Information Theory, Inference, and Learning Algorithms

David J.C. MacKay

CAMBRIDGE
UNIVERSITY PRESS

Information Theory, Inference, and Learning Algorithms

David J.C. MacKay
mackay@mrao.cam.ac.uk

Version 7.2 (fourth printing) March 28, 2005

Please send feedback on this book via
http://www.inference.phy.cam.ac.uk/mackay/itila/

Version 6.0 of this book was published by C.U.P. in September 2003. It will remain viewable on-screen on the above website, in postscript, djvu, and pdf formats.

In the second printing (version 6.6) minor typos were corrected, and the book design was slightly altered to modify the placement of section numbers.

In the third printing (version 7.0) minor typos were corrected, and chapter 8 was renamed 'Dependent random variables' (instead of 'Correlated').

In the fourth printing (version 7.2) minor typos were corrected.

(C.U.P. replace this page with their own page ii.)

Contents

Preface

This book is aimed at senior undergraduates and graduate students in Engineering, Science, Mathematics, and Computing. It expects familiarity with calculus, probability theory, and linear algebra as taught in a first- or second-year undergraduate course on mathematics for scientists and engineers.

Conventional courses on information theory cover not only the beautiful *theoretical* ideas of Shannon, but also *practical* solutions to communication problems. This book goes further, bringing in Bayesian data modelling, Monte Carlo methods, variational methods, clustering algorithms, and neural networks.

Why unify information theory and machine learning? Because they are two sides of the same coin. In the 1960s, a single field, cybernetics, was populated by information theorists, computer scientists, and neuroscientists, all studying common problems. Information theory and machine learning still belong together. Brains are the ultimate compression and communication systems. And the state-of-the-art algorithms for both data compression and error-correcting codes use the same tools as machine learning.

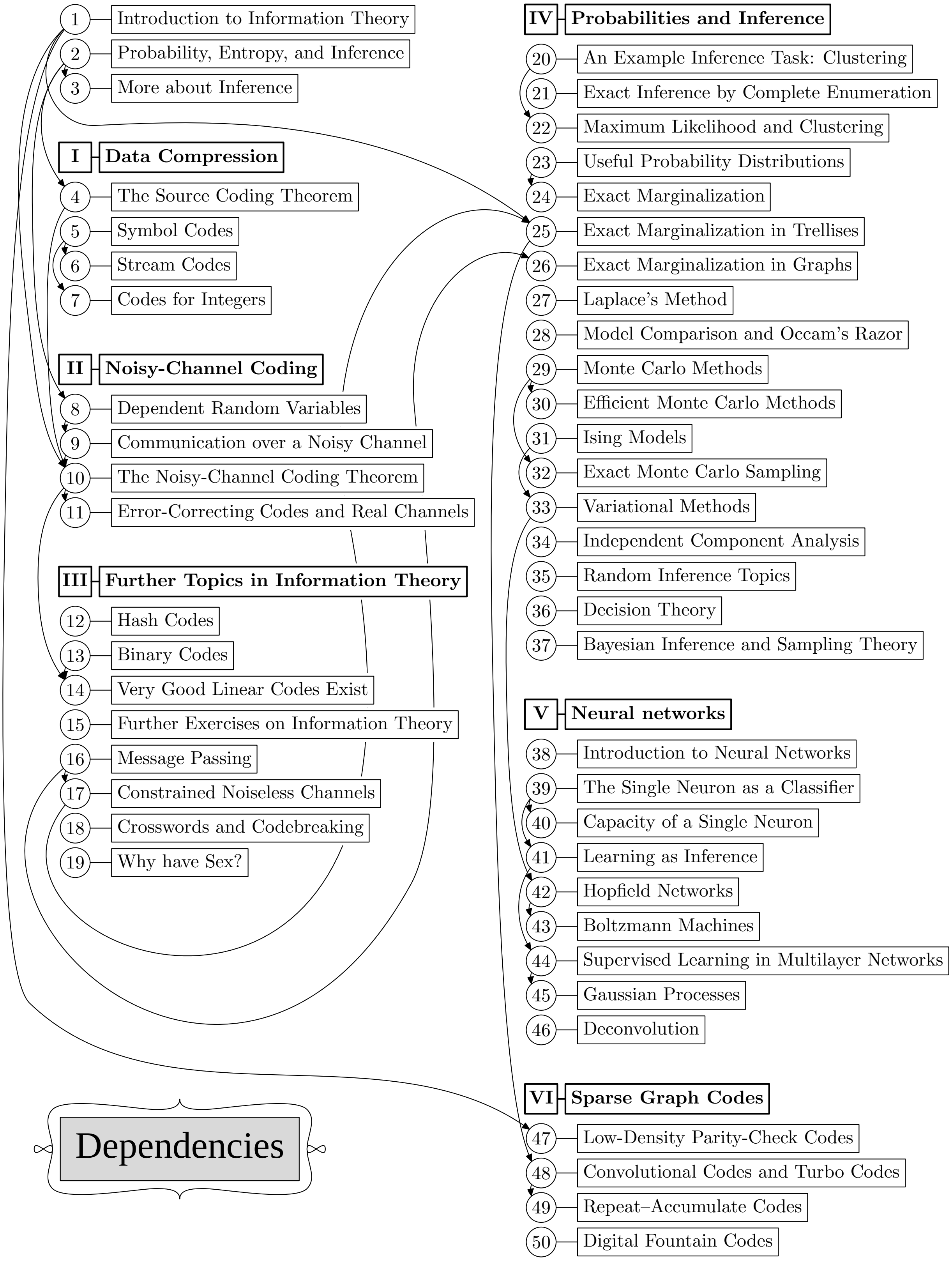
How to use this book

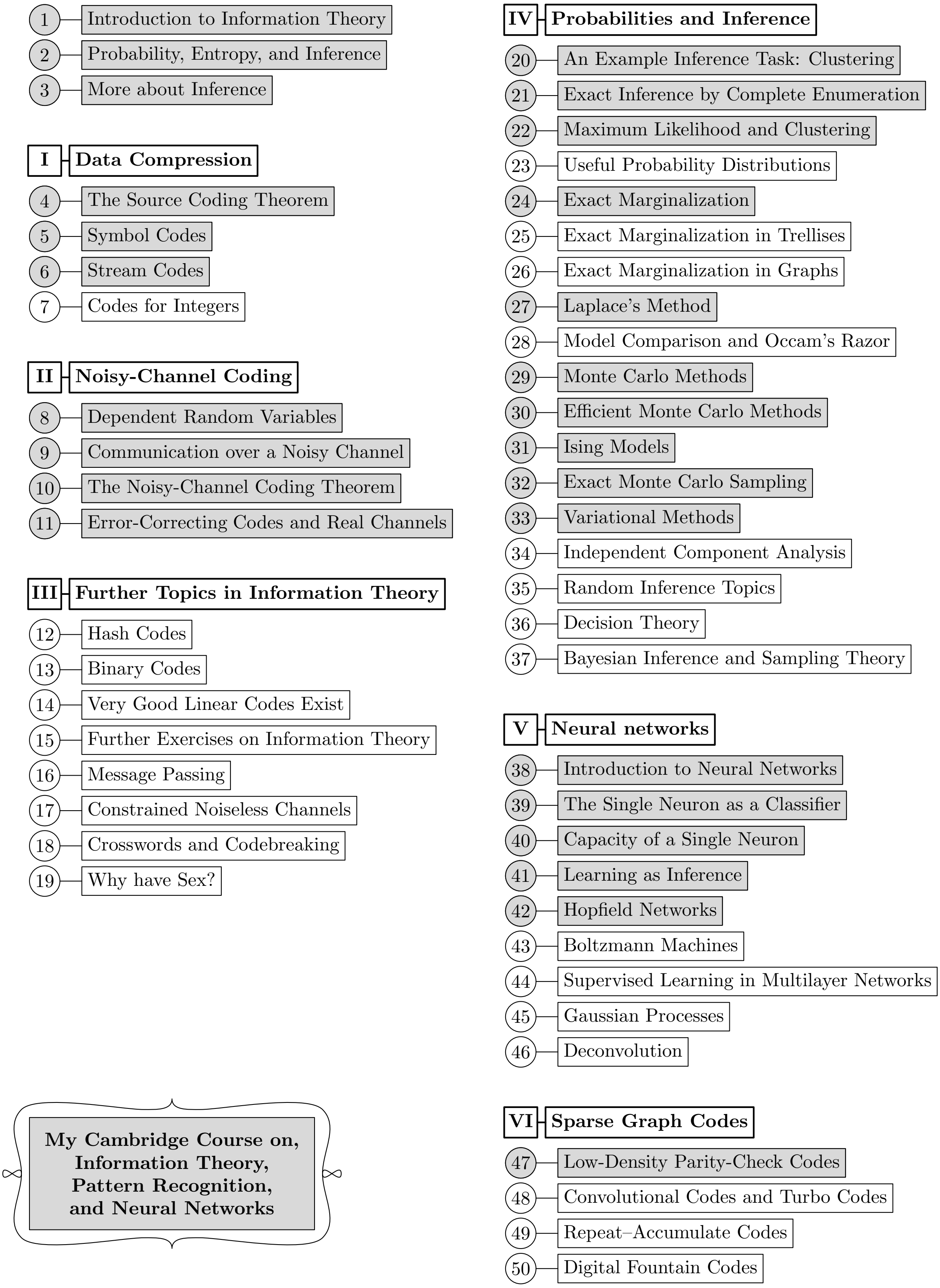
The essential dependencies between chapters are indicated in the figure on the next page. An arrow from one chapter to another indicates that the second chapter requires some of the first.

Within Parts I, II, IV, and V of this book, chapters on advanced or optional topics are towards the end. All chapters of Part III are optional on a first reading, except perhaps for Chapter 16 (Message Passing).

The same system sometimes applies within a chapter: the final sections often deal with advanced topics that can be skipped on a first reading. For example in two key chapters – Chapter 4 (The Source Coding Theorem) and Chapter 10 (The Noisy-Channel Coding Theorem) – the first-time reader should detour at section 4.5 and section 10.4 respectively.

Pages vii–x show a few ways to use this book. First, I give the roadmap for a course that I teach in Cambridge: 'Information theory, pattern recognition, and neural networks'. The book is also intended as a textbook for traditional courses in information theory. The second roadmap shows the chapters for an introductory information theory course and the third for a course aimed at an understanding of state-of-the-art error-correcting codes. The fourth roadmap shows how to use the text in a conventional course on machine learning.


1 Introduction to Information Theory
2 Probability, Entropy, and Inference
3 More about Inference
I Data Compression
4 The Source Coding Theorem
5 Symbol Codes
6 Stream Codes
7 Codes for Integers
II Noisy-Channel Coding
8 Dependent Random Variables
9 Communication over a Noisy Channel
10 The Noisy-Channel Coding Theorem
11 Error-Correcting Codes and Real Channels
III Further Topics in Information Theory
12 Hash Codes
13 Binary Codes
14 Very Good Linear Codes Exist
15 Further Exercises on Information Theory
16 Message Passing
17 Constrained Noiseless Channels
18 Crosswords and Codebreaking
19 Why have Sex?
Dependencies
IV Probabilities and Inference
20 An Example Inference Task: Clustering
21 Exact Inference by Complete Enumeration
22 Maximum Likelihood and Clustering
23 Useful Probability Distributions
24 Exact Marginalization
25 Exact Marginalization in Trellises
26 Exact Marginalization in Graphs
27 Laplace's Method
28 Model Comparison and Occam's Razor
29 Monte Carlo Methods
30 Efficient Monte Carlo Methods
31 Ising Models
32 Exact Monte Carlo Sampling
33 Variational Methods
34 Independent Component Analysis
35 Random Inference Topics
36 Decision Theory
37 Bayesian Inference and Sampling Theory
V Neural networks
38 Introduction to Neural Networks
39 The Single Neuron as a Classifier
40 Capacity of a Single Neuron
41 Learning as Inference
42 Hopfield Networks
43 Boltzmann Machines
44 Supervised Learning in Multilayer Networks
45 Gaussian Processes
46 Deconvolution
VI Sparse Graph Codes
47 Low-Density Parity-Check Codes
48 Convolutional Codes and Turbo Codes
49 Repeat–Accumulate Codes
50 Digital Fountain Codes



1 Introduction to Information Theory
2 Probability, Entropy, and Inference
3 More about Inference

I Data Compression

4 The Source Coding Theorem
5 Symbol Codes
6 Stream Codes
7 Codes for Integers

II Noisy-Channel Coding

8 Dependent Random Variables
9 Communication over a Noisy Channel
10 The Noisy-Channel Coding Theorem
11 Error-Correcting Codes and Real Channels

III Further Topics in Information Theory

12 Hash Codes
13 Binary Codes
14 Very Good Linear Codes Exist
15 Further Exercises on Information Theory
16 Message Passing
17 Constrained Noiseless Channels
18 Crosswords and Codebreaking
19 Why have Sex?

My Cambridge Course on, Information Theory, Pattern Recognition, and Neural Networks

IV Probabilities and Inference

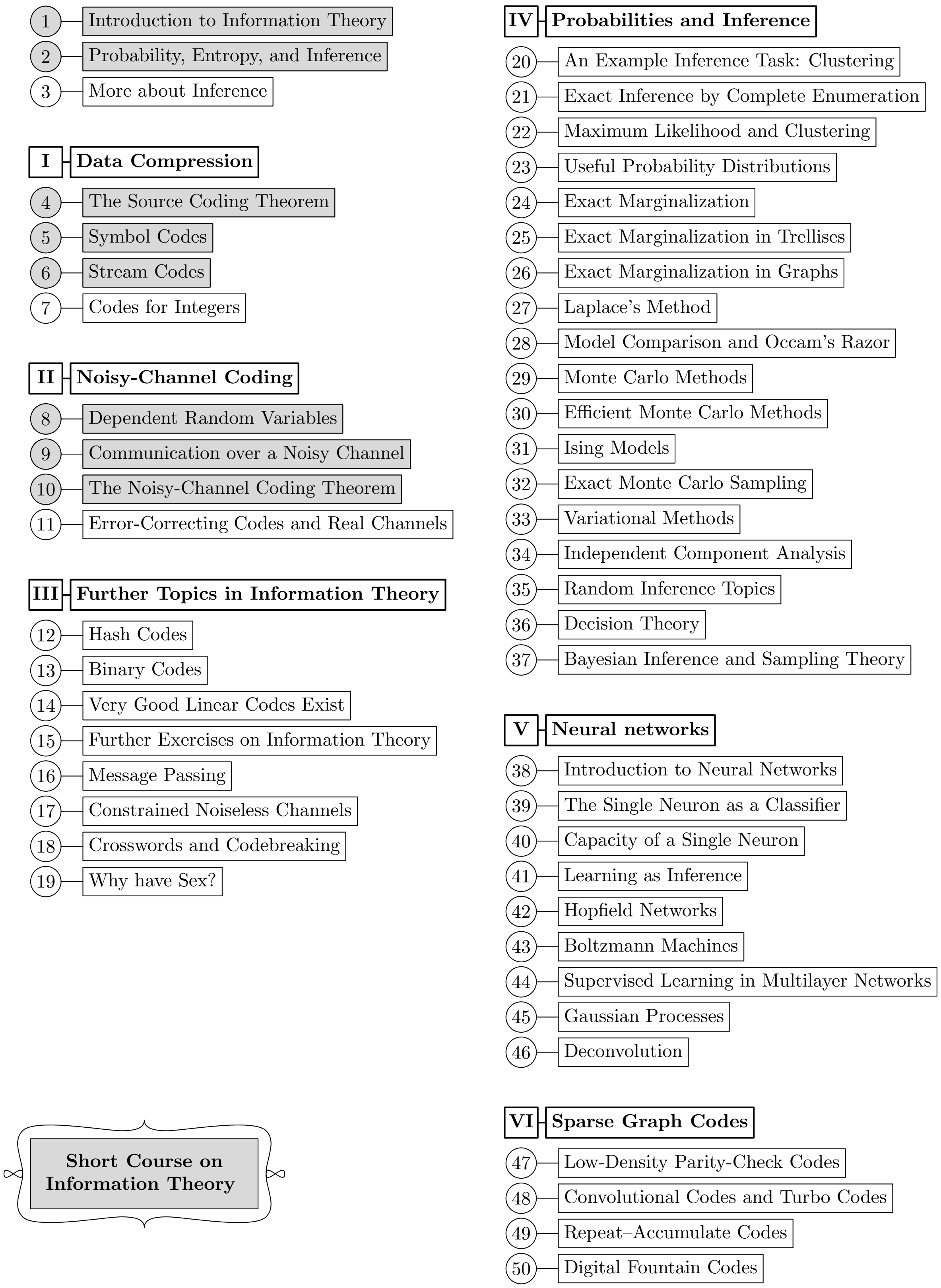
20 An Example Inference Task: Clustering
21 Exact Inference by Complete Enumeration
22 Maximum Likelihood and Clustering
23 Useful Probability Distributions
24 Exact Marginalization
25 Exact Marginalization in Trellises
26 Exact Marginalization in Graphs
27 Laplace's Method
28 Model Comparison and Occam's Razor
29 Monte Carlo Methods
30 Efficient Monte Carlo Methods
31 Ising Models
32 Exact Monte Carlo Sampling
33 Variational Methods
34 Independent Component Analysis
35 Random Inference Topics
36 Decision Theory
37 Bayesian Inference and Sampling Theory

V Neural networks

38 Introduction to Neural Networks
39 The Single Neuron as a Classifier
40 Capacity of a Single Neuron
41 Learning as Inference
42 Hopfield Networks
43 Boltzmann Machines
44 Supervised Learning in Multilayer Networks
45 Gaussian Processes
46 Deconvolution

VI Sparse Graph Codes

47 Low-Density Parity-Check Codes
48 Convolutional Codes and Turbo Codes
49 Repeat–Accumulate Codes
50 Digital Fountain Codes



1 Introduction to Information Theory
2 Probability, Entropy, and Inference
3 More about Inference

I Data Compression
4 The Source Coding Theorem
5 Symbol Codes
6 Stream Codes
7 Codes for Integers

II Noisy-Channel Coding
8 Dependent Random Variables
9 Communication over a Noisy Channel
10 The Noisy-Channel Coding Theorem
11 Error-Correcting Codes and Real Channels

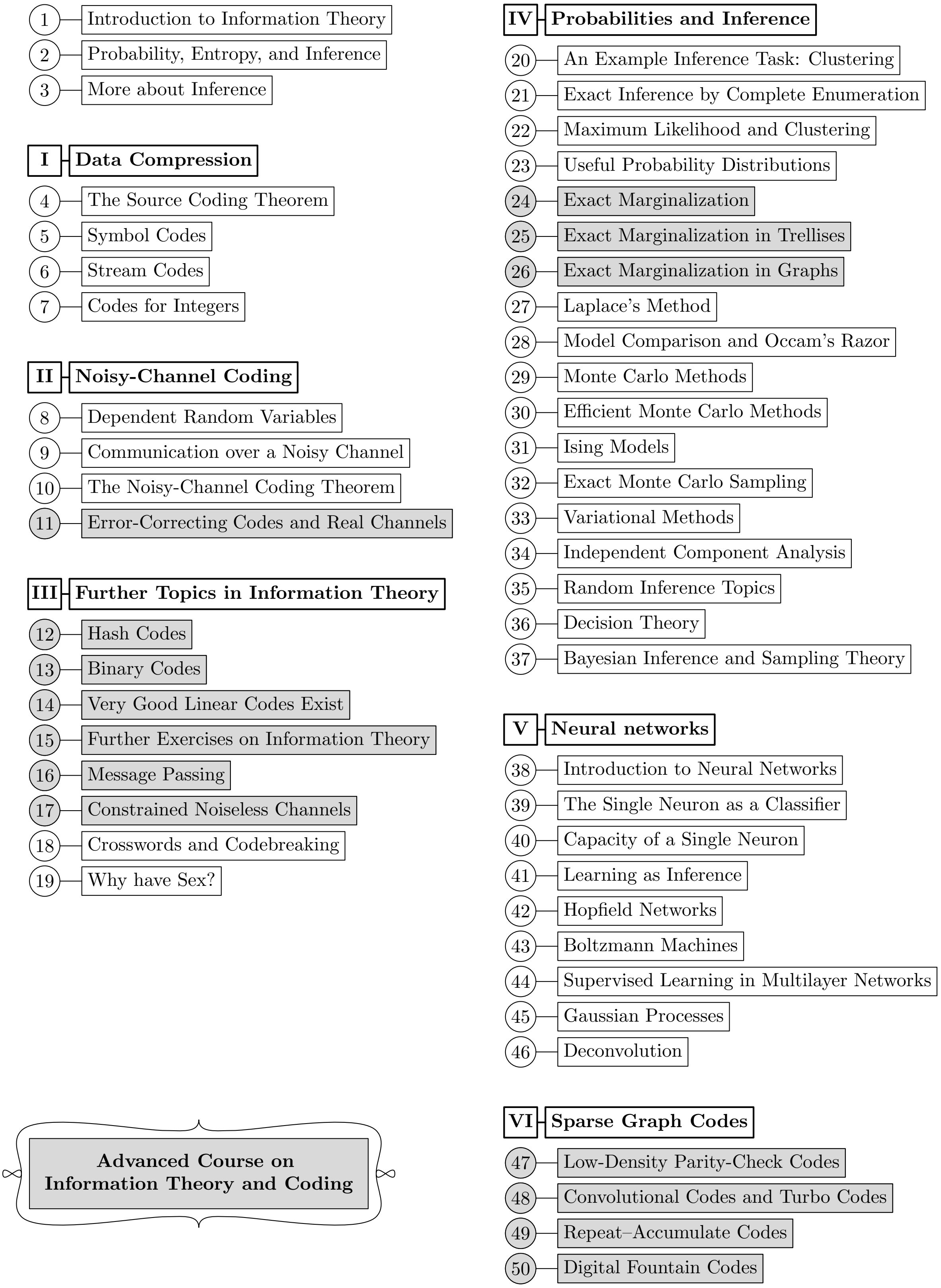
III Further Topics in Information Theory
12 Hash Codes
13 Binary Codes
14 Very Good Linear Codes Exist
15 Further Exercises on Information Theory
16 Message Passing
17 Constrained Noiseless Channels
18 Crosswords and Codebreaking
19 Why have Sex?

