in  $\epsilon$ , and 40% in  $\zeta$ .

**6.5.4. LIMITING BEHAVIOUR AND APERIODICITY**

In the Cambridge weather Markov chain, the ergodic theorem tells us that the long-run fraction of rainy days is equal to  $\pi_r$ , where  $\pi$  is the stationary distribution. So we'd expect that, if we pick a day arbitrarily, the probability of rain is  $\pi_r$ . This works for the Cambridge weather example, but it doesn't always work... the caveat is illustrated by states  $\beta$  and  $\gamma$  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 6.4.

This is most useful as a tool for generating a random variable from distribution  $\pi$ . In many applications, from statistical physics to Bayesian inference, we want to be able to generate a random variable from a distribution  $\pi$  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  $\pi$ . Then we can generate a random variable from distribution  $\pi$  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 6.15.** Consider the three-state Markov chain consisting of states  $\delta$ ,  $\epsilon$ , and  $\zeta$  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  $\delta$ , 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  $\delta$  to  $\delta$  in  $n$  steps?

$n$	can get from $\delta$ to $\delta$ 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  $\delta$  to  $\delta$  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.)

## A. Standard random variables

The first place to look up a random variable is Wikipedia. Here's an example, the entry for the Geometric distribution. This table has two columns, because there are two standard ways to defined this random variable. Some terminology: *support* means “ $X$  takes values in ...”; *pmf* and *pdf* refer to  $\Pr_X$ ; and *CDF* is the cumulative distribution function  $\mathbb{P}(X \leq \cdot)$ .

Parameters	$0 < p < 1$ success probability (real)	$0 < p \leq 1$ success probability (real)
Support	$k$ trials where $k \in \{1, 2, 3, \dots\}$	$k$ failures where $k \in \{0, 1, 2, 3, \dots\}$
Probability mass function (pmf)	$(1-p)^{k-1}p$	$(1-p)^k p$
CDF	$1 - (1-p)^k$	$1 - (1-p)^{k+1}$
Mean	$\frac{1}{p}$	$\frac{1-p}{p}$
Median	$\lceil \frac{-1}{\log_2(1-p)} \rceil$ (not unique if $-1/\log_2(1-p)$ is an integer)	$\lceil \frac{-1}{\log_2(1-p)} \rceil - 1$ (not unique if $-1/\log_2(1-p)$ is an integer)
Mode	1	0
Variance	$\frac{1-p}{p^2}$	$\frac{1-p}{p^2}$

In Python, `numpy` and `scipy.stats` have useful functions for working with random variables. They have a consistent naming convention, shown here for the Normal distribution.

`numpy.random.normal(..., size= $n$ )`

Generate  $n$  independent random variables from the Normal distribution. The ... are parameters, different for each distribution.

`scipy.stats.norm.pdf(x= $x$ , ...)`

the probability density function  $\Pr(x)$

`scipy.stats.norm.cdf(x= $x$ , ...)`

the cumulative distribution function  $\mathbb{P}(X \leq x)$

`scipy.stats.norm.ppf(q= $q$ , ...)`

the inverse of the cumulative distribution function, returns  $x$  such that  $\mathbb{P}(X \leq x) = q$ ;

for discrete random variables, when `cdf` jumps up in steps, returns  $\min\{x : \mathbb{P}(X \leq x) \geq q\}$

`scipy.stats.norm.mean(...), median, var, std`

summaries of the distribution

Data science computation often involves small probabilities, so watch out for bugs arising from numerical overflow and underflow. It's usually a good idea to work with log probabilities and with the *survival function*  $\text{sf}(x) = \mathbb{P}(X > x)$ .

`scipy.stats.norm.logpdf(x, ...)`

$\log \Pr(x)$

`scipy.stats.norm.logcdf(x, ...)`

$\log \mathbb{P}(X \leq x)$

`scipy.stats.norm.sf(x, ...), logsf`

$\mathbb{P}(X > x)$  and  $\log \mathbb{P}(X > x)$

## A.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

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

Mean  $1/p$ , variance  $(1 - p)/p^2$ . 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

$$\Pr(x) = \lambda e^{-\lambda x}, \quad \mathbb{P}(X \geq x) = e^{-\lambda x}.$$

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

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

Mean  $1/\lambda$ , variance  $1/\lambda^2$ . 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  $X \sim \text{Binom}(n, p)$ . This random variable takes values in  $\{0, 1, \dots, n\}$ , and

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

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 6.3 when we calculated  $\mathbb{P}(A_t = a \text{ when Bob delivers})$ .

Mean  $np$ , variance  $np(1 - p)$ . In Python, `numpy.random.binomial(n, p)`.

**Multinomial:** If we have  $n$  individuals each of whom falls into one of  $K$  categories, and the probability of falling into category  $k$  is  $p_k$ , then the total number in each category is a multivariate random variable  $X \sim \text{Multinom}(n, p)$ . It takes values in  $\{0, 1, \dots, n\}^K$ , and

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

(The binomial distribution is the special case when  $k = 2$ .)

In Python, `numpy.random.multinomial(n, p)`.

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

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

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  $X \sim \text{Poisson}(\lambda t)$ .

Mean  $\lambda$ , variance  $\lambda$ . In Python, `numpy.random.poisson(lam=\lambda)`.