Short Course on Information Theory

IV Probabilities and Inference
20 An Example Inference Task: Clustering
21 Exact Inference by Complete Enumeration
22 Maximum Likelihood and Clustering
23 Useful Probability Distributions
24 Exact Marginalization
25 Exact Marginalization in Trellises
26 Exact Marginalization in Graphs
27 Laplace's Method
28 Model Comparison and Occam's Razor
29 Monte Carlo Methods
30 Efficient Monte Carlo Methods
31 Ising Models
32 Exact Monte Carlo Sampling
33 Variational Methods
34 Independent Component Analysis
35 Random Inference Topics
36 Decision Theory
37 Bayesian Inference and Sampling Theory

V Neural networks
38 Introduction to Neural Networks
39 The Single Neuron as a Classifier
40 Capacity of a Single Neuron
41 Learning as Inference
42 Hopfield Networks
43 Boltzmann Machines
44 Supervised Learning in Multilayer Networks
45 Gaussian Processes
46 Deconvolution

VI Sparse Graph Codes
47 Low-Density Parity-Check Codes
48 Convolutional Codes and Turbo Codes
49 Repeat–Accumulate Codes
50 Digital Fountain Codes



1 Introduction to Information Theory
2 Probability, Entropy, and Inference
3 More about Inference

I Data Compression
4 The Source Coding Theorem
5 Symbol Codes
6 Stream Codes
7 Codes for Integers

II Noisy-Channel Coding
8 Dependent Random Variables
9 Communication over a Noisy Channel
10 The Noisy-Channel Coding Theorem
11 Error-Correcting Codes and Real Channels

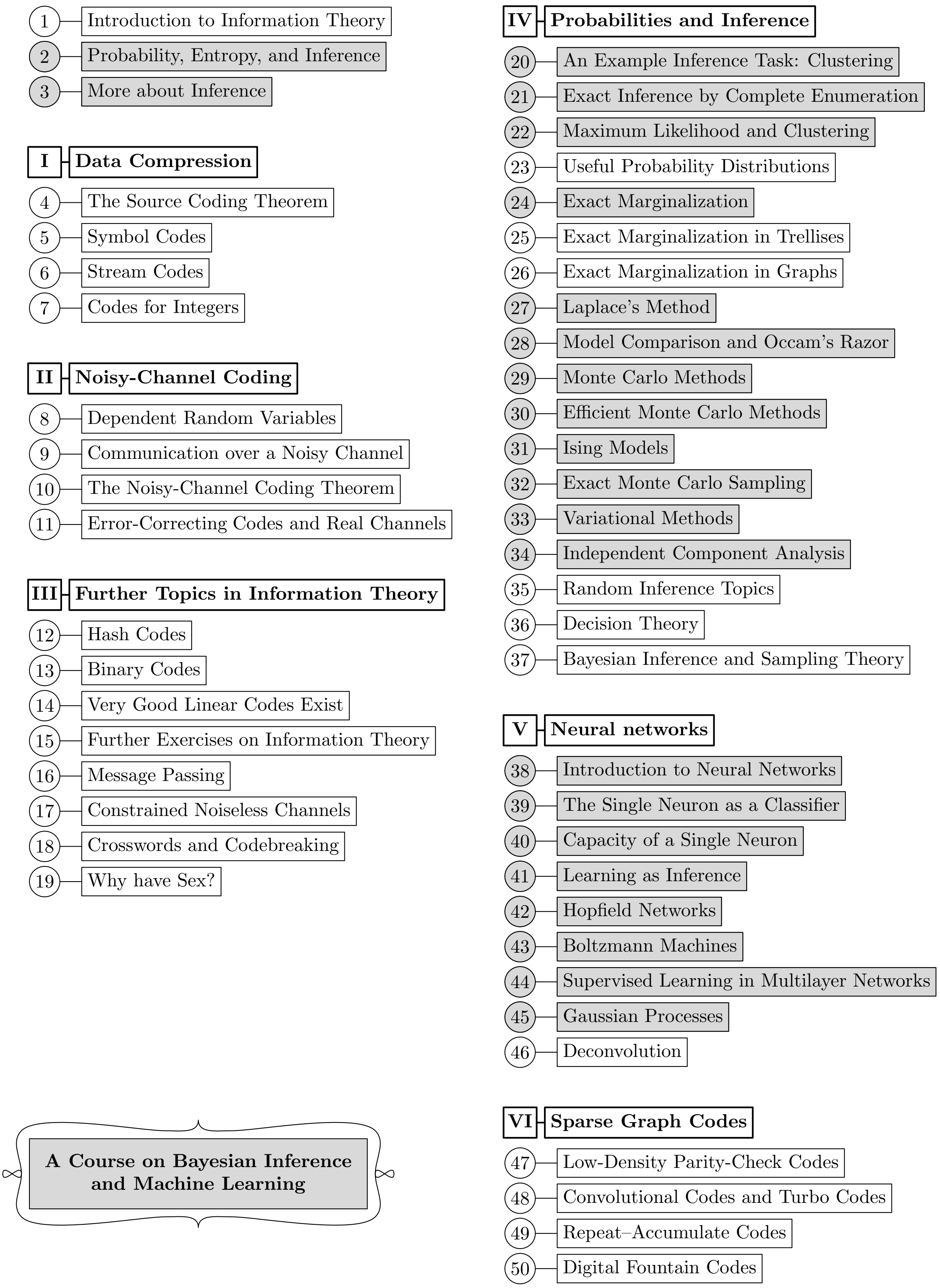
III Further Topics in Information Theory
12 Hash Codes
13 Binary Codes
14 Very Good Linear Codes Exist
15 Further Exercises on Information Theory
16 Message Passing
17 Constrained Noiseless Channels
18 Crosswords and Codebreaking
19 Why have Sex?

Advanced Course on Information Theory and Coding

IV Probabilities and Inference
20 An Example Inference Task: Clustering
21 Exact Inference by Complete Enumeration
22 Maximum Likelihood and Clustering
23 Useful Probability Distributions
24 Exact Marginalization
25 Exact Marginalization in Trellises
26 Exact Marginalization in Graphs
27 Laplace's Method
28 Model Comparison and Occam's Razor
29 Monte Carlo Methods
30 Efficient Monte Carlo Methods
31 Ising Models
32 Exact Monte Carlo Sampling
33 Variational Methods
34 Independent Component Analysis
35 Random Inference Topics
36 Decision Theory
37 Bayesian Inference and Sampling Theory

V Neural networks
38 Introduction to Neural Networks
39 The Single Neuron as a Classifier
40 Capacity of a Single Neuron
41 Learning as Inference
42 Hopfield Networks
43 Boltzmann Machines
44 Supervised Learning in Multilayer Networks
45 Gaussian Processes
46 Deconvolution

VI Sparse Graph Codes
47 Low-Density Parity-Check Codes
48 Convolutional Codes and Turbo Codes
49 Repeat–Accumulate Codes
50 Digital Fountain Codes



1 Introduction to Information Theory
2 Probability, Entropy, and Inference
3 More about Inference

I Data Compression
4 The Source Coding Theorem
5 Symbol Codes
6 Stream Codes
7 Codes for Integers

II Noisy-Channel Coding
8 Dependent Random Variables
9 Communication over a Noisy Channel
10 The Noisy-Channel Coding Theorem
11 Error-Correcting Codes and Real Channels

III Further Topics in Information Theory
12 Hash Codes
13 Binary Codes
14 Very Good Linear Codes Exist
15 Further Exercises on Information Theory
16 Message Passing
17 Constrained Noiseless Channels
18 Crosswords and Codebreaking
19 Why have Sex?

A Course on Bayesian Inference and Machine Learning

IV Probabilities and Inference
20 An Example Inference Task: Clustering
21 Exact Inference by Complete Enumeration
22 Maximum Likelihood and Clustering
23 Useful Probability Distributions
24 Exact Marginalization
25 Exact Marginalization in Trellises
26 Exact Marginalization in Graphs
27 Laplace's Method
28 Model Comparison and Occam's Razor
29 Monte Carlo Methods
30 Efficient Monte Carlo Methods
31 Ising Models
32 Exact Monte Carlo Sampling
33 Variational Methods
34 Independent Component Analysis
35 Random Inference Topics
36 Decision Theory
37 Bayesian Inference and Sampling Theory

V Neural networks
38 Introduction to Neural Networks
39 The Single Neuron as a Classifier
40 Capacity of a Single Neuron
41 Learning as Inference
42 Hopfield Networks
43 Boltzmann Machines
44 Supervised Learning in Multilayer Networks
45 Gaussian Processes
46 Deconvolution

VI Sparse Graph Codes
47 Low-Density Parity-Check Codes
48 Convolutional Codes and Turbo Codes
49 Repeat–Accumulate Codes
50 Digital Fountain Codes

About the exercises

You can understand a subject only by creating it for yourself. The exercises play an essential role in this book. For guidance, each has a rating (similar to that used by Knuth (1968)) from 1 to 5 to indicate its difficulty.

🐀 In addition, exercises that are especially recommended are marked by a marginal encouraging rat. Some exercises that require the use of a computer are marked with a *C*.

Answers to many exercises are provided. Use them wisely. Where a solution is provided, this is indicated by including its page number alongside the difficulty rating.

Solutions to many of the other exercises will be supplied to instructors using this book in their teaching; please email `solutions@cambridge.org`.

Summary of codes for exercises

🐀	Especially recommended	[*1*]	Simple (one minute)
		[*2*]	Medium (quarter hour)
▷	Recommended	[*3*]	Moderately hard
C	Parts require a computer	[*4*]	Hard
[p. 42]	Solution provided on page 42	[*5*]	Research project

Internet resources

The website

`http://www.inference.phy.cam.ac.uk/mackay/itila`

contains several resources:

1. *Software.* Teaching software that I use in lectures, interactive software, and research software, written in `perl`, `octave`, `tcl`, `C`, and `gnuplot`. Also some animations.

2. *Corrections to the book.* Thank you in advance for emailing these!

3. *This book.* The book is provided in `postscript`, `pdf`, and `djvu` formats for on-screen viewing. The same copyright restrictions apply as to a normal book.

About this edition

This is the fourth printing of the first edition. In the second printing, the design of the book was altered slightly. Page-numbering generally remained unchanged, except in chapters 1, 6, and 28, where a few paragraphs, figures, and equations moved around. All equation, section, and exercise numbers were unchanged. In the third printing, chapter 8 was renamed ‘Dependent Random Variables’, instead of ‘Correlated’, which was sloppy.

Acknowledgments

I am most grateful to the organizations who have supported me while this book gestated: the Royal Society and Darwin College who gave me a fantastic research fellowship in the early years; the University of Cambridge; the Keck Centre at the University of California in San Francisco, where I spent a productive sabbatical; and the Gatsby Charitable Foundation, whose support gave me the freedom to break out of the Escher staircase that book-writing had become.

My work has depended on the generosity of free software authors. I wrote the book in $\LaTeX\,2_\varepsilon$. Three cheers for Donald Knuth and Leslie Lamport! Our computers run the GNU/Linux operating system. I use `emacs`, `perl`, and `gnuplot` every day. Thank you Richard Stallman, thank you Linus Torvalds, thank you everyone.

Many readers, too numerous to name here, have given feedback on the book, and to them all I extend my sincere acknowledgments. I especially wish to thank all the students and colleagues at Cambridge University who have attended my lectures on information theory and machine learning over the last nine years.

The members of the Inference research group have given immense support, and I thank them all for their generosity and patience over the last ten years: Mark Gibbs, Michelle Povinelli, Simon Wilson, Coryn Bailer-Jones, Matthew Davey, Katriona Macphee, James Miskin, David Ward, Edward Ratzer, Seb Wills, John Barry, John Winn, Phil Cowans, Hanna Wallach, Matthew Garrett, and especially Sanjoy Mahajan. Thank you too to Graeme Mitchison, Mike Cates, and Davin Yap.

Finally I would like to express my debt to my personal heroes, the mentors from whom I have learned so much: Yaser Abu-Mostafa, Andrew Blake, John Bridle, Peter Cheeseman, Steve Gull, Geoff Hinton, John Hopfield, Steve Luttrell, Robert MacKay, Bob McEliece, Radford Neal, Roger Sewell, and John Skilling.

Dedication

This book is dedicated to the campaign against the arms trade.

`www.caat.org.uk`

Peace cannot be kept by force.
It can only be achieved through understanding.
– *Albert Einstein*

About Chapter 1

In the first chapter, you will need to be familiar with the binomial distribution. And to solve the exercises in the text – which I urge you to do – you will need to know *Stirling's approximation* for the factorial function, $x! \simeq x^x \, e^{-x}$, and be able to apply it to $\binom{N}{r} = \frac{N!}{(N-r)!\,r!}$. These topics are reviewed below.

Unfamiliar notation?
See Appendix A, p.598.

The binomial distribution

Example 1.1. A bent coin has probability f of coming up heads. The coin is tossed N times. What is the probability distribution of the number of heads, r? What are the mean and variance of r?

Solution. The number of heads has a binomial distribution.

$$P(r \mid f, N) = \binom{N}{r} f^r (1-f)^{N-r}. \tag{1.1}$$

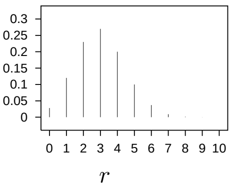


Figure 1.1. The binomial distribution $P(r \mid f = 0.3,\, N = 10)$.

The mean, $\mathcal{E}[r]$, and variance, $\mathrm{var}[r]$, of this distribution are defined by

$$\mathcal{E}[r] \equiv \sum_{r=0}^{N} P(r \mid f, N)\, r \tag{1.2}$$

$$\begin{aligned} \mathrm{var}[r] &\equiv \mathcal{E}\left[(r - \mathcal{E}[r])^2\right] && (1.3)\\ &= \mathcal{E}[r^2] - (\mathcal{E}[r])^2 = \sum_{r=0}^{N} P(r \mid f, N) r^2 - (\mathcal{E}[r])^2 . && (1.4) \end{aligned}$$

Rather than evaluating the sums over r in (1.2) and (1.4) directly, it is easiest to obtain the mean and variance by noting that r is the sum of N *independent* random variables, namely, the number of heads in the first toss (which is either zero or one), the number of heads in the second toss, and so forth. In general,

$$\begin{array}{rcll} \mathcal{E}[x+y] & = & \mathcal{E}[x] + \mathcal{E}[y] & \text{for any random variables } x \text{ and } y; \\ \mathrm{var}[x+y] & = & \mathrm{var}[x] + \mathrm{var}[y] & \text{if } x \text{ and } y \text{ are independent.} \end{array} \tag{1.5}$$

So the mean of r is the sum of the means of those random variables, and the variance of r is the sum of their variances. The mean number of heads in a single toss is $f \times 1 + (1-f) \times 0 = f$, and the variance of the number of heads in a single toss is

$$\left[f \times 1^2 + (1-f) \times 0^2\right] - f^2 = f - f^2 = f(1-f), \tag{1.6}$$

so the mean and variance of r are:

$$\mathcal{E}[r] = Nf \qquad \text{and} \qquad \mathrm{var}[r] = Nf(1-f). \qquad \Box \tag{1.7}$$

Approximating $x!$ and $\binom{N}{r}$

Let's derive Stirling's approximation by an unconventional route. We start from the Poisson distribution with mean λ,

$$P(r\,|\,\lambda) = e^{-\lambda}\frac{\lambda^r}{r!} \qquad r \in \{0,1,2,\ldots\}. \tag{1.8}$$

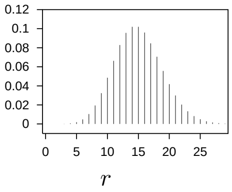


Figure 1.2. The Poisson distribution $P(r\,|\,\lambda\!=\!15)$.

For large λ, this distribution is well approximated – at least in the vicinity of $r \simeq \lambda$ – by a Gaussian distribution with mean λ and variance λ:

$$e^{-\lambda}\frac{\lambda^r}{r!} \simeq \frac{1}{\sqrt{2\pi\lambda}}\,e^{-\frac{(r-\lambda)^2}{2\lambda}}. \tag{1.9}$$

Let's plug $r = \lambda$ into this formula, then rearrange it.

$$e^{-\lambda}\frac{\lambda^\lambda}{\lambda!} \;\simeq\; \frac{1}{\sqrt{2\pi\lambda}} \tag{1.10}$$

$$\Rightarrow\quad \lambda! \;\simeq\; \lambda^\lambda\,e^{-\lambda}\sqrt{2\pi\lambda}. \tag{1.11}$$

This is Stirling's approximation for the factorial function.

$$x! \;\simeq\; x^x\,e^{-x}\sqrt{2\pi x} \quad\Leftrightarrow\quad \ln x! \;\simeq\; x\ln x - x + \tfrac{1}{2}\ln 2\pi x. \tag{1.12}$$

We have derived not only the leading order behaviour, $x! \simeq x^x\,e^{-x}$, but also, at no cost, the next-order correction term $\sqrt{2\pi x}$. We now apply Stirling's approximation to $\ln\binom{N}{r}$:

$$\ln\binom{N}{r} \;\equiv\; \ln\frac{N!}{(N-r)!\,r!} \;\simeq\; (N-r)\ln\frac{N}{N-r} + r\ln\frac{N}{r}. \tag{1.13}$$

Since all the terms in this equation are logarithms, this result can be rewritten in any base. We will denote natural logarithms ($\log_e$) by 'ln', and logarithms to base 2 ($\log_2$) by 'log'.

Recall that $\log_2 x = \dfrac{\log_e x}{\log_e 2}$.

Note that $\dfrac{\partial \log_2 x}{\partial x} = \dfrac{1}{\log_e 2}\,\dfrac{1}{x}$.

If we introduce the *binary entropy function*,

$$H_2(x) \equiv x\log\frac{1}{x} + (1-x)\log\frac{1}{(1-x)}, \tag{1.14}$$

then we can rewrite the approximation (1.13) as

$$\log\binom{N}{r} \;\simeq\; NH_2(r/N), \tag{1.15}$$

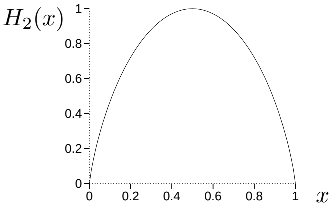


Figure 1.3. The binary entropy function.

or, equivalently,

$$\binom{N}{r} \;\simeq\; 2^{NH_2(r/N)}. \tag{1.16}$$

If we need a more accurate approximation, we can include terms of the next order from Stirling's approximation (1.12):

$$\log\binom{N}{r} \;\simeq\; NH_2(r/N) - \tfrac{1}{2}\log\left[2\pi N\,\frac{N-r}{N}\,\frac{r}{N}\right]. \tag{1.17}$$

1

Introduction to Information Theory

> The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point.
>
> *(Claude Shannon, 1948)*

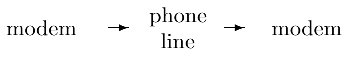
In the first half of this book we study how to measure information content; we learn how to compress data; and we learn how to communicate perfectly over imperfect communication channels.

We start by getting a feeling for this last problem.

1.1 How can we achieve perfect communication over an imperfect, noisy communication channel?

Some examples of noisy communication channels are:

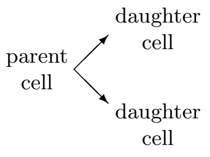
- an analogue telephone line, over which two modems communicate digital information;



- the radio communication link from Galileo, the Jupiter-orbiting spacecraft, to earth;

Galileo
radio waves
Earth

- reproducing cells, in which the daughter cells' DNA contains information from the parent cells;



- a disk drive.

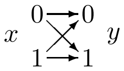
computer memory
disk drive
computer memory

The last example shows that communication doesn't have to involve information going from one *place* to another. When we write a file on a disk drive, we'll read it off in the same location – but at a later *time*.

These channels are noisy. A telephone line suffers from cross-talk with other lines; the hardware in the line distorts and adds noise to the transmitted signal. The deep space network that listens to Galileo's puny transmitter receives background radiation from terrestrial and cosmic sources. DNA is subject to mutations and damage. A disk drive, which writes a binary digit (a one or zero, also known as a *bit*) by aligning a patch of magnetic material in one of two orientations, may later fail to read out the stored binary digit: the patch of material might spontaneously flip magnetization, or a glitch of background noise might cause the reading circuit to report the wrong value for the binary digit, or the writing head might not induce the magnetization in the first place because of interference from neighbouring bits.

In all these cases, if we transmit data, e.g., a string of bits, over the channel, there is some probability that the received message will not be identical to the transmitted message. We would prefer to have a communication channel for which this probability was zero – or so close to zero that for practical purposes it is indistinguishable from zero.

Let's consider a noisy disk drive that transmits each bit correctly with probability $(1-f)$ and incorrectly with probability f. This model communication channel is known as the *binary symmetric channel* (figure 1.4).



$$\begin{array}{lcl lcl} P(y\!=\!0\,|\,x\!=\!0) &=& 1-f; & P(y\!=\!0\,|\,x\!=\!1) &=& f; \\ P(y\!=\!1\,|\,x\!=\!0) &=& f; & P(y\!=\!1\,|\,x\!=\!1) &=& 1-f. \end{array}$$

Figure 1.4. The binary symmetric channel. The transmitted symbol is x and the received symbol y. The noise level, the probability that a bit is flipped, is f.

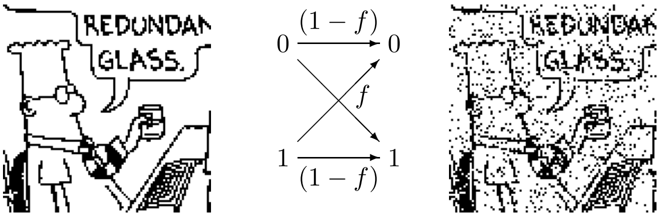


Figure 1.5. A binary data sequence of length 10 000 transmitted over a binary symmetric channel with noise level $f = 0.1$. [Dilbert image Copyright©1997 United Feature Syndicate, Inc., used with permission.]

As an example, let's imagine that $f = 0.1$, that is, ten per cent of the bits are flipped (figure 1.5). A useful disk drive would flip no bits at all in its entire lifetime. If we expect to read and write a gigabyte per day for ten years, we require a bit error probability of the order of 10^{-15}, or smaller. There are two approaches to this goal.

The physical solution

The physical solution is to improve the physical characteristics of the communication channel to reduce its error probability. We could improve our disk drive by

1. using more reliable components in its circuitry;
2. evacuating the air from the disk enclosure so as to eliminate the turbulence that perturbs the reading head from the track;
3. using a larger magnetic patch to represent each bit; or
4. using higher-power signals or cooling the circuitry in order to reduce thermal noise.

These physical modifications typically increase the cost of the communication channel.

The 'system' solution

Information theory and coding theory offer an alternative (and much more exciting) approach: we accept the given noisy channel as it is and add communication *systems* to it so that we can detect and correct the errors introduced by the channel. As shown in figure 1.6, we add an *encoder* before the channel and a *decoder* after it. The encoder encodes the source message $\mathbf{s}$ into a *transmitted* message $\mathbf{t}$, adding *redundancy* to the original message in some way. The channel adds noise to the transmitted message, yielding a received message $\mathbf{r}$. The decoder uses the known redundancy introduced by the encoding system to infer both the original signal $\mathbf{s}$ and the added noise.

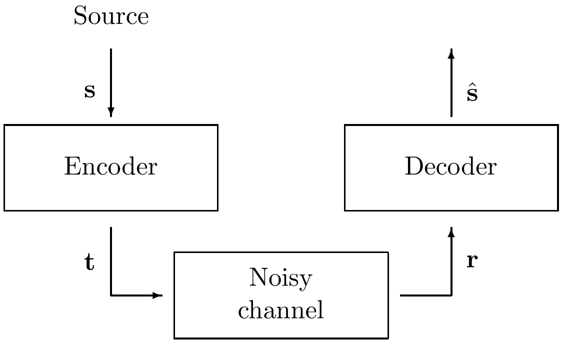


Figure 1.6. The 'system' solution for achieving reliable communication over a noisy channel. The encoding system introduces systematic redundancy into the transmitted vector $\mathbf{t}$. The decoding system uses this known redundancy to deduce from the received vector $\mathbf{r}$ *both* the original source vector *and* the noise introduced by the channel.

Whereas physical solutions give incremental channel improvements only at an ever-increasing cost, system solutions can turn noisy channels into reliable communication channels with the only cost being a *computational* requirement at the encoder and decoder.

Information theory is concerned with the theoretical limitations and potentials of such systems. 'What is the best error-correcting performance we could achieve?'

Coding theory is concerned with the creation of practical encoding and decoding systems.

▶ 1.2 Error-correcting codes for the binary symmetric channel

We now consider examples of encoding and decoding systems. What is the simplest way to add useful redundancy to a transmission? [To make the rules of the game clear: we want to be able to detect *and* correct errors; and retransmission is not an option. We get only one chance to encode, transmit, and decode.]

Repetition codes

A straightforward idea is to repeat every bit of the message a prearranged number of times – for example, three times, as shown in table 1.7. We call this *repetition code* 'R_3'.

Source sequence $\mathbf{s}$	Transmitted sequence $\mathbf{t}$
0	000
1	111

Table 1.7. The repetition code R_3.

Imagine that we transmit the source message

$$\mathbf{s} = 0\ 0\ 1\ 0\ 1\ 1\ 0$$

over a binary symmetric channel with noise level $f = 0.1$ using this repetition code. We can describe the channel as 'adding' a sparse noise vector $\mathbf{n}$ to the transmitted vector – adding in modulo 2 arithmetic, i.e., the binary algebra in which 1+1=0. A possible noise vector $\mathbf{n}$ and received vector $\mathbf{r} = \mathbf{t} + \mathbf{n}$ are shown in figure 1.8.

$$\begin{array}{cccccccc}
\mathbf{s} & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
\mathbf{t} & \overbrace{0\,0\,0} & \overbrace{0\,0\,0} & \overbrace{1\,1\,1} & \overbrace{0\,0\,0} & \overbrace{1\,1\,1} & \overbrace{1\,1\,1} & \overbrace{0\,0\,0} \\
\mathbf{n} & 0\,0\,0 & 0\,0\,1 & 0\,0\,0 & 0\,0\,0 & 1\,0\,1 & 0\,0\,0 & 0\,0\,0 \\
\hline
\mathbf{r} & 0\,0\,0 & 0\,0\,1 & 1\,1\,1 & 0\,0\,0 & 0\,1\,0 & 1\,1\,1 & 0\,0\,0
\end{array}$$

Figure 1.8. An example transmission using R_3.

How should we decode this received vector? The optimal algorithm looks at the received bits three at a time and takes a majority vote (algorithm 1.9).

Received sequence $\mathbf{r}$	Likelihood ratio $\frac{P(\mathbf{r}\,|\,s=1)}{P(\mathbf{r}\,|\,s=0)}$	Decoded sequence $\hat{\mathbf{s}}$
000	γ^{-3}	0
001	γ^{-1}	0
010	γ^{-1}	0
100	γ^{-1}	0
101	γ^{1}	1
110	γ^{1}	1
011	γ^{1}	1
111	γ^{3}	1

Algorithm 1.9. Majority-vote decoding algorithm for R_3. Also shown are the likelihood ratios (1.23), assuming the channel is a binary symmetric channel; $\gamma \equiv (1-f)/f$.

At the risk of explaining the obvious, let's prove this result. The optimal decoding decision (optimal in the sense of having the smallest probability of being wrong) is to find which value of $\mathbf{s}$ is most probable, given $\mathbf{r}$. Consider the decoding of a single bit s, which was encoded as $\mathbf{t}(s)$ and gave rise to three received bits $\mathbf{r} = r_1r_2r_3$. By Bayes' theorem, the *posterior probability* of s is

$$P(s\,|\,r_1r_2r_3) = \frac{P(r_1r_2r_3\,|\,s)P(s)}{P(r_1r_2r_3)}. \tag{1.18}$$

We can spell out the posterior probability of the two alternatives thus:

$$P(s=1\,|\,r_1r_2r_3) = \frac{P(r_1r_2r_3\,|\,s=1)P(s=1)}{P(r_1r_2r_3)}; \tag{1.19}$$

$$P(s=0\,|\,r_1r_2r_3) = \frac{P(r_1r_2r_3\,|\,s=0)P(s=0)}{P(r_1r_2r_3)}. \tag{1.20}$$

This posterior probability is determined by two factors: the *prior probability* $P(s)$, and the data-dependent term $P(r_1r_2r_3\,|\,s)$, which is called the *likelihood* of s. The normalizing constant $P(r_1r_2r_3)$ needn't be computed when finding the optimal decoding decision, which is to guess $\hat{s}=0$ if $P(s=0\,|\,\mathbf{r}) > P(s=1\,|\,\mathbf{r})$, and $\hat{s}=1$ otherwise.

To find $P(s=0\,|\,\mathbf{r})$ and $P(s=1\,|\,\mathbf{r})$, we must make an assumption about the prior probabilities of the two hypotheses $s=0$ and $s=1$, and we must make an assumption about the probability of $\mathbf{r}$ given s. We assume that the prior probabilities are equal: $P(s=0) = P(s=1) = 0.5$; then maximizing the posterior probability $P(s\,|\,\mathbf{r})$ is equivalent to maximizing the likelihood $P(\mathbf{r}\,|\,s)$. And we assume that the channel is a binary symmetric channel with noise level $f < 0.5$, so that the likelihood is

$$P(\mathbf{r}\,|\,s) = P(\mathbf{r}\,|\,\mathbf{t}(s)) = \prod_{n=1}^{N} P(r_n\,|\,t_n(s)), \tag{1.21}$$

where $N = 3$ is the number of transmitted bits in the block we are considering, and

$$P(r_n\,|\,t_n) = \begin{cases} (1-f) & \text{if} \quad r_n = t_n \\ f & \text{if} \quad r_n \neq t_n. \end{cases} \tag{1.22}$$

Thus the likelihood ratio for the two hypotheses is

$$\frac{P(\mathbf{r}\,|\,s=1)}{P(\mathbf{r}\,|\,s=0)} = \prod_{n=1}^{N} \frac{P(r_n\,|\,t_n(1))}{P(r_n\,|\,t_n(0))}; \tag{1.23}$$

each factor $\frac{P(r_n|t_n(1))}{P(r_n|t_n(0))}$ equals $\frac{(1-f)}{f}$ if $r_n = 1$ and $\frac{f}{(1-f)}$ if $r_n = 0$. The ratio $\gamma \equiv \frac{(1-f)}{f}$ is greater than 1, since $f < 0.5$, so the winning hypothesis is the one with the most 'votes', each vote counting for a factor of γ in the likelihood ratio.

> Thus the majority-vote decoder shown in algorithm 1.9 is the optimal decoder if we assume that the channel is a binary symmetric channel and that the two possible source messages `0` and `1` have equal prior probability.

We now apply the majority vote decoder to the received vector of figure 1.8. The first three received bits are all `0`, so we decode this triplet as a `0`. In the second triplet of figure 1.8, there are two `0`s and one `1`, so we decode this triplet as a `0` – which in this case corrects the error. Not all errors are corrected, however. If we are unlucky and two errors fall in a single block, as in the fifth triplet of figure 1.8, then the decoding rule gets the wrong answer, as shown in figure 1.10.

$$\begin{array}{rccccccc}
\mathbf{s} & \texttt{0} & \texttt{0} & \texttt{1} & \texttt{0} & \texttt{1} & \texttt{1} & \texttt{0}\\
\mathbf{t} & \overbrace{\texttt{000}} & \overbrace{\texttt{000}} & \overbrace{\texttt{111}} & \overbrace{\texttt{000}} & \overbrace{\texttt{111}} & \overbrace{\texttt{111}} & \overbrace{\texttt{000}}\\
\mathbf{n} & \texttt{000} & \texttt{001} & \texttt{000} & \texttt{000} & \texttt{101} & \texttt{000} & \texttt{000}\\
\hline
\mathbf{r} & \underbrace{\texttt{000}} & \underbrace{\texttt{001}} & \underbrace{\texttt{111}} & \underbrace{\texttt{000}} & \underbrace{\texttt{010}} & \underbrace{\texttt{111}} & \underbrace{\texttt{000}}\\
\hat{\mathbf{s}} & \texttt{0} & \texttt{0} & \texttt{1} & \texttt{0} & \texttt{0} & \texttt{1} & \texttt{0}\\
\text{corrected errors} & & \star & & & & & \\
\text{undetected errors} & & & & & \star & &
\end{array}$$

Figure 1.10. Decoding the received vector from figure 1.8.

Exercise 1.2.[2, p.16] Show that the error probability is reduced by the use of R_3 by computing the error probability of this code for a binary symmetric channel with noise level f.

The exercise's rating, e.g.'[2]', indicates its difficulty: '1' exercises are the easiest. Exercises that are accompanied by a marginal rat are especially recommended. If a solution or partial solution is provided, the page is indicated after the difficulty rating; for example, this exercise's solution is on page 16.

The error probability is dominated by the probability that two bits in a block of three are flipped, which scales as f^2. In the case of the binary symmetric channel with $f = 0.1$, the R_3 code has a probability of error, after decoding, of $p_b \simeq 0.03$ per bit. Figure 1.11 shows the result of transmitting a binary image over a binary symmetric channel using the repetition code.

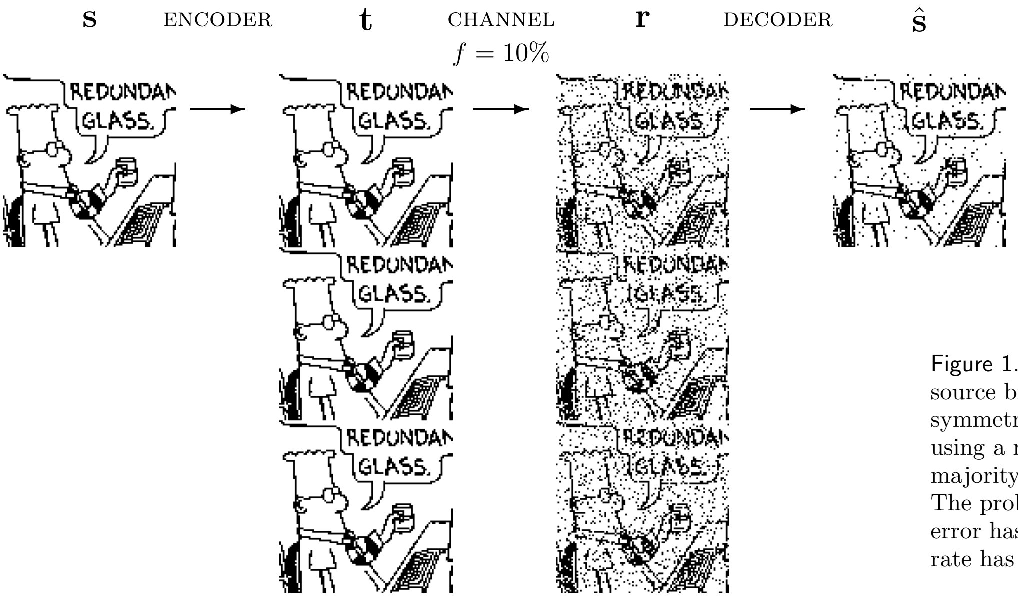


Figure 1.11. Transmitting 10 000 source bits over a binary symmetric channel with $f = 10\%$ using a repetition code and the majority vote decoding algorithm. The probability of decoded bit error has fallen to about 3%; the rate has fallen to 1/3.

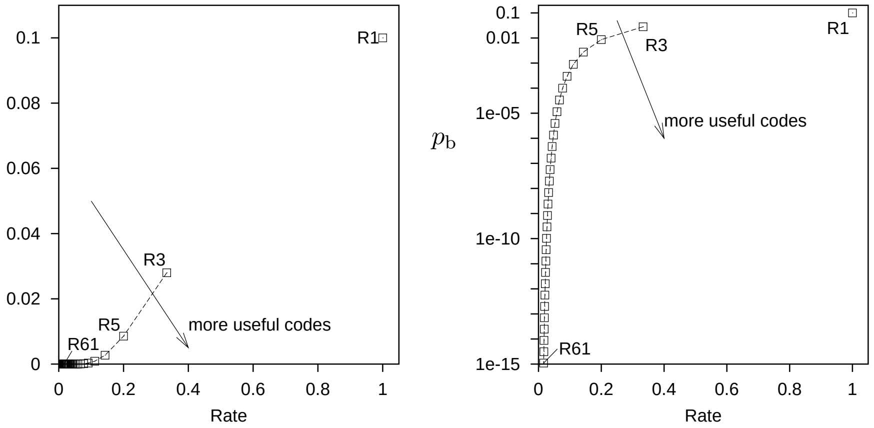


Figure 1.12. Error probability p_{b} versus rate for repetition codes over a binary symmetric channel with $f = 0.1$. The right-hand figure shows p_{b} on a logarithmic scale. We would like the rate to be large and p_{b} to be small.

The repetition code R_3 has therefore reduced the probability of error, as desired. Yet we have lost something: our *rate* of information transfer has fallen by a factor of three. So if we use a repetition code to communicate data over a telephone line, it will reduce the error frequency, but it will also reduce our communication rate. We will have to pay three times as much for each phone call. Similarly, we would need three of the original noisy gigabyte disk drives in order to create a one-gigabyte disk drive with $p_{\mathrm{b}} = 0.03$.

Can we push the error probability lower, to the values required for a sellable disk drive – 10^{-15}? We could achieve lower error probabilities by using repetition codes with more repetitions.

Exercise 1.3.[*3*, p.16] (a) Show that the probability of error of R_N, the repetition code with N repetitions, is

$$p_{\mathrm{b}} = \sum_{n=(N+1)/2}^{N} \binom{N}{n} f^n (1-f)^{N-n}, \tag{1.24}$$

for odd N.

(b) Assuming $f = 0.1$, which of the terms in this sum is the biggest? How much bigger is it than the second-biggest term?

(c) Use Stirling's approximation (p.2) to approximate the $\binom{N}{n}$ in the largest term, and find, approximately, the probability of error of the repetition code with N repetitions.

(d) Assuming $f = 0.1$, find how many repetitions are required to get the probability of error down to 10^{-15}. [Answer: about 60.]

So to build a *single* gigabyte disk drive with the required reliability from noisy gigabyte drives with $f = 0.1$, we would need *sixty* of the noisy disk drives. The tradeoff between error probability and rate for repetition codes is shown in figure 1.12.

Block codes – the $(7, 4)$ *Hamming code*

We would like to communicate with tiny probability of error *and* at a substantial rate. Can we improve on repetition codes? What if we add redundancy to *blocks* of data instead of encoding one bit at a time? We now study a simple *block code.*

A *block code* is a rule for converting a sequence of source bits $\mathbf{s}$, of length K, say, into a transmitted sequence $\mathbf{t}$ of length N bits. To add redundancy, we make N greater than K. In a *linear* block code, the extra $N - K$ bits are linear functions of the original K bits; these extra bits are called *parity-check bits*. An example of a linear block code is the $(7, 4)$ *Hamming code*, which transmits $N = 7$ bits for every $K = 4$ source bits.

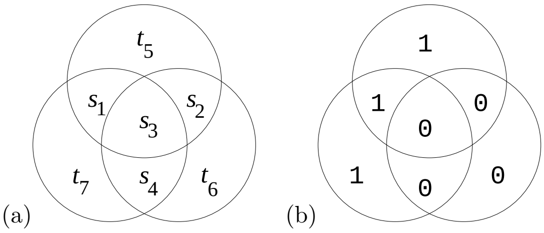


Figure 1.13. Pictorial representation of encoding for the $(7, 4)$ Hamming code.

The encoding operation for the code is shown pictorially in figure 1.13. We arrange the seven transmitted bits in three intersecting circles. The first four transmitted bits, $t_1t_2t_3t_4$, are set equal to the four source bits, $s_1s_2s_3s_4$. The parity-check bits $t_5t_6t_7$ are set so that the *parity* within each circle is even: the first parity-check bit is the parity of the first three source bits (that is, it is 0 if the sum of those bits is even, and 1 if the sum is odd); the second is the parity of the last three; and the third parity bit is the parity of source bits one, three and four.

As an example, figure 1.13b shows the transmitted codeword for the case $\mathbf{s} =$ 1000. Table 1.14 shows the codewords generated by each of the $2^4 =$ sixteen settings of the four source bits. These codewords have the special property that any pair differ from each other in at least three bits.

s	t	s	t	s	t	s	t
0000	0000000	0100	0100110	1000	1000101	1100	1100011
0001	0001011	0101	0101101	1001	1001110	1101	1101000
0010	0010111	0110	0110001	1010	1010010	1110	1110100
0011	0011100	0111	0111010	1011	1011001	1111	1111111

Table 1.14. The sixteen codewords $\{\mathbf{t}\}$ of the $(7, 4)$ Hamming code. Any pair of codewords differ from each other in at least three bits.

Because the Hamming code is a linear code, it can be written compactly in terms of matrices as follows. The transmitted codeword $\mathbf{t}$ is obtained from the source sequence $\mathbf{s}$ by a linear operation,

$$\mathbf{t} = \mathbf{G}^{\mathsf{T}}\mathbf{s}, \tag{1.25}$$

where $\mathbf{G}$ is the *generator matrix* of the code,

$$\mathbf{G}^{\mathsf{T}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}, \tag{1.26}$$

and the encoding operation (1.25) uses modulo-2 arithmetic ($1 + 1 = 0$, $0 + 1 = 1$, etc.).