## A.2. 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  $X \sim \text{Normal}(\mu, \sigma^2)$ , then  $X$  is a continuous random variable taking values in the entire real line, and

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

There is also a multivariate version, called the multivariate normal. Here are some useful facts about the Normal distribution. If  $X \sim \text{Normal}(\mu, \sigma^2)$ , and  $Y \sim \text{Normal}(\nu, \rho^2)$  is independent, and  $a$  and  $b$  are real numbers, then

$$\begin{aligned} \mathbb{P}(\mu - 1.96\sigma \leq X \leq \mu + 1.96\sigma) &= 95\% \\ aX + b &\sim \text{Normal}(a\mu + b, a^2\sigma^2) \\ (X - \mu)/\sigma &\sim \text{Normal}(0, 1) \\ X + Y &\sim \text{Normal}(\mu + \nu, \sigma^2 + \rho^2) \end{aligned}$$

In Python, `numpy.random.normal(loc= $\mu$ , scale= $\sigma$ )`, and watch out for  $\sigma$  versus  $\sigma^2$ !

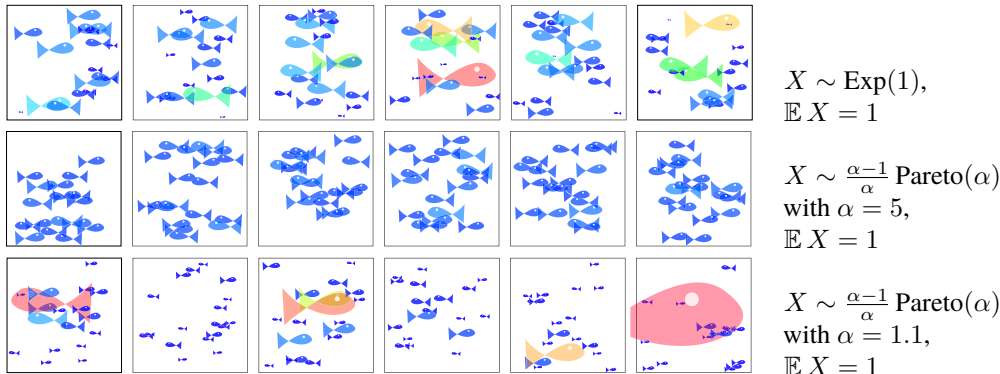
**Pareto and lognormal:** Some natural phenomena, like sizes of forest fires, or insurance claims, or Internet traffic volumes, or stock market crashes, have the characteristic that there are events of wildly different sizes. This tends to cause problems for simulations and forecasting, since the entire outcome can hinge on one ‘black swan’ event<sup>38</sup>. 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

$$\Pr(x) = \alpha x^{-(\alpha+1)}, \quad \mathbb{P}(X \geq x) = x^{-\alpha}.$$

The mean and variance become  $\infty$  for small  $\alpha$ ,

$$\mathbb{E} X = \begin{cases} \infty & \text{if } \alpha \leq 1 \\ \alpha/(\alpha - 1) & \text{otherwise,} \end{cases} \quad \text{Var } X = \begin{cases} \infty & \text{if } \alpha \leq 2 \\ \alpha / (\alpha - 1)^2(\alpha - 2)^2 & \text{otherwise.} \end{cases}$$

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.

<sup>38</sup>Nassim 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 texts. 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  $\text{Zipf}(n, 1/\alpha)$ .

### A.3. Variables for inference

**Beta:** If we toss a biased coin  $n$  times, and each coin has chance  $p$  of heads, then the number of heads has a  $\text{Bin}(n, p)$  distribution. In Bayesian inference, a common prior distribution for  $p$  is  $\text{Beta}(\alpha, \beta)$ . It takes values in  $(0, 1)$ , and has parameters  $\alpha > 0$  and  $\beta > 0$ , and density

$$\text{Pr}(p) = \binom{\alpha + \beta - 1}{\alpha - 1} p^{\alpha-1} (1-p)^{\beta-1}$$

(but with a generalized form of the binomial coefficient when  $\alpha$  and  $\beta$  are non-integer). It has mean  $\alpha/(\alpha + \beta)$ , and the rough interpretation is “I’ve seen  $\alpha$  heads and  $\beta$  tails”.

In Python, `numpy.random.beta(a= $\alpha$ , b= $\beta$ )`.

**Dirichlet:** The Dirichlet distribution  $\text{Dir}(\alpha)$  is a generalization of the Beta distribution. Instead of two categories (heads and tails), it allows  $K \geq 2$  categories, and  $\alpha$  is a vector in  $\mathbb{R}^K$ . It is a continuous random variable, and it takes values in

$$\Omega = \{[x_1, \dots, x_K] \in (0, 1)^K : x_1 + \dots + x_K = 1\}.$$

In other words, it generates probability distributions over the  $K$  categories. It is used in Bayesian inference to describe belief about a multinomial distribution, and the rough interpretation is “I’ve seen  $\alpha_k$  items in category  $k$ ”. Its density function is

$$\text{Pr}([x_1, \dots, x_K]) \propto x_1^{\alpha_1-1} x_2^{\alpha_2-1} \dots x_K^{\alpha_K-1}.$$

In Python, `numpy.random.dirichlet(alpha= $\alpha$ )`.

**Gamma:** The Gamma distribution  $X \sim \Gamma(k, \lambda)$  is a continuous random variable taking values in  $[0, \infty)$ , and its parameters are  $k > 0$  and  $\lambda > 0$ . It arises in two places: it’s the sum of  $k$  independent Exponential random variables; and it’s a common choice of prior distribution for  $1/\sigma^2$  in Bayesian calculations with  $\text{Normal}(\mu, \sigma^2)$  random variables. (Engineers call  $1/\sigma^2$  the ‘precision’.) It has density

$$\text{Pr}(x) = \frac{\lambda^k x^{k-1} e^{-\lambda x}}{(k-1)!}$$

(but with  $(k-1)!$  replaced by the gamma function  $\Gamma(k)$  for non-integer  $k$ ).

Mean  $k/\lambda$ , variance  $k/\lambda^2$ . In Python, `numpy.random.gamma(shape= $k$ , scale= $1/\lambda$ )`.