In the encoding operation (1.25) I have assumed that $\mathbf{s}$ and $\mathbf{t}$ are column vectors. If instead they are row vectors, then this equation is replaced by

$$\mathbf{t} = \mathbf{s}\mathbf{G}, \tag{1.27}$$

where

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}. \tag{1.28}$$

I find it easier to relate to the right-multiplication (1.25) than the left-multiplication (1.27). Many coding theory texts use the left-multiplying conventions (1.27–1.28), however.

The rows of the generator matrix (1.28) can be viewed as defining four basis vectors lying in a seven-dimensional binary space. The sixteen codewords are obtained by making all possible linear combinations of these vectors.

Decoding the $(7,4)$ *Hamming code*

When we invent a more complex encoder $\mathbf{s} \rightarrow \mathbf{t}$, the task of decoding the received vector $\mathbf{r}$ becomes less straightforward. Remember that *any* of the bits may have been flipped, including the parity bits.

If we assume that the channel is a binary symmetric channel and that all source vectors are equiprobable, then the optimal decoder identifies the source vector $\mathbf{s}$ whose encoding $\mathbf{t}(\mathbf{s})$ differs from the received vector $\mathbf{r}$ in the fewest bits. [Refer to the likelihood function (1.23) to see why this is so.] We could solve the decoding problem by measuring how far $\mathbf{r}$ is from each of the sixteen codewords in table 1.14, then picking the closest. Is there a more efficient way of finding the most probable source vector?

Syndrome decoding for the Hamming code

For the $(7,4)$ Hamming code there is a pictorial solution to the decoding problem, based on the encoding picture, figure 1.13.

As a first example, let's assume the transmission was $\mathbf{t} = \mathtt{1000101}$ and the noise flips the second bit, so the received vector is $\mathbf{r} = \mathtt{1000101} \oplus \mathtt{0100000} = \mathtt{1100101}$. We write the received vector into the three circles as shown in figure 1.15a, and look at each of the three circles to see whether its parity is even. The circles whose parity is *not* even are shown by dashed lines in figure 1.15b. The decoding task is to find the smallest set of flipped bits that can account for these violations of the parity rules. [The pattern of violations of the parity checks is called the *syndrome*, and can be written as a binary vector – for example, in figure 1.15b, the syndrome is $\mathbf{z} = (1,1,0)$, because the first two circles are 'unhappy' (parity 1) and the third circle is 'happy' (parity 0).]

To solve the decoding task, we ask the question: can we find a unique bit that lies *inside* all the 'unhappy' circles and *outside* all the 'happy' circles? If so, the flipping of that bit would account for the observed syndrome. In the case shown in figure 1.15b, the bit r_2 lies inside the two unhappy circles and outside the happy circle; no other single bit has this property, so r_2 is the only single bit capable of explaining the syndrome.

Let's work through a couple more examples. Figure 1.15c shows what happens if one of the parity bits, t_5, is flipped by the noise. Just one of the checks is violated. Only r_5 lies inside this unhappy circle and outside the other two happy circles, so r_5 is identified as the only single bit capable of explaining the syndrome.

If the central bit r_3 is received flipped, figure 1.15d shows that all three checks are violated; only r_3 lies inside all three circles, so r_3 is identified as the suspect bit.

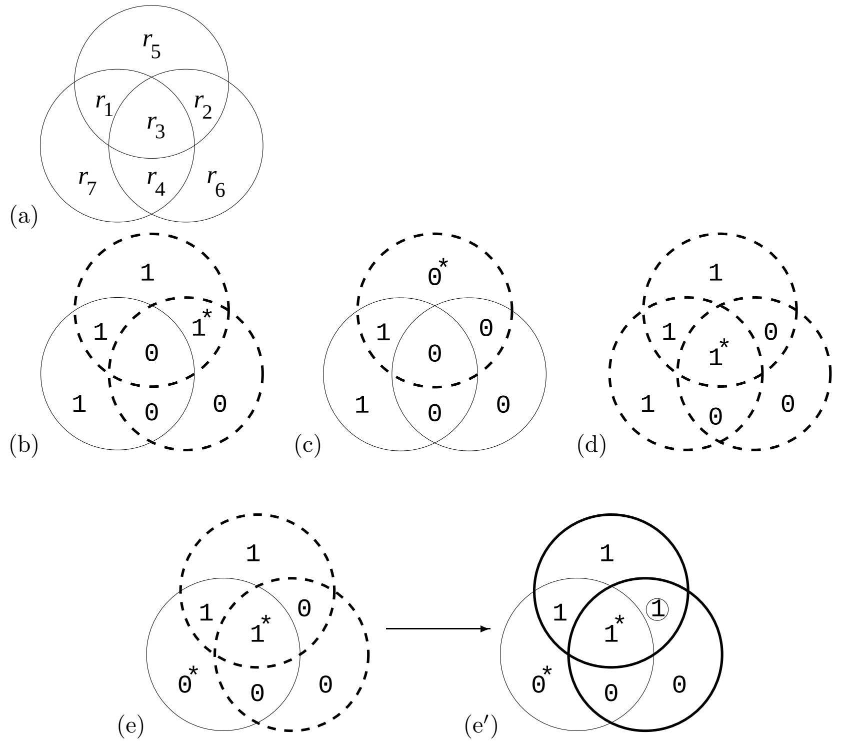


Figure 1.15. Pictorial representation of decoding of the Hamming $(7, 4)$ code. The received vector is written into the diagram as shown in (a). In (b,c,d,e), the received vector is shown, assuming that the transmitted vector was as in figure 1.13b and the bits labelled by $\star$ were flipped. The violated parity checks are highlighted by dashed circles. One of the seven bits is the most probable suspect to account for each 'syndrome', i.e., each pattern of violated and satisfied parity checks.
In examples (b), (c), and (d), the most probable suspect is the one bit that was flipped.
In example (e), two bits have been flipped, s_3 and t_7. The most probable suspect is r_2, marked by a circle in (e′), which shows the output of the decoding algorithm.

Syndrome $\mathbf{z}$	000	001	010	011	100	101	110	111
Unflip this bit	*none*	r_7	r_6	r_4	r_5	r_1	r_2	r_3

Algorithm 1.16. Actions taken by the optimal decoder for the $(7, 4)$ Hamming code, assuming a binary symmetric channel with small noise level f. The syndrome vector $\mathbf{z}$ lists whether each parity check is violated (1) or satisfied (0), going through the checks in the order of the bits r_5, r_6, and r_7.

If you try flipping any one of the seven bits, you'll find that a different syndrome is obtained in each case – seven non-zero syndromes, one for each bit. There is only one other syndrome, the all-zero syndrome. So if the channel is a binary symmetric channel with a small noise level f, the optimal decoder unflips at most one bit, depending on the syndrome, as shown in algorithm 1.16. Each syndrome could have been caused by other noise patterns too, but any other noise pattern that has the same syndrome must be less probable because it involves a larger number of noise events.

What happens if the noise actually flips more than one bit? Figure 1.15e shows the situation when two bits, r_3 and r_7, are received flipped. The syndrome, 110, makes us suspect the single bit r_2; so our optimal decoding algorithm flips this bit, giving a decoded pattern with three errors as shown in figure 1.15e′. If we use the optimal decoding algorithm, any two-bit error pattern will lead to a decoded seven-bit vector that contains three errors.

General view of decoding for linear codes: syndrome decoding

We can also describe the decoding problem for a linear code in terms of matrices. The first four received bits, $r_1r_2r_3r_4$, purport to be the four source bits; and the received bits $r_5r_6r_7$ purport to be the parities of the source bits, as defined by the generator matrix $\mathbf{G}$. We evaluate the three parity-check bits for the received bits, $r_1r_2r_3r_4$, and see whether they match the three received bits, $r_5r_6r_7$. The differences (modulo 2) between these two triplets are called the *syndrome* of the received vector. If the syndrome is zero – if all three parity checks are happy – then the received vector is a codeword, and the most probable decoding is

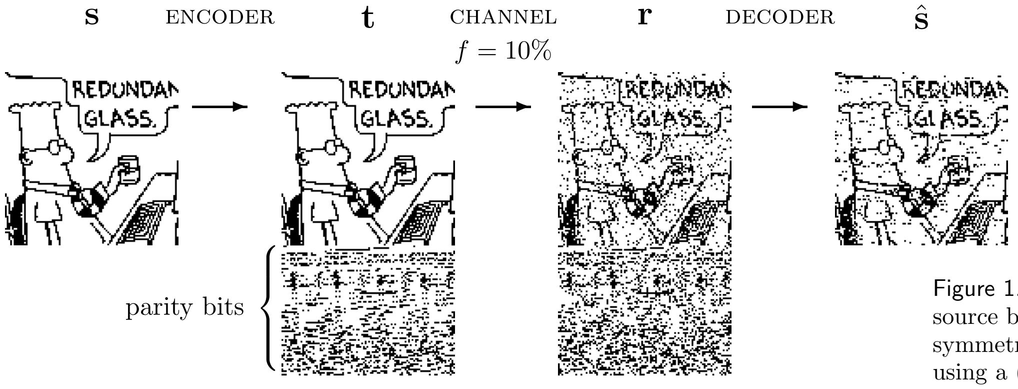


Figure 1.17. Transmitting 10 000 source bits over a binary symmetric channel with $f = 10\%$ using a $(7, 4)$ Hamming code. The probability of decoded bit error is about 7%.

given by reading out its first four bits. If the syndrome is non-zero, then the noise sequence for this block was non-zero, and the syndrome is our pointer to the most probable error pattern.

The computation of the syndrome vector is a linear operation. If we define the 3×4 matrix $\mathbf{P}$ such that the matrix of equation (1.26) is

$$\mathbf{G}^{\mathsf{T}} = \begin{bmatrix} \mathbf{I}_4 \\ \mathbf{P} \end{bmatrix}, \tag{1.29}$$

where $\mathbf{I}_4$ is the 4×4 identity matrix, then the syndrome vector is $\mathbf{z} = \mathbf{H}\mathbf{r}$, where the *parity-check matrix* $\mathbf{H}$ is given by $\mathbf{H} = \begin{bmatrix} -\mathbf{P} & \mathbf{I}_3 \end{bmatrix}$; in modulo 2 arithmetic, $-1 \equiv 1$, so

$$\mathbf{H} = \begin{bmatrix} \mathbf{P} & \mathbf{I}_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{1.30}$$

All the codewords $\mathbf{t} = \mathbf{G}^{\mathsf{T}}\mathbf{s}$ of the code satisfy

$$\mathbf{H}\mathbf{t} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \tag{1.31}$$

▷ Exercise 1.4.$^{[1]}$ Prove that this is so by evaluating the 3×4 matrix $\mathbf{H}\mathbf{G}^{\mathsf{T}}$.

Since the received vector $\mathbf{r}$ is given by $\mathbf{r} = \mathbf{G}^{\mathsf{T}}\mathbf{s} + \mathbf{n}$, the syndrome-decoding problem is to find the most probable noise vector $\mathbf{n}$ satisfying the equation

$$\mathbf{H}\mathbf{n} = \mathbf{z}. \tag{1.32}$$

A decoding algorithm that solves this problem is called a *maximum-likelihood decoder*. We will discuss decoding problems like this in later chapters.

Summary of the $(7, 4)$ *Hamming code's properties*

Every possible received vector of length 7 bits is either a codeword, or it's one flip away from a codeword.

Since there are three parity constraints, each of which might or might not be violated, there are $2 \times 2 \times 2 = 8$ distinct syndromes. They can be divided into seven non-zero syndromes – one for each of the one-bit error patterns – and the all-zero syndrome, corresponding to the zero-noise case.

The optimal decoder takes no action if the syndrome is zero, otherwise it uses this mapping of non-zero syndromes onto one-bit error patterns to unflip the suspect bit.

There is a *decoding error* if the four decoded bits $\hat{s}_1, \hat{s}_2, \hat{s}_3, \hat{s}_4$ do not all match the source bits s_1, s_2, s_3, s_4. The *probability of block error* $p_{\rm B}$ is the probability that one or more of the decoded bits in one block fail to match the corresponding source bits,

$$p_{\rm B} = P(\hat{\mathbf{s}} \neq \mathbf{s}). \tag{1.33}$$

The *probability of bit error* $p_{\rm b}$ is the average probability that a decoded bit fails to match the corresponding source bit,

$$p_{\rm b} = \frac{1}{K} \sum_{k=1}^{K} P(\hat{s}_k \neq s_k). \tag{1.34}$$

In the case of the Hamming code, a decoding error will occur whenever the noise has flipped more than one bit in a block of seven. The probability of block error is thus the probability that two or more bits are flipped in a block. This probability scales as $O(f^2)$, as did the probability of error for the repetition code $\rm R_3$. But notice that the Hamming code communicates at a greater rate, $R = 4/7$.

Figure 1.17 shows a binary image transmitted over a binary symmetric channel using the $(7, 4)$ Hamming code. About 7% of the decoded bits are in error. Notice that the errors are correlated: often two or three successive decoded bits are flipped.

Exercise 1.5.[1] This exercise and the next three refer to the $(7, 4)$ Hamming code. Decode the received strings:

(a) $\mathbf{r} =$ `1101011`

(b) $\mathbf{r} =$ `0110110`

(c) $\mathbf{r} =$ `0100111`

(d) $\mathbf{r} =$ `1111111`.

Exercise 1.6.[2, p.17] (a) Calculate the probability of block error $p_{\rm B}$ of the $(7, 4)$ Hamming code as a function of the noise level f and show that to leading order it goes as $21f^2$.

(b) [3] Show that to leading order the probability of bit error $p_{\rm b}$ goes as $9f^2$.

Exercise 1.7.[2, p.19] Find some noise vectors that give the all-zero syndrome (that is, noise vectors that leave all the parity checks unviolated). How many such noise vectors are there?

▷ Exercise 1.8.[2] I asserted above that a block decoding error will result whenever two or more bits are flipped in a single block. Show that this is indeed so. [In principle, there might be error patterns that, after decoding, led only to the corruption of the parity bits, with no source bits incorrectly decoded.]

Summary of codes' performances

Figure 1.18 shows the performance of repetition codes and the Hamming code. It also shows the performance of a family of linear block codes that are generalizations of Hamming codes, called BCH codes.

This figure shows that we can, using linear block codes, achieve better performance than repetition codes; but the asymptotic situation still looks grim.

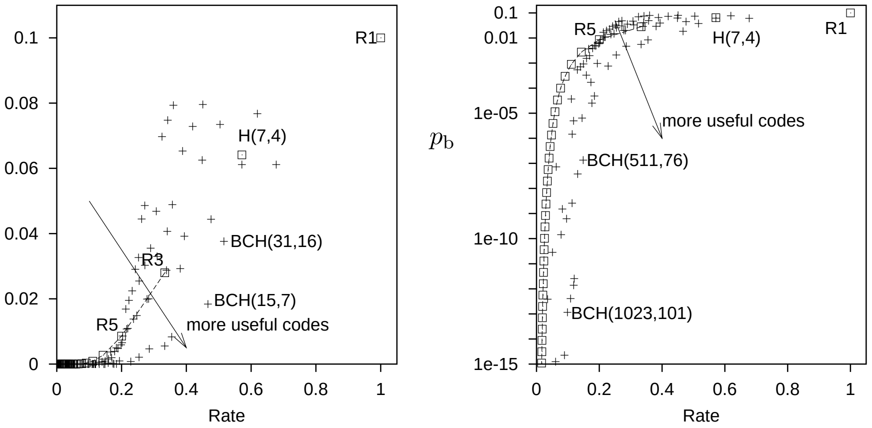


Figure 1.18. Error probability p_b versus rate R for repetition codes, the $(7, 4)$ Hamming code and BCH codes with blocklengths up to 1023 over a binary symmetric channel with $f = 0.1$. The righthand figure shows p_b on a logarithmic scale.

Exercise 1.9.[*4*, p.19] Design an error-correcting code and a decoding algorithm for it, estimate its probability of error, and add it to figure 1.18. [Don't worry if you find it difficult to make a code better than the Hamming code, or if you find it difficult to find a good decoder for your code; that's the point of this exercise.]

Exercise 1.10.[*3*, p.20] A $(7, 4)$ Hamming code can correct any *one* error; might there be a $(14, 8)$ code that can correct any two errors?

Optional extra: Does the answer to this question depend on whether the code is linear or nonlinear?

Exercise 1.11.[*4*, p.21] Design an error-correcting code, other than a repetition code, that can correct any *two* errors in a block of size N.

▶ 1.3 What performance can the best codes achieve?

There seems to be a trade-off between the decoded bit-error probability p_b (which we would like to reduce) and the rate R (which we would like to keep large). How can this trade-off be characterized? What points in the (R, p_b) plane are achievable? This question was addressed by Claude Shannon in his pioneering paper of 1948, in which he both created the field of information theory and solved most of its fundamental problems.

At that time there was a widespread belief that the boundary between achievable and nonachievable points in the (R, p_b) plane was a curve passing through the origin $(R, p_b) = (0, 0)$; if this were so, then, in order to achieve a vanishingly small error probability p_b, one would have to reduce the rate correspondingly close to zero. 'No pain, no gain.'

However, Shannon proved the remarkable result that the boundary between achievable and nonachievable points meets the R axis at a *non-zero* value $R = C$, as shown in figure 1.19. For any channel, there exist codes that make it possible to communicate with *arbitrarily small* probability of error p_b at non-zero rates. The first half of this book (Parts I–III) will be devoted to understanding this remarkable result, which is called the *noisy-channel coding theorem*. *

Example: $f = 0.1$

The maximum rate at which communication is possible with arbitrarily small p_b is called the *capacity* of the channel. The formula for the capacity of a

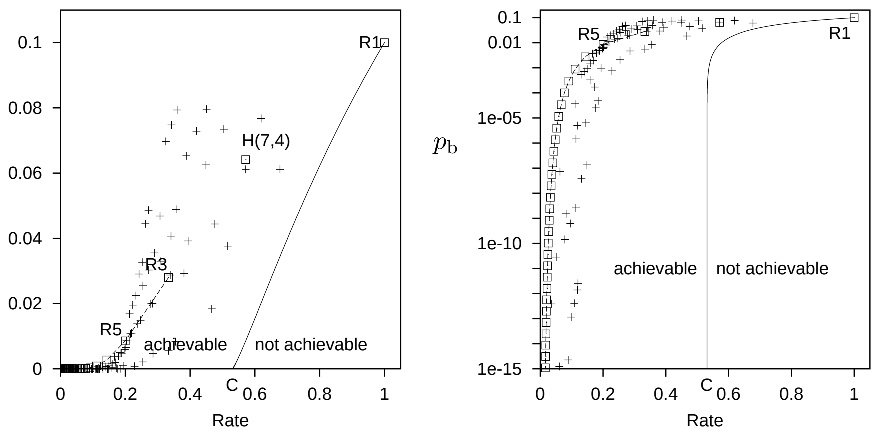


Figure 1.19. Shannon's noisy-channel coding theorem. The solid curve shows the Shannon limit on achievable values of (R, p_b) for the binary symmetric channel with $f = 0.1$. Rates up to $R = C$ are achievable with arbitrarily small p_b. The points show the performance of some textbook codes, as in figure 1.18.

The equation defining the Shannon limit (the solid curve) is $R = C/(1 - H_2(p_b))$, where C and H_2 are defined in equation (1.35).

binary symmetric channel with noise level f is

$$C(f) = 1 - H_2(f) = 1 - \left[f \log_2 \frac{1}{f} + (1-f) \log_2 \frac{1}{1-f} \right]; \tag{1.35}$$

the channel we were discussing earlier with noise level $f = 0.1$ has capacity $C \simeq 0.53$. Let us consider what this means in terms of noisy disk drives. The repetition code R_3 could communicate over this channel with $p_b = 0.03$ at a rate $R = 1/3$. Thus we know how to build a single gigabyte disk drive with $p_b = 0.03$ from three noisy gigabyte disk drives. We also know how to make a single gigabyte disk drive with $p_b \simeq 10^{-15}$ from sixty noisy one-gigabyte drives (exercise 1.3, p.8). And now Shannon passes by, notices us juggling with disk drives and codes and says:

> 'What performance are you trying to achieve? 10^{-15}? You don't need *sixty* disk drives – you can get that performance with just *two* disk drives (since 1/2 is less than 0.53). And if you want $p_b = 10^{-18}$ or 10^{-24} or anything, you can get there with two disk drives too!'

[Strictly, the above statements might not be quite right, since, as we shall see, Shannon proved his noisy-channel coding theorem by studying sequences of block codes with ever-increasing blocklengths, and the required blocklength might be bigger than a gigabyte (the size of our disk drive), in which case, Shannon might say 'well, you can't do it with those *tiny* disk drives, but if you had two noisy *terabyte* drives, you could make a single high-quality terabyte drive from them'.]

▶ 1.4 Summary

The (7, 4) Hamming Code

By including three parity-check bits in a block of 7 bits it is possible to detect and correct any single bit error in each block.

Shannon's noisy-channel coding theorem

Information can be communicated over a noisy channel at a non-zero rate with arbitrarily small error probability.

Information theory addresses both the *limitations* and the *possibilities* of communication. The noisy-channel coding theorem, which we will prove in Chapter 10, asserts both that reliable communication at any rate beyond the capacity is impossible, and that reliable communication at all rates up to capacity is possible.

The next few chapters lay the foundations for this result by discussing *how to measure information content* and the intimately related topic of *data compression*.

▶ 1.5 Further exercises

▷ Exercise 1.12.[2, p.21] Consider the repetition code R_9. One way of viewing this code is as a *concatenation* of R_3 with R_3. We first encode the source stream with R_3, then encode the resulting output with R_3. We could call this code 'R_3^2'. This idea motivates an alternative decoding algorithm, in which we decode the bits three at a time using the decoder for R_3; then decode the decoded bits from that first decoder using the decoder for R_3.

Evaluate the probability of error for this decoder and compare it with the probability of error for the optimal decoder for R_9.

Do the concatenated encoder and decoder for R_3^2 have advantages over those for R_9?

▶ 1.6 Solutions

Solution to exercise 1.2 (p.7). An error is made by R_3 if two or more bits are flipped in a block of three. So the error probability of R_3 is a sum of two terms: the probability that all three bits are flipped, f^3; and the probability that exactly two bits are flipped, $3f^2(1-f)$. [If these expressions are not obvious, see example 1.1 (p.1): the expressions are $P(r=3\,|\,f,N=3)$ and $P(r=2\,|\,f,N=3)$.]

$$p_{\mathrm{b}} = p_{\mathrm{B}} = 3f^2(1-f) + f^3 = 3f^2 - 2f^3. \tag{1.36}$$

This probability is dominated for small f by the term $3f^2$.

See exercise 2.38 (p.39) for further discussion of this problem.

Solution to exercise 1.3 (p.8). The probability of error for the repetition code R_N is dominated by the probability that $\lceil N/2 \rceil$ bits are flipped, which goes (for odd N) as

$$\binom{N}{\lceil N/2 \rceil} f^{(N+1)/2}(1-f)^{(N-1)/2}. \tag{1.37}$$

Notation: $\lceil N/2 \rceil$ denotes the smallest integer greater than or equal to $N/2$.

The term $\binom{N}{K}$ can be approximated using the binary entropy function:

$$\frac{1}{N+1} 2^{NH_2(K/N)} \leq \binom{N}{K} \leq 2^{NH_2(K/N)} \Rightarrow \binom{N}{K} \simeq 2^{NH_2(K/N)}, \tag{1.38}$$

where this approximation introduces an error of order $\sqrt{N}$ – as shown in equation (1.17). So

$$p_{\mathrm{b}} = p_{\mathrm{B}} \simeq 2^N (f(1-f))^{N/2} = (4f(1-f))^{N/2}. \tag{1.39}$$

Setting this equal to the required value of 10^{-15} we find $N \simeq 2\frac{\log 10^{-15}}{\log 4f(1-f)} = 68$. This answer is a little out because the approximation we used overestimated $\binom{N}{K}$ and we did not distinguish between $\lceil N/2 \rceil$ and $N/2$.

A slightly more careful answer (short of explicit computation) goes as follows. Taking the approximation for $\binom{N}{K}$ to the next order, we find:

$$\binom{N}{N/2} \simeq 2^N \frac{1}{\sqrt{2\pi N/4}}. \tag{1.40}$$

This approximation can be proved from an accurate version of Stirling's approximation (1.12), or by considering the binomial distribution with $p = 1/2$ and noting

$$1 = \sum_K \binom{N}{K} 2^{-N} \simeq 2^{-N} \binom{N}{N/2} \sum_{r=-N/2}^{N/2} e^{-r^2/2\sigma^2} \simeq 2^{-N} \binom{N}{N/2} \sqrt{2\pi}\sigma, \tag{1.41}$$

where $\sigma = \sqrt{N/4}$, from which equation (1.40) follows. The distinction between $\lceil N/2 \rceil$ and $N/2$ is not important in this term since $\binom{N}{K}$ has a maximum at $K = N/2$.

Then the probability of error (for odd N) is to leading order

$$\begin{aligned} p_{\rm b} &\simeq \binom{N}{(N+1)/2} f^{(N+1)/2}(1-f)^{(N-1)/2} & (1.42)\\ &\simeq 2^N \frac{1}{\sqrt{\pi N/2}} f[f(1-f)]^{(N-1)/2} \simeq \frac{1}{\sqrt{\pi N/8}} f[4f(1-f)]^{(N-1)/2}. & (1.43)\end{aligned}$$

The equation $p_{\rm b} = 10^{-15}$ can be written

In equation (1.44), the logarithms can be taken to any base, as long as it's the same base throughout. In equation (1.45), I use base 10.

$$(N-1)/2 \simeq \frac{\log 10^{-15} + \log \frac{\sqrt{\pi N/8}}{f}}{\log 4f(1-f)} \tag{1.44}$$

which may be solved for N iteratively, the first iteration starting from $\hat{N}_1 = 68$:

$$(\hat{N}_2 - 1)/2 \simeq \frac{-15 + 1.7}{-0.44} = 29.9 \;\Rightarrow\; \hat{N}_2 \simeq 60.9. \tag{1.45}$$

This answer is found to be stable, so $N \simeq 61$ is the blocklength at which $p_{\rm b} \simeq 10^{-15}$.

Solution to exercise 1.6 (p.13).

(a) The probability of block error of the Hamming code is a sum of six terms – the probabilities that 2, 3, 4, 5, 6, or 7 errors occur in one block.

$$p_{\rm B} = \sum_{r=2}^{7} \binom{7}{r} f^r (1-f)^{7-r}. \tag{1.46}$$

To leading order, this goes as

$$p_{\rm B} \simeq \binom{7}{2} f^2 = 21 f^2. \tag{1.47}$$

(b) The probability of bit error of the Hamming code is smaller than the probability of block error because a block error rarely corrupts all bits in the decoded block. The leading-order behaviour is found by considering the outcome in the most probable case where the noise vector has weight two. The decoder will erroneously flip a *third* bit, so that the modified received vector (of length 7) differs in three bits from the transmitted vector. That means, if we average over all seven bits, the probability that a randomly chosen bit is flipped is 3/7 times the block error probability, to leading order. Now, what we really care about is the probability that a source bit is flipped. Are parity bits or source bits more likely to be among these three flipped bits, or are all seven bits equally likely to be corrupted when the noise vector has weight two? The Hamming code is in fact completely symmetric in the protection it affords to the seven bits (assuming a binary symmetric channel). [This symmetry can be proved by showing that the role of a parity bit can be exchanged with a source bit and the resulting code is still a $(7, 4)$ Hamming code; see below.] The probability that any one bit ends up corrupted is the same for all seven bits. So the probability of bit error (for the source bits) is simply three sevenths of the probability of block error.

$$p_{\rm b} \simeq \frac{3}{7} p_{\rm B} \simeq 9f^2. \tag{1.48}$$

Symmetry of the Hamming $(7, 4)$ *code*

To prove that the $(7, 4)$ code protects all bits equally, we start from the parity-check matrix

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{1.49}$$

The symmetry among the seven transmitted bits will be easiest to see if we reorder the seven bits using the permutation $(t_1 t_2 t_3 t_4 t_5 t_6 t_7) \rightarrow (t_5 t_2 t_3 t_4 t_1 t_6 t_7)$. Then we can rewrite $\mathbf{H}$ thus:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}. \tag{1.50}$$

Now, if we take any two parity constraints that $\mathbf{t}$ satisfies and add them together, we get another parity constraint. For example, row 1 asserts $t_5 + t_2 + t_3 + t_1 = \text{even}$, and row 2 asserts $t_2 + t_3 + t_4 + t_6 = \text{even}$, and the sum of these two constraints is

$$t_5 + 2t_2 + 2t_3 + t_1 + t_4 + t_6 = \text{even}; \tag{1.51}$$

we can drop the terms $2t_2$ and $2t_3$, since they are even whatever t_2 and t_3 are; thus we have derived the parity constraint $t_5 + t_1 + t_4 + t_6 = \text{even}$, which we can if we wish add into the parity-check matrix as a fourth row. [The set of vectors satisfying $\mathbf{Ht} = \mathbf{0}$ will not be changed.] We thus define

$$\mathbf{H}' = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}. \tag{1.52}$$

The fourth row is the sum (modulo two) of the top two rows. Notice that *the second, third, and fourth rows are all cyclic shifts of the top row.* If, having added the fourth redundant constraint, we drop the first constraint, we obtain a new parity-check matrix $\mathbf{H}''$,

$$\mathbf{H}'' = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}, \tag{1.53}$$

which still satisfies $\mathbf{H}''\mathbf{t} = 0$ for all codewords, and which looks just like the starting $\mathbf{H}$ in (1.50), except that all the columns have shifted along one to the right, and the rightmost column has reappeared at the left (a cyclic permutation of the columns).

This establishes the symmetry among the seven bits. Iterating the above procedure five more times, we can make a total of seven different **H** matrices for the same original code, each of which assigns each bit to a different role.

We may also construct the super-redundant seven-row parity-check matrix for the code,

$$\mathbf{H}''' = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{1.54}$$

This matrix is 'redundant' in the sense that the space spanned by its rows is only three-dimensional, not seven.

This matrix is also a *cyclic* matrix. Every row is a cyclic permutation of the top row.

Cyclic codes: if there is an ordering of the bits $t_1 \dots t_N$ such that a linear code has a *cyclic* parity-check matrix, then the code is called a *cyclic code*.

The codewords of such a code also have cyclic properties: any cyclic permutation of a codeword is a codeword.

For example, the Hamming $(7, 4)$ code, with its bits ordered as above, consists of all seven cyclic shifts of the codewords `1110100` and `1011000`, and the codewords `0000000` and `1111111`.

Cyclic codes are a cornerstone of the algebraic approach to error-correcting codes. We won't use them again in this book, however, as they have been superceded by sparse-graph codes (Part VI).

Solution to exercise 1.7 (p.13). There are fifteen non-zero noise vectors which give the all-zero syndrome; these are precisely the fifteen non-zero codewords of the Hamming code. Notice that because the Hamming code is *linear*, the sum of any two codewords is a codeword.

Graphs corresponding to codes

Solution to exercise 1.9 (p.14). When answering this question, you will probably find that it is easier to invent new codes than to find optimal decoders for them. There are many ways to design codes, and what follows is just one possible train of thought. We make a linear block code that is similar to the $(7, 4)$ Hamming code, but bigger.

Many codes can be conveniently expressed in terms of graphs. In figure 1.13, we introduced a pictorial representation of the $(7, 4)$ Hamming code. If we replace that figure's big circles, each of which shows that the parity of four particular bits is even, by a 'parity-check node' that is connected to the four bits, then we obtain the representation of the $(7, 4)$ Hamming code by a *bipartite graph* as shown in figure 1.20. The 7 circles are the 7 transmitted bits. The 3 squares are the parity-check nodes (not to be confused with the 3 parity-check *bits*, which are the three most peripheral circles). The graph is a 'bipartite' graph because its nodes fall into two classes – bits and checks

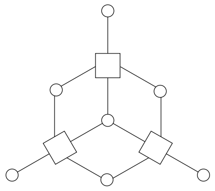


Figure 1.20. The graph of the $(7, 4)$ Hamming code. The 7 circles are the bit nodes and the 3 squares are the parity-check nodes.

– and there are edges only between nodes in different classes. The graph and the code's parity-check matrix (1.30) are simply related to each other: each parity-check node corresponds to a row of $\mathbf{H}$ and each bit node corresponds to a column of $\mathbf{H}$; for every 1 in $\mathbf{H}$, there is an edge between the corresponding pair of nodes.

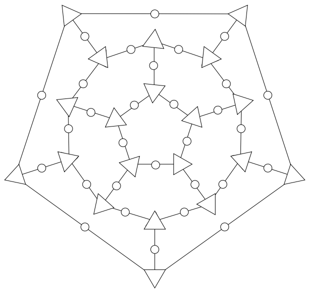


Figure 1.21. The graph defining the $(30, 11)$ dodecahedron code. The circles are the 30 transmitted bits and the triangles are the 20 parity checks. One parity check is redundant.

Having noticed this connection between linear codes and graphs, one way to invent linear codes is simply to think of a bipartite graph. For example, a pretty bipartite graph can be obtained from a dodecahedron by calling the vertices of the dodecahedron the parity-check nodes, and putting a transmitted bit on each edge in the dodecahedron. This construction defines a parity-check matrix in which every column has weight 2 and every row has weight 3. [The weight of a binary vector is the number of 1s it contains.]

This code has $N = 30$ bits, and it appears to have $M_{\text{apparent}} = 20$ parity-check constraints. Actually, there are only $M = 19$ *independent* constraints; the 20th constraint is redundant (that is, if 19 constraints are satisfied, then the 20th is automatically satisfied); so the number of source bits is $K = N - M = 11$. The code is a $(30, 11)$ code.

It is hard to find a decoding algorithm for this code, but we can estimate its probability of error by finding its lowest-weight codewords. If we flip all the bits surrounding one face of the original dodecahedron, then all the parity checks will be satisfied; so the code has 12 codewords of weight 5, one for each face. Since the lowest-weight codewords have weight 5, we say that the code has distance $d = 5$; the $(7, 4)$ Hamming code had distance 3 and could correct all single bit-flip errors. A code with distance 5 can correct all double bit-flip errors, but there are some triple bit-flip errors that it cannot correct. So the error probability of this code, assuming a binary symmetric channel, will be dominated, at least for low noise levels f, by a term of order f^3, perhaps something like

$$12\binom{5}{3}f^3(1-f)^{27}. \tag{1.55}$$

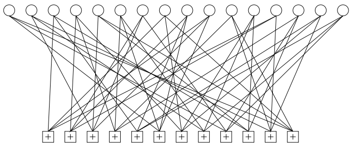


Figure 1.22. Graph of a rate-$1/4$ low-density parity-check code (Gallager code) with blocklength $N = 16$, and $M = 12$ parity-check constraints. Each white circle represents a transmitted bit. Each bit participates in $j = 3$ constraints, represented by ⊞ squares. The edges between nodes were placed at random. (See Chapter 47 for more.)

Of course, there is no obligation to make codes whose graphs can be represented on a plane, as this one can; the best linear codes, which have simple graphical descriptions, have graphs that are more tangled, as illustrated by the tiny $(16, 4)$ code of figure 1.22.

Furthermore, there is no reason for sticking to linear codes; indeed some nonlinear codes – codes whose codewords cannot be defined by a linear equation like $\mathbf{Ht} = \mathbf{0}$ – have very good properties. But the encoding and decoding of a nonlinear code are even trickier tasks.

Solution to exercise 1.10 (p.14). First let's assume we are making a linear code and decoding it with syndrome decoding. If there are N transmitted bits, then the number of possible error patterns of weight up to two is

$$\binom{N}{2}+\binom{N}{1}+\binom{N}{0}. \tag{1.56}$$

For $N = 14$, that's $91 + 14 + 1 = 106$ patterns. Now, every distinguishable error pattern must give rise to a distinct syndrome; and the syndrome is a list of M bits, so the maximum possible number of syndromes is 2^M. For a $(14, 8)$ code, $M = 6$, so there are at most $2^6 = 64$ syndromes. The number of possible error patterns of weight up to two, 106, is bigger than the number of syndromes, 64, so we can immediately rule out the possibility that there is a $(14, 8)$ code that is 2-error-correcting.

The same counting argument works fine for nonlinear codes too. When the decoder receives $\mathbf{r} = \mathbf{t} + \mathbf{n}$, his aim is to deduce both $\mathbf{t}$ and $\mathbf{n}$ from $\mathbf{r}$. If it is the case that the sender can select any transmission $\mathbf{t}$ from a code of size $S_{\mathbf{t}}$, and the channel can select any noise vector from a set of size $S_{\mathbf{n}}$, and those two selections can be recovered from the received bit string $\mathbf{r}$, which is one of at most 2^N possible strings, then it must be the case that

$$S_{\mathbf{t}} S_{\mathbf{n}} \leq 2^N. \tag{1.57}$$

So, for a (N, K) two-error-correcting code, whether linear or nonlinear,

$$2^K \left[\binom{N}{2} + \binom{N}{1} + \binom{N}{0} \right] \leq 2^N. \tag{1.58}$$

Solution to exercise 1.11 (p.14). There are various strategies for making codes that can correct multiple errors, and I strongly recommend you think out one or two of them for yourself.

If your approach uses a linear code, e.g., one with a collection of M parity checks, it is helpful to bear in mind the counting argument given in the previous exercise, in order to anticipate how many parity checks, M, you might need.

Examples of codes that can correct any two errors are the $(30, 11)$ dodecahedron code on page 20, and the $(15, 6)$ pentagonful code to be introduced on p. 221. Further simple ideas for making codes that can correct multiple errors from codes that can correct only one error are discussed in section 13.7.

Solution to exercise 1.12 (p.16). The probability of error of R_3^2 is, to leading order,

$$p_{\mathrm{b}}(\mathrm{R}_3^2) \simeq 3\,[p_{\mathrm{b}}(\mathrm{R}_3)]^2 = 3(3f^2)^2 + \cdots = 27f^4 + \cdots, \tag{1.59}$$

whereas the probability of error of R_9 is dominated by the probability of five flips,

$$p_{\mathrm{b}}(\mathrm{R}_9) \simeq \binom{9}{5} f^5 (1-f)^4 \simeq 126 f^5 + \cdots. \tag{1.60}$$

The R_3^2 decoding procedure is therefore suboptimal, since there are noise vectors of weight four that cause it to make a decoding error.

It has the advantage, however, of requiring smaller computational resources: only memorization of three bits, and counting up to three, rather than counting up to nine.

This simple code illustrates an important concept. Concatenated codes are widely used in practice because concatenation allows large codes to be implemented using simple encoding and decoding hardware. Some of the best known practical codes are concatenated codes.

2

Probability, Entropy, and Inference

This chapter, and its sibling, Chapter 8, devote some time to notation. Just as the White Knight distinguished between the song, the name of the song, and what the name of the song was called (Carroll, 1998), we will sometimes need to be careful to distinguish between a random variable, the value of the random variable, and the proposition that asserts that the random variable has a particular value. In any particular chapter, however, I will use the most simple and friendly notation possible, at the risk of upsetting pure-minded readers. For example, if something is 'true with probability 1', I will usually simply say that it is 'true'.

▶ 2.1 Probabilities and ensembles

An ensemble X is a triple $(x, \mathcal{A}_X, \mathcal{P}_X)$, where the *outcome* x is the value of a random variable, which takes on one of a set of possible values, $\mathcal{A}_X = \{a_1, a_2, \ldots, a_i, \ldots, a_I\}$, having probabilities $\mathcal{P}_X = \{p_1, p_2, \ldots, p_I\}$, with $P(x\!=\!a_i) = p_i$, $p_i \geq 0$ and $\sum_{a_i \in \mathcal{A}_X} P(x\!=\!a_i) = 1$.

The name $\mathcal{A}$ is mnemonic for 'alphabet'. One example of an ensemble is a letter that is randomly selected from an English document. This ensemble is shown in figure 2.1. There are twenty-seven possible letters: a–z, and a space character '-'.

i	a_i	p_i
1	a	0.0575
2	b	0.0128
3	c	0.0263
4	d	0.0285
5	e	0.0913
6	f	0.0173
7	g	0.0133
8	h	0.0313
9	i	0.0599
10	j	0.0006
11	k	0.0084
12	l	0.0335
13	m	0.0235
14	n	0.0596
15	o	0.0689
16	p	0.0192
17	q	0.0008
18	r	0.0508
19	s	0.0567
20	t	0.0706
21	u	0.0334
22	v	0.0069
23	w	0.0119
24	x	0.0073
25	y	0.0164
26	z	0.0007
27	–	0.1928

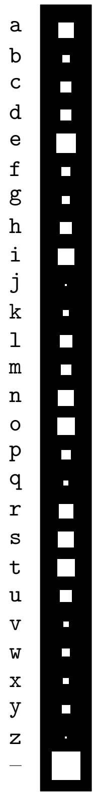


Figure 2.1. Probability distribution over the 27 outcomes for a randomly selected letter in an English language document (estimated from *The Frequently Asked Questions Manual for Linux*). The picture shows the probabilities by the areas of white squares.

Abbreviations. Briefer notation will sometimes be used. For example, $P(x\!=\!a_i)$ may be written as $P(a_i)$ or $P(x)$.

Probability of a subset. If T is a subset of $\mathcal{A}_X$ then:

$$P(T) = P(x \in T) = \sum_{a_i \in T} P(x\!=\!a_i). \tag{2.1}$$

For example, if we define V to be vowels from figure 2.1, $V = \{\mathtt{a}, \mathtt{e}, \mathtt{i}, \mathtt{o}, \mathtt{u}\}$, then

$$P(V) = 0.06 + 0.09 + 0.06 + 0.07 + 0.03 = 0.31. \tag{2.2}$$

A joint ensemble XY is an ensemble in which each outcome is an ordered pair x, y with $x \in \mathcal{A}_X = \{a_1, \ldots, a_I\}$ and $y \in \mathcal{A}_Y = \{b_1, \ldots, b_J\}$.

We call $P(x, y)$ the joint probability of x and y.

Commas are optional when writing ordered pairs, so $xy \Leftrightarrow x, y$.

N.B. In a joint ensemble XY the two variables are not necessarily independent.

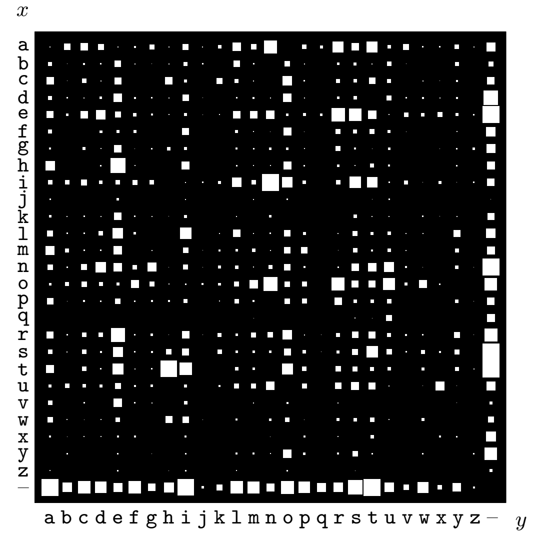


Figure 2.2. The probability distribution over the 27×27 possible bigrams xy in an English language document, *The Frequently Asked Questions Manual for Linux.*

Marginal probability. We can obtain the marginal probability $P(x)$ from the joint probability $P(x, y)$ by summation:

$$P(x\!=\!a_i) \equiv \sum_{y \in \mathcal{A}_Y} P(x\!=\!a_i, y). \qquad (2.3)$$

Similarly, using briefer notation, the marginal probability of y is:

$$P(y) \equiv \sum_{x \in \mathcal{A}_X} P(x, y). \qquad (2.4)$$

Conditional probability

$$P(x\!=\!a_i \,|\, y\!=\!b_j) \equiv \frac{P(x\!=\!a_i, y\!=\!b_j)}{P(y\!=\!b_j)} \text{ if } P(y\!=\!b_j) \neq 0. \qquad (2.5)$$

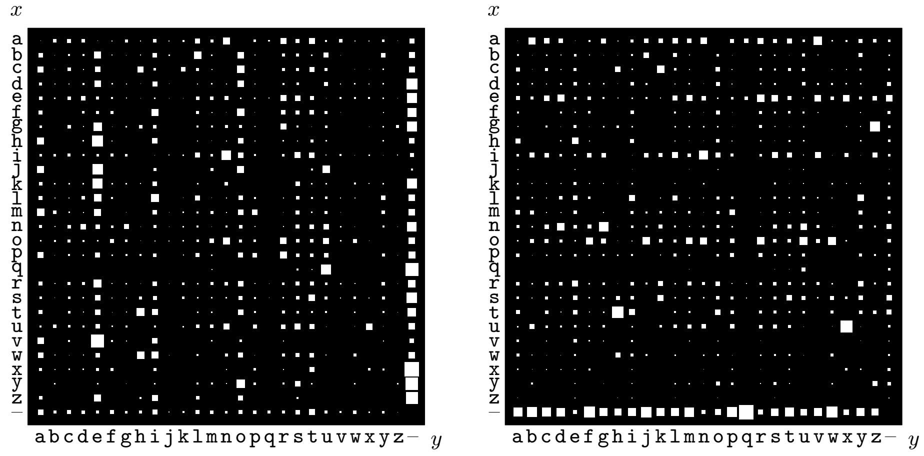
[If $P(y\!=\!b_j) = 0$ then $P(x\!=\!a_i \,|\, y\!=\!b_j)$ is undefined.]

We pronounce $P(x\!=\!a_i \,|\, y\!=\!b_j)$ 'the probability that x equals a_i, given y equals b_j'.

Example 2.1. An example of a joint ensemble is the ordered pair XY consisting of two successive letters in an English document. The possible outcomes are ordered pairs such as `aa`, `ab`, `ac`, and `zz`; of these, we might expect `ab` and `ac` to be more probable than `aa` and `zz`. An estimate of the joint probability distribution for two neighbouring characters is shown graphically in figure 2.2.

This joint ensemble has the special property that its two marginal distributions, $P(x)$ and $P(y)$, are identical. They are both equal to the monogram distribution shown in figure 2.1.

From this joint ensemble $P(x, y)$ we can obtain conditional distributions, $P(y \,|\, x)$ and $P(x \,|\, y)$, by normalizing the rows and columns, respectively (figure 2.3). The probability $P(y \,|\, x\!=\!\texttt{q})$ is the probability distribution of the second letter given that the first letter is a `q`. As you can see in figure 2.3a, the two most probable values for the second letter y given



(a) $P(y \,|\, x)$

(b) $P(x \,|\, y)$

Figure 2.3. Conditional probability distributions. (a) $P(y \,|\, x)$: Each *row* shows the conditional distribution of the second letter, y, given the first letter, x, in a bigram xy. (b) $P(x \,|\, y)$: Each *column* shows the conditional distribution of the first letter, x, given the second letter, y.

that the first letter x is `q` are `u` and `-`. (The space is common after `q` because the source document makes heavy use of the word `FAQ`.)

The probability $P(x \,|\, y = \texttt{u})$ is the probability distribution of the first letter x given that the second letter y is a `u`. As you can see in figure 2.3b the two most probable values for x given $y = \texttt{u}$ are `n` and `o`.

Rather than writing down the joint probability directly, we often define an ensemble in terms of a collection of conditional probabilities. The following rules of probability theory will be useful. ($\mathcal{H}$ denotes assumptions on which the probabilities are based.)

Product rule – obtained from the definition of conditional probability:

$$P(x, y \,|\, \mathcal{H}) = P(x \,|\, y, \mathcal{H}) P(y \,|\, \mathcal{H}) = P(y \,|\, x, \mathcal{H}) P(x \,|\, \mathcal{H}). \tag{2.6}$$

This rule is also known as the chain rule.

Sum rule – a rewriting of the marginal probability definition:

$$\begin{aligned} P(x \,|\, \mathcal{H}) &= \sum_y P(x, y \,|\, \mathcal{H}) && (2.7) \\ &= \sum_y P(x \,|\, y, \mathcal{H}) P(y \,|\, \mathcal{H}). && (2.8) \end{aligned}$$

Bayes' theorem – obtained from the product rule:

$$\begin{aligned} P(y \,|\, x, \mathcal{H}) &= \frac{P(x \,|\, y, \mathcal{H}) P(y \,|\, \mathcal{H})}{P(x \,|\, \mathcal{H})} && (2.9) \\ &= \frac{P(x \,|\, y, \mathcal{H}) P(y \,|\, \mathcal{H})}{\sum_{y'} P(x \,|\, y', \mathcal{H}) P(y' \,|\, \mathcal{H})}. && (2.10) \end{aligned}$$

Independence. Two random variables X and Y are *independent* (sometimes written $X \bot Y$) if and only if

$$P(x, y) = P(x) P(y). \tag{2.11}$$

Exercise 2.2.$^{[\mathit{1},\ \mathrm{p.40}]}$ Are the random variables X and Y in the joint ensemble of figure 2.2 independent?

I said that we often define an ensemble in terms of a collection of conditional probabilities. The following example illustrates this idea.

Example 2.3. Jo has a test for a nasty disease. We denote Jo's state of health by the variable a and the test result by b.

$$\begin{array}{ll} a=1 & \text{Jo has the disease} \\ a=0 & \text{Jo does not have the disease.} \end{array} \tag{2.12}$$

The result of the test is either 'positive' ($b = 1$) or 'negative' ($b = 0$); the test is 95% reliable: in 95% of cases of people who really have the disease, a positive result is returned, and in 95% of cases of people who do not have the disease, a negative result is obtained. The final piece of background information is that 1% of people of Jo's age and background have the disease.

OK – Jo has the test, and the result is positive. What is the probability that Jo has the disease?

Solution. We write down all the provided probabilities. The test reliability specifies the conditional probability of b given a:

$$\begin{array}{ll} P(b\!=\!1 \,|\, a\!=\!1) = 0.95 & P(b\!=\!1 \,|\, a\!=\!0) = 0.05 \\ P(b\!=\!0 \,|\, a\!=\!1) = 0.05 & P(b\!=\!0 \,|\, a\!=\!0) = 0.95; \end{array} \tag{2.13}$$

and the disease prevalence tells us about the marginal probability of a:

$$P(a\!=\!1) = 0.01 \qquad P(a\!=\!0) = 0.99. \tag{2.14}$$

From the marginal $P(a)$ and the conditional probability $P(b \,|\, a)$ we can deduce the joint probability $P(a, b) = P(a)P(b \,|\, a)$ and any other probabilities we are interested in. For example, by the sum rule, the marginal probability of $b\!=\!1$ – the probability of getting a positive result – is

$$P(b\!=\!1) = P(b\!=\!1 \,|\, a\!=\!1)P(a\!=\!1) + P(b\!=\!1 \,|\, a\!=\!0)P(a\!=\!0). \tag{2.15}$$

Jo has received a positive result $b\!=\!1$ and is interested in how plausible it is that she has the disease (i.e., that $a\!=\!1$). The man in the street might be duped by the statement 'the test is 95% reliable, so Jo's positive result implies that there is a 95% chance that Jo has the disease', but this is incorrect. The correct solution to an inference problem is found using Bayes' theorem.

$$\begin{aligned} P(a\!=\!1 \,|\, b\!=\!1) &= \frac{P(b\!=\!1 \,|\, a\!=\!1)P(a\!=\!1)}{P(b\!=\!1 \,|\, a\!=\!1)P(a\!=\!1) + P(b\!=\!1 \,|\, a\!=\!0)P(a\!=\!0)} && (2.16) \\ &= \frac{0.95 \times 0.01}{0.95 \times 0.01 + 0.05 \times 0.99} && (2.17) \\ &= 0.16. && (2.18) \end{aligned}$$

So in spite of the positive result, the probability that Jo has the disease is only 16%. □

▶ 2.2 The meaning of probability

Probabilities can be used in two ways.

Probabilities can describe *frequencies of outcomes in random experiments*, but giving noncircular definitions of the terms 'frequency' and 'random' is a challenge – what does it mean to say that the frequency of a tossed coin's

Notation. Let ‘the degree of belief in proposition x’ be denoted by $B(x)$. The negation of x (NOT-x) is written $\overline{x}$. The degree of belief in a conditional proposition, ‘x, assuming proposition y to be true’, is represented by $B(x \mid y)$.

Axiom 1. Degrees of belief can be ordered; if $B(x)$ is ‘greater’ than $B(y)$, and $B(y)$ is ‘greater’ than $B(z)$, then $B(x)$ is ‘greater’ than $B(z)$.

[Consequence: beliefs can be mapped onto real numbers.]

Axiom 2. The degree of belief in a proposition x and its negation $\overline{x}$ are related. There is a function f such that

$$B(x) = f[B(\overline{x})].$$

Axiom 3. The degree of belief in a conjunction of propositions x, y (x AND y) is related to the degree of belief in the conditional proposition $x \mid y$ and the degree of belief in the proposition y. There is a function g such that

$$B(x, y) = g\left[B(x \mid y), B(y)\right].$$

Box 2.4. The Cox axioms. If a set of beliefs satisfy these axioms then they can be mapped onto probabilities satisfying $P(\text{FALSE}) = 0$, $P(\text{TRUE}) = 1$, $0 \leq P(x) \leq 1$, and the rules of probability:

$$P(x) = 1 - P(\overline{x}),$$

and

$$P(x, y) = P(x \mid y)P(y).$$

coming up heads is $1/2$? If we say that this frequency is the average fraction of heads in long sequences, we have to define ‘average’; and it is hard to define ‘average’ without using a word synonymous to probability! I will not attempt to cut this philosophical knot.

Probabilities can also be used, more generally, to describe *degrees of belief* in propositions that do not involve random variables – for example ‘the probability that Mr. S. was the murderer of Mrs. S., given the evidence’ (he either was or wasn’t, and it’s the jury’s job to assess how probable it is that he was); ‘the probability that Thomas Jefferson had a child by one of his slaves’; ‘the probability that Shakespeare’s plays were written by Francis Bacon’; or, to pick a modern-day example, ‘the probability that a particular signature on a particular cheque is genuine’.

The man in the street is happy to use probabilities in both these ways, but some books on probability restrict probabilities to refer only to frequencies of outcomes in repeatable random experiments.

Nevertheless, degrees of belief *can* be mapped onto probabilities if they satisfy simple consistency rules known as the Cox axioms (Cox, 1946) (figure 2.4). Thus probabilities can be used to describe assumptions, and to describe inferences given those assumptions. The rules of probability ensure that if two people make the same assumptions and receive the same data then they will draw identical conclusions. This more general use of probability to quantify beliefs is known as the *Bayesian* viewpoint. It is also known as the *subjective* interpretation of probability, since the probabilities depend on assumptions. Advocates of a Bayesian approach to data modelling and pattern recognition do not view this subjectivity as a defect, since in their view,

> you cannot do inference without making assumptions.

In this book it will from time to time be taken for granted that a Bayesian approach makes sense, but the reader is warned that this is not yet a globally held view – the field of statistics was dominated for most of the 20th century by non-Bayesian methods in which probabilities are allowed to describe only random variables. The big difference between the two approaches is that

Bayesians also use probabilities to describe *inferences.*

▶ 2.3 Forward probabilities and inverse probabilities

Probability calculations often fall into one of two categories: *forward probability* and *inverse probability*. Here is an example of a forward probability problem:

Exercise 2.4.$^{[2,\ \text{p.}40]}$ An urn contains K balls, of which B are black and $W = K - B$ are white. Fred draws a ball at random from the urn and replaces it, N times.

(a) What is the probability distribution of the number of times a black ball is drawn, n_B?

(b) What is the expectation of n_B? What is the variance of n_B? What is the standard deviation of n_B? Give numerical answers for the cases $N = 5$ and $N = 400$, when $B = 2$ and $K = 10$.

Forward probability problems involve a *generative model* that describes a process that is assumed to give rise to some data; the task is to compute the probability distribution or expectation of some quantity that depends on the data. Here is another example of a forward probability problem:

Exercise 2.5.$^{[2,\ \text{p.}40]}$ An urn contains K balls, of which B are black and $W = K - B$ are white. We define the fraction $f_B \equiv B/K$. Fred draws N times from the urn, exactly as in exercise 2.4, obtaining n_B blacks, and computes the quantity

$$z = \frac{(n_B - f_B N)^2}{N f_B (1 - f_B)}. \tag{2.19}$$

What is the expectation of z? In the case $N = 5$ and $f_B = 1/5$, what is the probability distribution of z? What is the probability that $z < 1$? [Hint: compare z with the quantities computed in the previous exercise.]

Like forward probability problems, *inverse probability problems* involve a generative model of a process, but instead of computing the probability distribution of some quantity *produced* by the process, we compute the conditional probability of one or more of the *unobserved variables* in the process, *given* the observed variables. This invariably requires the use of Bayes' theorem.

Example 2.6. There are eleven urns labelled by $u \in \{0, 1, 2, \ldots, 10\}$, each containing ten balls. Urn u contains u black balls and $10 - u$ white balls. Fred selects an urn u at random and draws N times with replacement from that urn, obtaining n_B blacks and $N - n_B$ whites. Fred's friend, Bill, looks on. If after $N = 10$ draws $n_B = 3$ blacks have been drawn, what is the probability that the urn Fred is using is urn u, from Bill's point of view? (Bill doesn't know the value of u.)

Solution. The joint probability distribution of the random variables u and n_B can be written

$$P(u, n_B \,|\, N) = P(n_B \,|\, u, N) P(u). \tag{2.20}$$

From the joint probability of u and n_B, we can obtain the conditional distribution of u given n_B:

$$\begin{aligned} P(u \,|\, n_B, N) &= \frac{P(u, n_B \,|\, N)}{P(n_B \,|\, N)} &(2.21)\\ &= \frac{P(n_B \,|\, u, N) P(u)}{P(n_B \,|\, N)}. &(2.22) \end{aligned}$$

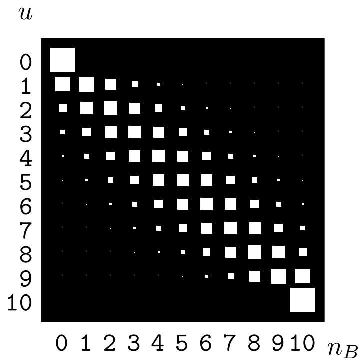


Figure 2.5. Joint probability of u and n_B for Bill and Fred's urn problem, after $N = 10$ draws.

The marginal probability of u is $P(u) = \frac{1}{11}$ for all u. You wrote down the probability of n_B given u and N, $P(n_B \,|\, u, N)$, when you solved exercise 2.4 (p.27). [You *are* doing the highly recommended exercises, aren't you?] If we define $f_u \equiv u/10$ then

$$P(n_B \,|\, u, N) = \binom{N}{n_B} f_u^{n_B} (1 - f_u)^{N - n_B}. \tag{2.23}$$

What about the denominator, $P(n_B \,|\, N)$? This is the marginal probability of n_B, which we can obtain using the sum rule:

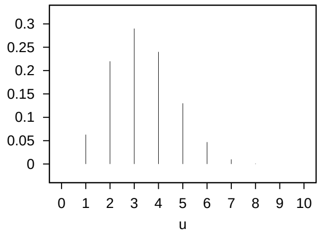
$$P(n_B \,|\, N) = \sum_u P(u, n_B \,|\, N) = \sum_u P(u) P(n_B \,|\, u, N). \tag{2.24}$$

So the conditional probability of u given n_B is

$$P(u \,|\, n_B, N) = \frac{P(u) P(n_B \,|\, u, N)}{P(n_B \,|\, N)} \tag{2.25}$$

$$= \frac{1}{P(n_B \,|\, N)} \frac{1}{11} \binom{N}{n_B} f_u^{n_B} (1 - f_u)^{N - n_B}. \tag{2.26}$$

This conditional distribution can be found by normalizing column 3 of figure 2.5 and is shown in figure 2.6. The normalizing constant, the marginal probability of n_B, is $P(n_B = 3 \,|\, N = 10) = 0.083$. The posterior probability (2.26) is correct for all u, including the end-points $u = 0$ and $u = 10$, where $f_u = 0$ and $f_u = 1$ respectively. The posterior probability that $u = 0$ given $n_B = 3$ is equal to zero, because if Fred were drawing from urn 0 it would be impossible for any black balls to be drawn. The posterior probability that $u = 10$ is also zero, because there are no white balls in that urn. The other hypotheses $u = 1$, $u = 2$, … $u = 9$ all have non-zero posterior probability. □



u	$P(u \,\|\, n_B = 3, N)$
0	0
1	0.063
2	0.22
3	0.29
4	0.24
5	0.13
6	0.047
7	0.0099
8	0.00086
9	0.0000096
10	0

Figure 2.6. Conditional probability of u given $n_B = 3$ and $N = 10$.

Terminology of inverse probability

In inverse probability problems it is convenient to give names to the probabilities appearing in Bayes' theorem. In equation (2.25), we call the marginal probability $P(u)$ the *prior* probability of u, and $P(n_B \,|\, u, N)$ is called the *likelihood* of u. It is important to note that the terms likelihood and probability are not synonyms. The quantity $P(n_B \,|\, u, N)$ is a function of both n_B and u. For fixed u, $P(n_B \,|\, u, N)$ defines a *probability* over n_B. For fixed n_B, $P(n_B \,|\, u, N)$ defines the *likelihood* of u.

> Never say 'the likelihood of the data'. Always say 'the likelihood of the parameters'. The likelihood function is not a probability distribution.

(If you want to mention the data that a likelihood function is associated with, you may say 'the likelihood of the parameters given the data'.)

The conditional probability $P(u \,|\, n_B, N)$ is called the *posterior probability* of u given n_B. The normalizing constant $P(n_B \,|\, N)$ has no u-dependence so its value is not important if we simply wish to evaluate the relative probabilities of the alternative hypotheses u. However, in most data-modelling problems of any complexity, this quantity becomes important, and it is given various names: $P(n_B \,|\, N)$ is known as the *evidence* or the *marginal likelihood*.

If $\boldsymbol{\theta}$ denotes the unknown parameters, D denotes the data, and $\mathcal{H}$ denotes the overall hypothesis space, the general equation:

$$P(\boldsymbol{\theta} \,|\, D, \mathcal{H}) = \frac{P(D \,|\, \boldsymbol{\theta}, \mathcal{H}) P(\boldsymbol{\theta} \,|\, \mathcal{H})}{P(D \,|\, \mathcal{H})} \tag{2.27}$$

is written:

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}. \tag{2.28}$$

Inverse probability and prediction

Example 2.6 (continued). Assuming again that Bill has observed $n_B = 3$ blacks in $N = 10$ draws, let Fred draw another ball from the same urn. What is the probability that the next drawn ball is a black? [You should make use of the posterior probabilities in figure 2.6.]

Solution. By the sum rule,

$$P(\text{ball}_{N+1} \text{ is black} \,|\, n_B, N) = \sum_u P(\text{ball}_{N+1} \text{ is black} \,|\, u, n_B, N) P(u \,|\, n_B, N). \tag{2.29}$$

Since the balls are drawn with replacement from the chosen urn, the probability $P(\text{ball}_{N+1} \text{ is black} \,|\, u, n_B, N)$ is just $f_u = u/10$, whatever n_B and N are. So

$$P(\text{ball}_{N+1} \text{ is black} \,|\, n_B, N) = \sum_u f_u P(u \,|\, n_B, N). \tag{2.30}$$

Using the values of $P(u \,|\, n_B, N)$ given in figure 2.6 we obtain

$$P(\text{ball}_{N+1} \text{ is black} \,|\, n_B\!=\!3, N\!=\!10) = 0.333. \qquad \square \tag{2.31}$$

Comment. Notice the difference between this prediction obtained using probability theory, and the widespread practice in statistics of making predictions by first selecting the most plausible hypothesis (which here would be that the urn is urn $u = 3$) and then making the predictions assuming that hypothesis to be true (which would give a probability of 0.3 that the next ball is black). The correct prediction is the one that takes into account the uncertainty by *marginalizing* over the possible values of the hypothesis u. Marginalization here leads to slightly more moderate, less extreme predictions.

Inference as inverse probability

Now consider the following exercise, which has the character of a simple scientific investigation.

Example 2.7. Bill tosses a bent coin N times, obtaining a sequence of heads and tails. We assume that the coin has a probability f_H of coming up heads; we do not know f_H. If n_H heads have occurred in N tosses, what is the probability distribution of f_H? (For example, N might be 10, and n_H might be 3; or, after a lot more tossing, we might have $N = 300$ and $n_H = 29$.) What is the probability that the $N+1$th outcome will be a head, given n_H heads in N tosses?

Unlike example 2.6 (p.27), this problem has a subjective element. Given a restricted definition of probability that says 'probabilities are the frequencies of random variables', this example is different from the eleven-urns example. Whereas the urn u was a random variable, the bias f_H of the coin would not normally be called a random variable. It is just a fixed but unknown parameter that we are interested in. Yet don't the two examples 2.6 and 2.7 seem to have an essential similarity? [Especially when $N = 10$ and $n_H = 3$!]

To solve example 2.7, we have to make an assumption about what the bias of the coin f_H might be. This prior probability distribution over f_H, $P(f_H)$, corresponds to the prior over u in the eleven-urns problem. In that example, the helpful problem definition specified $P(u)$. In real life, we have to make assumptions in order to assign priors; these assumptions will be subjective, and our answers will depend on them. Exactly the same can be said for the other probabilities in our generative model too. We are assuming, for example, that the balls are drawn from an urn independently; but could there not be correlations in the sequence because Fred's ball-drawing action is not perfectly random? Indeed there could be, so the likelihood function that we use depends on assumptions too. In real data modelling problems, priors are subjective *and so are likelihoods.*

Here $P(f)$ denotes a probability density, rather than a probability distribution.

We are now using $P()$ to denote probability *densities* over continuous variables as well as probabilities over discrete variables and probabilities of logical propositions. The probability that a continuous variable v lies between values a and b (where $b > a$) is defined to be $\int_a^b \mathrm{d}v\, P(v)$. $P(v)\mathrm{d}v$ is dimensionless. The density $P(v)$ is a dimensional quantity, having dimensions inverse to the dimensions of v – in contrast to discrete probabilities, which are dimensionless. Don't be surprised to see probability densities greater than 1. This is normal, and nothing is wrong, as long as $\int_a^b \mathrm{d}v\, P(v) \leq 1$ for any interval (a, b).

Conditional and joint probability densities are defined in just the same way as conditional and joint probabilities.

▷ Exercise 2.8.$^{[2]}$ Assuming a uniform prior on f_H, $P(f_H) = 1$, solve the problem posed in example 2.7 (p.30). Sketch the posterior distribution of f_H and compute the probability that the $N+1$th outcome will be a head, for

(a) $N = 3$ and $n_H = 0$;

(b) $N = 3$ and $n_H = 2$;

(c) $N = 10$ and $n_H = 3$;

(d) $N = 300$ and $n_H = 29$.

You will find the beta integral useful:

$$\int_0^1 \mathrm{d}p_a\, p_a^{F_a}(1-p_a)^{F_b} = \frac{\Gamma(F_a+1)\Gamma(F_b+1)}{\Gamma(F_a+F_b+2)} = \frac{F_a!F_b!}{(F_a+F_b+1)!}. \quad (2.32)$$

You may also find it instructive to look back at example 2.6 (p. 27) and equation (2.31).

People sometimes confuse assigning a prior distribution to an unknown parameter such as f_H with making an initial guess of the *value* of the parameter. But the prior over f_H, $P(f_H)$, is not a simple statement like 'initially, I would guess $f_H = 1/2$'. The prior is a probability density over f_H which specifies the prior degree of belief that f_H lies in any interval $(f, f + \delta f)$. It may well be the case that our prior for f_H is symmetric about $1/2$, so that the *mean* of f_H under the prior is $1/2$. In this case, the predictive distribution *for the first toss* x_1 would indeed be

$$P(x_1 = \text{head}) = \int \mathrm{d}f_H \; P(f_H) P(x_1 = \text{head} \,|\, f_H) = \int \mathrm{d}f_H \; P(f_H) f_H = 1/2. \tag{2.33}$$

But the prediction for subsequent tosses will depend on the whole prior distribution, not just its mean.

Data compression and inverse probability

Consider the following task.

Example 2.9. Write a computer program capable of compressing binary files like this one:

```
0000000000000000000010010001000000100000010000000000000000000000000000000000001010000000000000110000
1000000000010000100000000010000000000000000000000100000000000000000100000000011000001000000011000100
0000000001001000000000010001000000000000000011000000000000000000000000000000100000000000000001000000000
```

The string shown contains $n_1 = 29$ 1s and $n_0 = 271$ 0s.

Intuitively, compression works by taking advantage of the predictability of a file. In this case, the source of the file appears more likely to emit 0s than 1s. A data compression program that compresses this file must, implicitly or explicitly, be addressing the question 'What is the probability that the next character in this file is a 1?'

Do you think this problem is similar in character to example 2.7 (p. 30)? I do. One of the themes of this book is that data compression and data modelling are one and the same, and that they should both be addressed, like the urn of example 2.6, using inverse probability. Example 2.9 is solved in Chapter 6.

The likelihood principle

Please solve the following two exercises.

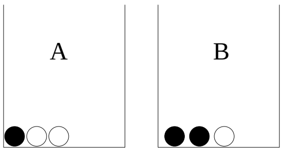


Figure 2.7. Urns for example 2.10.

Example 2.10. Urn A contains three balls: one black, and two white; urn B contains three balls: two black, and one white. One of the urns is selected at random and one ball is drawn. The ball is black. What is the probability that the selected urn is urn A?

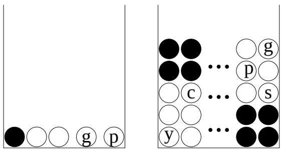


Figure 2.8. Urns for example 2.11.

Example 2.11. Urn A contains five balls: one black, two white, one green and one pink; urn B contains five hundred balls: two hundred black, one hundred white, 50 yellow, 40 cyan, 30 sienna, 25 green, 25 silver, 20 gold, and 10 purple. [One fifth of A's balls are black; two-fifths of B's are black.] One of the urns is selected at random and one ball is drawn. The ball is black. What is the probability that the urn is urn A?

What do you notice about your solutions? Does each answer depend on the detailed contents of each urn?

The details of the other possible outcomes and their probabilities are irrelevant. All that matters is the probability of the outcome that actually happened (here, that the ball drawn was black) given the different hypotheses. We need only to know the *likelihood*, i.e., how the probability of the data that happened varies with the hypothesis. This simple rule about inference is known as the *likelihood principle*.

> The likelihood principle: given a generative model for data d given parameters $\boldsymbol{\theta}$, $P(d \,|\, \boldsymbol{\theta})$, and having observed a particular outcome d_1, all inferences and predictions should depend only on the function $P(d_1 \,|\, \boldsymbol{\theta})$.

In spite of the simplicity of this principle, many classical statistical methods violate it.

i	a_i	p_i	$h(p_i)$
1	`a`	.0575	4.1
2	`b`	.0128	6.3
3	`c`	.0263	5.2
4	`d`	.0285	5.1
5	`e`	.0913	3.5
6	`f`	.0173	5.9
7	`g`	.0133	6.2
8	`h`	.0313	5.0
9	`i`	.0599	4.1
10	`j`	.0006	10.7
11	`k`	.0084	6.9
12	`l`	.0335	4.9
13	`m`	.0235	5.4
14	`n`	.0596	4.1
15	`o`	.0689	3.9
16	`p`	.0192	5.7
17	`q`	.0008	10.3
18	`r`	.0508	4.3
19	`s`	.0567	4.1
20	`t`	.0706	3.8
21	`u`	.0334	4.9
22	`v`	.0069	7.2
23	`w`	.0119	6.4
24	`x`	.0073	7.1
25	`y`	.0164	5.9
26	`z`	.0007	10.4
27	`-`	.1928	2.4
$\sum_i p_i \log_2 \frac{1}{p_i}$			4.1

Table 2.9. Shannon information contents of the outcomes `a`–`z`.

2.4 Definition of entropy and related functions

The Shannon information content of an outcome x is defined to be

$$h(x) = \log_2 \frac{1}{P(x)}. \tag{2.34}$$

It is measured in bits. [The word 'bit' is also used to denote a variable whose value is 0 or 1; I hope context will always make clear which of the two meanings is intended.]

In the next few chapters, we will establish that the Shannon information content $h(a_i)$ is indeed a natural measure of the information content of the event $x = a_i$. At that point, we will shorten the name of this quantity to 'the information content'.

The fourth column in table 2.9 shows the Shannon information content of the 27 possible outcomes when a random character is picked from an English document. The outcome $x = \texttt{z}$ has a Shannon information content of 10.4 bits, and $x = \texttt{e}$ has an information content of 3.5 bits.

The entropy of an ensemble X is defined to be the average Shannon information content of an outcome:

$$H(X) \equiv \sum_{x \in \mathcal{A}_X} P(x) \log \frac{1}{P(x)}, \tag{2.35}$$

with the convention for $P(x) = 0$ that $0 \times \log 1/0 \equiv 0$, since $\lim_{\theta \rightarrow 0^+} \theta \log 1/\theta = 0$.

Like the information content, entropy is measured in bits.

When it is convenient, we may also write $H(X)$ as $H(\mathbf{p})$, where $\mathbf{p}$ is the vector $(p_1, p_2, \ldots, p_I)$. Another name for the entropy of X is the uncertainty of X.

Example 2.12. The entropy of a randomly selected letter in an English document is about 4.11 bits, assuming its probability is as given in table 2.9. We obtain this number by averaging $\log 1/p_i$ (shown in the fourth column) under the probability distribution p_i (shown in the third column).

We now note some properties of the entropy function.

- $H(X) \geq 0$ with equality iff $p_i = 1$ for one i. ['iff' means 'if and only if'.]
- Entropy is maximized if $\mathbf{p}$ is uniform:

$$H(X) \leq \log(|\mathcal{A}_X|) \quad \text{with equality iff } p_i = 1/|\mathcal{A}_X| \text{ for all } i. \tag{2.36}$$

Notation: the vertical bars '$|\cdot|$' have two meanings. If $\mathcal{A}_X$ is a set, $|\mathcal{A}_X|$ denotes the number of elements in $\mathcal{A}_X$; if x is a number, then $|x|$ is the absolute value of x.

The *redundancy* measures the fractional difference between $H(X)$ and its maximum possible value, $\log(|\mathcal{A}_X|)$.

The redundancy of X is:

$$1 - \frac{H(X)}{\log|\mathcal{A}_X|}. \tag{2.37}$$

We won't make use of 'redundancy' in this book, so I have not assigned a symbol to it.

The joint entropy of X, Y is:

$$H(X,Y) = \sum_{xy \in \mathcal{A}_X \mathcal{A}_Y} P(x,y) \log \frac{1}{P(x,y)}. \tag{2.38}$$

Entropy is additive for independent random variables:

$$H(X,Y) = H(X) + H(Y) \;\text{ iff }\; P(x,y) = P(x)P(y). \tag{2.39}$$

Our definitions for information content so far apply only to discrete probability distributions over finite sets $\mathcal{A}_X$. The definitions can be extended to infinite sets, though the entropy may then be infinite. The case of a probability *density* over a continuous set is addressed in section 11.3. Further important definitions and exercises to do with entropy will come along in section 8.1.

▶ 2.5 Decomposability of the entropy

The entropy function satisfies a recursive property that can be very useful when computing entropies. For convenience, we'll stretch our notation so that we can write $H(X)$ as $H(\mathbf{p})$, where $\mathbf{p}$ is the probability vector associated with the ensemble X.

Let's illustrate the property by an example first. Imagine that a random variable $x \in \{0,1,2\}$ is created by first flipping a fair coin to determine whether $x = 0$; then, if x is not 0, flipping a fair coin a second time to determine whether x is 1 or 2. The probability distribution of x is

$$P(x\!=\!0) = \frac{1}{2};\;\; P(x\!=\!1) = \frac{1}{4};\;\; P(x\!=\!2) = \frac{1}{4}. \tag{2.40}$$

What is the entropy of X? We can either compute it by brute force:

$$H(X) = {}^1\!/\!_2 \log 2 + {}^1\!/\!_4 \log 4 + {}^1\!/\!_4 \log 4 = 1.5; \tag{2.41}$$

or we can use the following decomposition, in which the value of x is revealed gradually. Imagine first learning whether $x\!=\!0$, and then, if x is not 0, learning which non-zero value is the case. The revelation of whether $x\!=\!0$ or not entails revealing a binary variable whose probability distribution is $\{1/2, 1/2\}$. This revelation has an entropy $H(1/2, 1/2) = \frac{1}{2}\log 2 + \frac{1}{2}\log 2 = 1$ bit. If x is not 0, we learn the value of the second coin flip. This too is a binary variable whose probability distribution is $\{1/2, 1/2\}$, and whose entropy is 1 bit. We only get to experience the second revelation half the time, however, so the entropy can be written:

$$H(X) = H(1/2, 1/2) + 1/2\, H(1/2, 1/2). \tag{2.42}$$

Generalizing, the observation we are making about the entropy of any probability distribution $\mathbf{p} = \{p_1, p_2, \ldots, p_I\}$ is that

$$H(\mathbf{p}) = H(p_1, 1-p_1) + (1-p_1) H\left(\frac{p_2}{1-p_1}, \frac{p_3}{1-p_1}, \ldots, \frac{p_I}{1-p_1}\right). \tag{2.43}$$

When it's written as a formula, this property looks regrettably ugly; nevertheless it is a simple property and one that you should make use of.

Generalizing further, the entropy has the property for any m that

$$\begin{aligned} H(\mathbf{p}) \;=\; & H\left[(p_1 + p_2 + \cdots + p_m), (p_{m+1} + p_{m+2} + \cdots + p_I)\right] \\ & + (p_1 + \cdots + p_m) H\left(\frac{p_1}{(p_1 + \cdots + p_m)}, \ldots, \frac{p_m}{(p_1 + \cdots + p_m)}\right) \\ & + (p_{m+1} + \cdots + p_I) H\left(\frac{p_{m+1}}{(p_{m+1} + \cdots + p_I)}, \ldots, \frac{p_I}{(p_{m+1} + \cdots + p_I)}\right). \end{aligned} \tag{2.44}$$

Example 2.13. A source produces a character x from the alphabet $\mathcal{A} = \{\texttt{0}, \texttt{1}, \ldots, \texttt{9}, \texttt{a}, \texttt{b}, \ldots, \texttt{z}\}$; with probability $1/3$, x is a numeral $(\texttt{0}, \ldots, \texttt{9})$; with probability $1/3$, x is a vowel $(\texttt{a}, \texttt{e}, \texttt{i}, \texttt{o}, \texttt{u})$; and with probability $1/3$ it's one of the 21 consonants. All numerals are equiprobable, and the same goes for vowels and consonants. Estimate the entropy of X.

Solution. $\log 3 + \frac{1}{3}(\log 10 + \log 5 + \log 21) = \log 3 + \frac{1}{3}\log 1050 \simeq \log 30$ bits. □

▶ 2.6 Gibbs' inequality

The 'ei' in **Lei**bler is pronounced the same as in h**ei**st.

The relative entropy *or* Kullback–Leibler divergence between two probability distributions $P(x)$ and $Q(x)$ that are defined over the same alphabet $\mathcal{A}_X$ is

$$D_{\rm KL}(P||Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}. \tag{2.45}$$

The relative entropy satisfies *Gibbs' inequality*

$$D_{\rm KL}(P||Q) \geq 0 \tag{2.46}$$

with equality only if $P = Q$. Note that in general the relative entropy is not symmetric under interchange of the distributions P and Q: in general $D_{\rm KL}(P||Q) \neq D_{\rm KL}(Q||P)$, so $D_{\rm KL}$, although it is sometimes called the 'KL distance', is not strictly a distance. The relative entropy is important in pattern recognition and neural networks, as well as in information theory.

Gibbs' inequality is probably the most important inequality in this book. It, and many other inequalities, can be proved using the concept of convexity.

▶ 2.7 Jensen's inequality for convex functions

The words 'convex $\smile$' and 'concave $\frown$' may be pronounced 'convex-smile' and 'concave-frown'. This terminology has useful redundancy: while one may forget which way up 'convex' and 'concave' are, it is harder to confuse a smile with a frown.

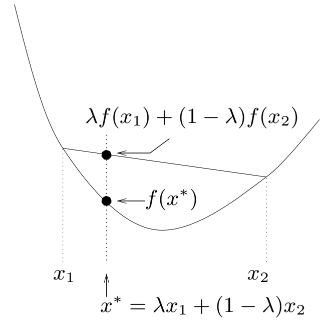

$\lambda f(x_1) + (1-\lambda) f(x_2)$
$f(x^*)$
x_1
x_2
$x^* = \lambda x_1 + (1-\lambda) x_2$

Figure 2.10. Definition of convexity.

Convex $\smile$ functions. A function $f(x)$ is *convex* $\smile$ over (a, b) if every chord of the function lies above the function, as shown in figure 2.10; that is, for all $x_1, x_2 \in (a, b)$ and $0 \leq \lambda \leq 1$,

$$f(\lambda x_1 + (1-\lambda)x_2) \ \leq \ \lambda f(x_1) + (1-\lambda) f(x_2). \qquad (2.47)$$

A function f is *strictly convex* $\smile$ if, for all $x_1, x_2 \in (a, b)$, the equality holds only for $\lambda = 0$ and $\lambda = 1$.

Similar definitions apply to concave $\frown$ and strictly concave $\frown$ functions.

Some strictly convex $\smile$ functions are

- x^2, e^x and e^{-x} for all x;
- $\log(1/x)$ and $x \log x$ for $x > 0$.

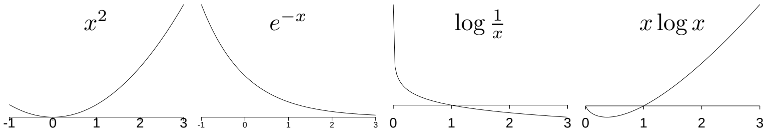


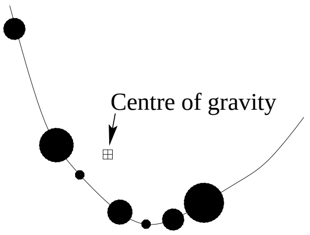
Figure 2.11. Convex $\smile$ functions.

Jensen's inequality. If f is a convex $\smile$ function and x is a random variable then:

$$\mathcal{E}\left[f(x)\right] \geq f(\mathcal{E}[x]), \qquad (2.48)$$

where $\mathcal{E}$ denotes expectation. If f is strictly convex $\smile$ and $\mathcal{E}\left[f(x)\right] = f(\mathcal{E}[x])$, then the random variable x is a constant.

Jensen's inequality can also be rewritten for a concave $\frown$ function, with the direction of the inequality reversed.



A physical version of Jensen's inequality runs as follows.

If a collection of masses p_i are placed on a convex $\smile$ curve $f(x)$ at locations $(x_i, f(x_i))$, then the centre of gravity of those masses, which is at $(\mathcal{E}[x], \mathcal{E}\left[f(x)\right])$, lies above the curve.

If this fails to convince you, then feel free to do the following exercise.

Exercise 2.14.[2, p.41] Prove Jensen's inequality.

Example 2.15. Three squares have average area $\bar{A} = 100\,\mathrm{m}^2$. The average of the lengths of their sides is $\bar{l} = 10\,\mathrm{m}$. What can be said about the size of the largest of the three squares? [Use Jensen's inequality.]

Solution. Let x be the length of the side of a square, and let the probability of x be $1/3, 1/3, 1/3$ over the three lengths l_1, l_2, l_3. Then the information that we have is that $\mathcal{E}\left[x\right] = 10$ and $\mathcal{E}\left[f(x)\right] = 100$, where $f(x) = x^2$ is the function mapping lengths to areas. This is a strictly convex $\smile$ function. We notice that the equality $\mathcal{E}\left[f(x)\right] = f(\mathcal{E}[x])$ holds, therefore x is a constant, and the three lengths must all be equal. The area of the largest square is $100\,\mathrm{m}^2$. □

Convexity and concavity also relate to maximization

If $f(\mathbf{x})$ is concave⌢ and there exists a point at which

$$\frac{\partial f}{\partial x_k} = 0 \text{ for all } k, \tag{2.49}$$

then $f(\mathbf{x})$ has its maximum value at that point.

The converse does not hold: if a concave⌢ $f(\mathbf{x})$ is maximized at some $\mathbf{x}$ it is not necessarily true that the gradient $\nabla f(\mathbf{x})$ is equal to zero there. For example, $f(x) = -|x|$ is maximized at $x = 0$ where its derivative is undefined; and $f(p) = \log(p)$, for a probability $p \in (0, 1)$, is maximized on the boundary of the range, at $p = 1$, where the gradient $\mathrm{d}f(p)/\mathrm{d}p = 1$.

▶ 2.8 Exercises

Sums of random variables

Exercise 2.16.[*3*, p.41] (a) Two ordinary dice with faces labelled $1, \ldots, 6$ are thrown. What is the probability distribution of the sum of the values? What is the probability distribution of the absolute difference between the values?

(b) One hundred ordinary dice are thrown. What, roughly, is the probability distribution of the sum of the values? Sketch the probability distribution and estimate its mean and standard deviation.

(c) How can two cubical dice be labelled using the numbers $\{0, 1, 2, 3, 4, 5, 6\}$ so that when the two dice are thrown the sum has a uniform probability distribution over the integers 1–12?

(d) Is there any way that one hundred dice could be labelled with integers such that the probability distribution of the sum is uniform?

This exercise is intended to help you think about the central-limit theorem, which says that if independent random variables $x_1, x_2, \ldots, x_N$ have means μ_n and finite variances σ_n^2, then, in the limit of large N, the sum $\sum_n x_n$ has a distribution that tends to a normal (Gaussian) distribution with mean $\sum_n \mu_n$ and variance $\sum_n \sigma_n^2$.

Inference problems

Exercise 2.17.[*2*, p.41] If $q = 1 - p$ and $a = \ln p/q$, show that

$$p = \frac{1}{1 + \exp(-a)}. \tag{2.50}$$

Sketch this function and find its relationship to the hyperbolic tangent function $\tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$.

It will be useful to be fluent in base-2 logarithms also. If $b = \log_2 p/q$, what is p as a function of b?

▷ Exercise 2.18.[*2*, p.42] Let x and y be dependent random variables with x a binary variable taking values in $\mathcal{A}_X = \{0, 1\}$. Use Bayes' theorem to show that the log posterior probability ratio for x given y is

$$\log \frac{P(x\!=\!1 \,|\, y)}{P(x\!=\!0 \,|\, y)} = \log \frac{P(y \,|\, x\!=\!1)}{P(y \,|\, x\!=\!0)} + \log \frac{P(x\!=\!1)}{P(x\!=\!0)}. \tag{2.51}$$

▷ Exercise 2.19.[*2*, p.42] Let x, d_1 and d_2 be random variables such that d_1 and d_2 are conditionally independent given a binary variable x. Use Bayes' theorem to show that the posterior probability ratio for x given $\{d_i\}$ is

$$\frac{P(x\!=\!1 \,|\, \{d_i\})}{P(x\!=\!0 \,|\, \{d_i\})} = \frac{P(d_1 \,|\, x\!=\!1)}{P(d_1 \,|\, x\!=\!0)} \frac{P(d_2 \,|\, x\!=\!1)}{P(d_2 \,|\, x\!=\!0)} \frac{P(x\!=\!1)}{P(x\!=\!0)}. \tag{2.52}$$

Life in high-dimensional spaces

Probability distributions and volumes have some unexpected properties in high-dimensional spaces.

Exercise 2.20.$^{[\textit{2}, \text{ p.42}]}$ Consider a sphere of radius r in an N-dimensional real space. Show that the fraction of the volume of the sphere that is in the surface shell lying at values of the radius between $r - \epsilon$ and r, where $0 < \epsilon < r$, is:

$$f = 1 - \left(1 - \frac{\epsilon}{r}\right)^N . \tag{2.53}$$

Evaluate f for the cases $N = 2$, $N = 10$ and $N = 1000$, with (a) $\epsilon/r = 0.01$; (b) $\epsilon/r = 0.5$.

Implication: points that are uniformly distributed in a sphere in N dimensions, where N is large, are very likely to be in a thin shell near the surface.

Expectations and entropies

You are probably familiar with the idea of computing the expectation of a function of x,

$$\mathcal{E}\left[f(x)\right] = \langle f(x) \rangle = \sum_x P(x) f(x). \tag{2.54}$$

Maybe you are not so comfortable with computing this expectation in cases where the function $f(x)$ depends on the probability $P(x)$. The next few examples address this concern.

Exercise 2.21.$^{[\textit{1}, \text{ p.43}]}$ Let $p_a = 0.1$, $p_b = 0.2$, and $p_c = 0.7$. Let $f(a) = 10$, $f(b) = 5$, and $f(c) = 10/7$. What is $\mathcal{E}\left[f(x)\right]$? What is $\mathcal{E}\left[1/P(x)\right]$?

Exercise 2.22.$^{[\textit{2}, \text{ p.43}]}$ For an arbitrary ensemble, what is $\mathcal{E}\left[1/P(x)\right]$?

▷ Exercise 2.23.$^{[\textit{1}, \text{ p.43}]}$ Let $p_a = 0.1$, $p_b = 0.2$, and $p_c = 0.7$. Let $g(a) = 0$, $g(b) = 1$, and $g(c) = 0$. What is $\mathcal{E}\left[g(x)\right]$?

▷ Exercise 2.24.$^{[\textit{1}, \text{ p.43}]}$ Let $p_a = 0.1$, $p_b = 0.2$, and $p_c = 0.7$. What is the probability that $P(x) \in [0.15, 0.5]$? What is

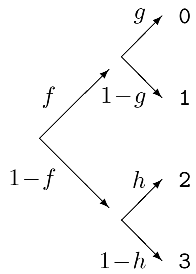
$$P\left(\left| \log \frac{P(x)}{0.2} \right| > 0.05 \right)?$$

Exercise 2.25.$^{[\textit{3}, \text{ p.43}]}$ Prove the assertion that $H(X) \leq \log(|\mathcal{A}_X|)$ with equality iff $p_i = 1/|\mathcal{A}_X|$ for all i. ($|\mathcal{A}_X|$ denotes the number of elements in the set $\mathcal{A}_X$.) [Hint: use Jensen's inequality (2.48); if your first attempt to use Jensen does not succeed, remember that Jensen involves both a random variable and a function, and you have quite a lot of freedom in choosing these; think about whether your chosen function f should be convex or concave.]

▷ Exercise 2.26.$^{[\textit{3}, \text{ p.44}]}$ Prove that the relative entropy (equation (2.45)) satisfies $D_{\mathrm{KL}}(P||Q) \geq 0$ (Gibbs' inequality) with equality only if $P = Q$.

▷ Exercise 2.27.$^{[\textit{2}]}$ Prove that the entropy is indeed decomposable as described in equations (2.43–2.44).

▷ Exercise 2.28.[2, p.45] A random variable $x \in \{0, 1, 2, 3\}$ is selected by flipping a bent coin with bias f to determine whether the outcome is in $\{0, 1\}$ or $\{2, 3\}$; then either flipping a second bent coin with bias g or a third bent coin with bias h respectively. Write down the probability distribution of x. Use the decomposability of the entropy (2.44) to find the entropy of X. [Notice how compact an expression is obtained if you make use of the binary entropy function $H_2(x)$, compared with writing out the four-term entropy explicitly.] Find the derivative of $H(X)$ with respect to f. [Hint: $\mathrm{d}H_2(x)/\mathrm{d}x = \log((1-x)/x)$.]



▷ Exercise 2.29.[2, p.45] An unbiased coin is flipped until one head is thrown. What is the entropy of the random variable $x \in \{1, 2, 3, \ldots\}$, the number of flips? Repeat the calculation for the case of a biased coin with probability f of coming up heads. [Hint: solve the problem both directly and by using the decomposability of the entropy (2.43).]

▶ 2.9 Further exercises

Forward probability

▷ Exercise 2.30.[1] An urn contains w white balls and b black balls. Two balls are drawn, one after the other, without replacement. Prove that the probability that the first ball is white is equal to the probability that the second is white.

▷ Exercise 2.31.[2] A circular coin of diameter a is thrown onto a square grid whose squares are $b \times b$. ($a < b$) What is the probability that the coin will lie entirely within one square? [Ans: $(1 - a/b)^2$]

▷ Exercise 2.32.[3] Buffon's needle. A needle of length a is thrown onto a plane covered with equally spaced parallel lines with separation b. What is the probability that the needle will cross a line? [Ans, if $a < b$: $^{2a}/_{\pi b}$] [Generalization – Buffon's noodle: on average, a random curve of length A is expected to intersect the lines $^{2A}/_{\pi b}$ times.]

Exercise 2.33.[2] Two points are selected at random on a straight line segment of length 1. What is the probability that a triangle can be constructed out of the three resulting segments?

Exercise 2.34.[2, p.45] An unbiased coin is flipped until one head is thrown. What is the expected number of tails and the expected number of heads?

Fred, who doesn't know that the coin is unbiased, estimates the bias using $\hat{f} \equiv h/(h+t)$, where h and t are the numbers of heads and tails tossed. Compute and sketch the probability distribution of $\hat{f}$.

N.B., this is a forward probability problem, a sampling theory problem, not an inference problem. Don't use Bayes' theorem.

Exercise 2.35.[2, p.45] Fred rolls an unbiased six-sided die once per second, noting the occasions when the outcome is a six.

(a) What is the mean number of rolls from one six to the next six?

(b) Between two rolls, the clock strikes one. What is the mean number of rolls until the next six?

(c) Now think back before the clock struck. What is the mean number of rolls, going back in time, until the most recent six?

(d) What is the mean number of rolls from the six before the clock struck to the next six?

(e) Is your answer to (d) different from your answer to (a)? Explain.

Another version of this exercise refers to Fred waiting for a bus at a bus-stop in Poissonville where buses arrive independently at random (a Poisson process), with, on average, one bus every six minutes. What is the average wait for a bus, after Fred arrives at the stop? [6 minutes.] So what is the time between the two buses, the one that Fred just missed, and the one that he catches? [12 minutes.] Explain the apparent paradox. Note the contrast with the situation in Clockville, where the buses are spaced exactly 6 minutes apart. There, as you can confirm, the mean wait at a bus-stop is 3 minutes, and the time between the missed bus and the next one is 6 minutes.

Conditional probability

▷ Exercise 2.36.[*2*] You meet Fred. Fred tells you he has two brothers, Alf and Bob.

What is the probability that Fred is older than Bob?

Fred tells you that he is older than Alf. Now, what is the probability that Fred is older than Bob? (That is, what is the conditional probability that $F > B$ given that $F > A$?)

▷ Exercise 2.37.[*2*] The inhabitants of an island tell the truth one third of the time. They lie with probability 2/3.

On an occasion, after one of them made a statement, you ask another 'was that statement true?' and he says 'yes'.

What is the probability that the statement was indeed true?

▷ Exercise 2.38.[*2*, p.46] Compare two ways of computing the probability of error of the repetition code R_3, assuming a binary symmetric channel (you did this once for exercise 1.2 (p.7)) and confirm that they give the same answer.

Binomial distribution method. Add the probability that all three bits are flipped to the probability that exactly two bits are flipped.

Sum rule method. Using the sum rule, compute the marginal probability that $\mathbf{r}$ takes on each of the eight possible values, $P(\mathbf{r})$. [$P(\mathbf{r}) = \sum_s P(s)P(\mathbf{r} \,|\, s)$.] Then compute the posterior probability of s for each of the eight values of $\mathbf{r}$. [In fact, by symmetry, only two example cases $\mathbf{r} = (\texttt{000})$ and $\mathbf{r} = (\texttt{001})$ need be considered.] Notice that some of the inferred bits are better determined than others. From the posterior probability $P(s \,|\, \mathbf{r})$ you can read out the case-by-case error probability, the probability that the more probable hypothesis is not correct, $P(\text{error} \,|\, \mathbf{r})$. Find the average error probability using the sum rule,

$$P(\text{error}) = \sum_{\mathbf{r}} P(\mathbf{r})P(\text{error} \,|\, \mathbf{r}). \tag{2.55}$$

Equation (1.18) gives the posterior probability of the input s, given the received vector $\mathbf{r}$.

▷ Exercise 2.39.$^{[3C, \text{ p.46}]}$ The frequency p_n of the nth most frequent word in English is roughly approximated by

$$p_n \simeq \begin{cases} \frac{0.1}{n} & \text{for } n \in 1, \ldots, 12\,367 \\ 0 & n > 12\,367. \end{cases} \tag{2.56}$$

[This remarkable $1/n$ law is known as Zipf's law, and applies to the word frequencies of many languages (Zipf, 1949).] If we assume that English is generated by picking words at random according to this distribution, what is the entropy of English (per word)? [This calculation can be found in 'Prediction and entropy of printed English', C.E. Shannon, *Bell Syst. Tech. J.* **30**, pp.50–64 (1950), but, inexplicably, the great man made numerical errors in it.]

▶ 2.10 Solutions

Solution to exercise 2.2 (p.24). No, they are not independent. If they were then all the conditional distributions $P(y \,|\, x)$ would be identical functions of y, regardless of x (cf. figure 2.3).

Solution to exercise 2.4 (p.27). We define the fraction $f_B \equiv B/K$.

(a) The number of black balls has a binomial distribution.

$$P(n_B \,|\, f_B, N) = \binom{N}{n_B} f_B^{n_B} (1 - f_B)^{N - n_B}. \tag{2.57}$$

(b) The mean and variance of this distribution are:

$$\mathcal{E}[n_B] = N f_B \tag{2.58}$$

$$\text{var}[n_B] = N f_B (1 - f_B). \tag{2.59}$$

These results were derived in example 1.1 (p.1). The standard deviation of n_B is $\sqrt{\text{var}[n_B]} = \sqrt{N f_B (1 - f_B)}$.

When $B/K = 1/5$ and $N = 5$, the expectation and variance of n_B are 1 and 4/5. The standard deviation is 0.89.

When $B/K = 1/5$ and $N = 400$, the expectation and variance of n_B are 80 and 64. The standard deviation is 8.

Solution to exercise 2.5 (p.27). The numerator of the quantity

$$z = \frac{(n_B - f_B N)^2}{N f_B (1 - f_B)}$$

can be recognized as $(n_B - \mathcal{E}[n_B])^2$; the denominator is equal to the variance of n_B (2.59), which is by definition the expectation of the numerator. So the expectation of z is 1. [A random variable like z, which measures the deviation of data from the expected value, is sometimes called χ^2 (chi-squared).]

In the case $N = 5$ and $f_B = 1/5$, $N f_B$ is 1, and $\text{var}[n_B]$ is 4/5. The numerator has five possible values, only one of which is smaller than 1: $(n_B - f_B N)^2 = 0$ has probability $P(n_B = 1) = 0.4096$; so the probability that $z < 1$ is 0.4096.

Solution to exercise 2.14 (p.35). We wish to prove, given the property

$$f(\lambda x_1 + (1-\lambda)x_2) \;\leq\; \lambda f(x_1) + (1-\lambda) f(x_2), \tag{2.60}$$

that, if $\sum p_i = 1$ and $p_i \geq 0$,

$$\sum_{i=1}^{I} p_i f(x_i) \geq f\left(\sum_{i=1}^{I} p_i x_i\right). \tag{2.61}$$

We proceed by recursion, working from the right-hand side. (This proof does not handle cases where some $p_i = 0$; such details are left to the pedantic reader.) At the first line we use the definition of convexity (2.60) with $\lambda = \frac{p_1}{\sum_{i=1}^{I} p_i} = p_1$; at the second line, $\lambda = \frac{p_2}{\sum_{i=2}^{I} p_i}$.

$$\begin{aligned} f\left(\sum_{i=1}^{I} p_i x_i\right) &= f\left(p_1 x_1 + \sum_{i=2}^{I} p_i x_i\right) \\ &\leq\; p_1 f(x_1) + \left[\sum_{i=2}^{I} p_i\right]\left[f\left(\sum_{i=2}^{I} p_i x_i \Big/ \sum_{i=2}^{I} p_i\right)\right] \\ &\leq\; p_1 f(x_1) + \left[\sum_{i=2}^{I} p_i\right]\left[\frac{p_2}{\sum_{i=2}^{I} p_i} f(x_2) + \frac{\sum_{i=3}^{I} p_i}{\sum_{i=2}^{I} p_i} f\left(\sum_{i=3}^{I} p_i x_i \Big/ \sum_{i=3}^{I} p_i\right)\right], \end{aligned} \tag{2.62}$$

and so forth. □

Solution to exercise 2.16 (p.36).

(a) For the outcomes $\{2,3,4,5,6,7,8,9,10,11,12\}$, the probabilities are $\mathcal{P} = \{\frac{1}{36}, \frac{2}{36}, \frac{3}{36}, \frac{4}{36}, \frac{5}{36}, \frac{6}{36}, \frac{5}{36}, \frac{4}{36}, \frac{3}{36}, \frac{2}{36}, \frac{1}{36}\}$.

(b) The value of one die has mean 3.5 and variance 35/12. So the sum of one hundred has mean 350 and variance $3500/12 \simeq 292$, and by the central-limit theorem the probability distribution is roughly Gaussian (but confined to the integers), with this mean and variance.

(c) In order to obtain a sum that has a uniform distribution we have to start from random variables some of which have a spiky distribution with the probability mass concentrated at the extremes. The unique solution is to have one ordinary die and one with faces 6, 6, 6, 0, 0, 0.

(d) Yes, a uniform distribution can be created in several ways, for example by labelling the rth die with the numbers $\{0,1,2,3,4,5\} \times 6^r$.

To think about: does this uniform distribution contradict the central-limit theorem?

Solution to exercise 2.17 (p.36).

$$a = \ln\frac{p}{q} \qquad \Rightarrow \qquad \frac{p}{q} = e^a \tag{2.63}$$

and $q = 1-p$ gives

$$\frac{p}{1-p} \;=\; e^a \tag{2.64}$$

$$\Rightarrow \qquad p \;=\; \frac{e^a}{e^a+1} = \frac{1}{1+\exp(-a)}. \tag{2.65}$$

The hyperbolic tangent is

$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \tag{2.66}$$

so

$$\begin{aligned} f(a) &\equiv \frac{1}{1+\exp(-a)} = \frac{1}{2}\left(\frac{1-e^{-a}}{1+e^{-a}}+1\right) \\ &= \frac{1}{2}\left(\frac{e^{a/2}-e^{-a/2}}{e^{a/2}+e^{-a/2}}+1\right) = \frac{1}{2}(\tanh(a/2)+1). \end{aligned} \tag{2.67}$$

In the case $b = \log_2 p/q$, we can repeat steps (2.63–2.65), replacing e by 2, to obtain

$$p = \frac{1}{1+2^{-b}}. \tag{2.68}$$

Solution to exercise 2.18 (p.36).

$$P(x\,|\,y) = \frac{P(y\,|\,x)P(x)}{P(y)} \tag{2.69}$$

$$\Rightarrow \frac{P(x\!=\!1\,|\,y)}{P(x\!=\!0\,|\,y)} = \frac{P(y\,|\,x\!=\!1)}{P(y\,|\,x\!=\!0)}\frac{P(x\!=\!1)}{P(x\!=\!0)} \tag{2.70}$$

$$\Rightarrow \log\frac{P(x\!=\!1\,|\,y)}{P(x\!=\!0\,|\,y)} = \log\frac{P(y\,|\,x\!=\!1)}{P(y\,|\,x\!=\!0)} + \log\frac{P(x\!=\!1)}{P(x\!=\!0)}. \tag{2.71}$$

Solution to exercise 2.19 (p.36). The conditional independence of d_1 and d_2 given x means

$$P(x, d_1, d_2) = P(x)P(d_1\,|\,x)P(d_2\,|\,x). \tag{2.72}$$

This gives a separation of the posterior probability ratio into a series of factors, one for each data point, times the prior probability ratio.

$$\frac{P(x\!=\!1\,|\,\{d_i\})}{P(x\!=\!0\,|\,\{d_i\})} = \frac{P(\{d_i\}\,|\,x\!=\!1)}{P(\{d_i\}\,|\,x\!=\!0)}\frac{P(x\!=\!1)}{P(x\!=\!0)} \tag{2.73}$$

$$= \frac{P(d_1\,|\,x\!=\!1)}{P(d_1\,|\,x\!=\!0)}\frac{P(d_2\,|\,x\!=\!1)}{P(d_2\,|\,x\!=\!0)}\frac{P(x\!=\!1)}{P(x\!=\!0)}. \tag{2.74}$$

Life in high-dimensional spaces

Solution to exercise 2.20 (p.37). The volume of a hypersphere of radius r in N dimensions is in fact

$$V(r, N) = \frac{\pi^{N/2}}{(N/2)!}r^N, \tag{2.75}$$

but you don't need to know this. For this question all that we need is the r-dependence, $V(r,N) \propto r^N$. So the fractional volume in $(r-\epsilon, r)$ is

$$\frac{r^N-(r-\epsilon)^N}{r^N} = 1-\left(1-\frac{\epsilon}{r}\right)^N. \tag{2.76}$$

The fractional volumes in the shells for the required cases are:

N	2	10	1000
$\epsilon/r = 0.01$	0.02	0.096	0.99996
$\epsilon/r = 0.5$	0.75	0.999	$1-2^{-1000}$

Notice that no matter how small ϵ is, for large enough N essentially all the probability mass is in the surface shell of thickness ϵ.

Solution to exercise 2.21 (p.37). $p_a = 0.1$, $p_b = 0.2$, $p_c = 0.7$. $f(a) = 10$, $f(b) = 5$, and $f(c) = 10/7$.

$$\mathcal{E}\left[f(x)\right] = 0.1 \times 10 + 0.2 \times 5 + 0.7 \times 10/7 = 3. \tag{2.77}$$

For each x, $f(x) = 1/P(x)$, so

$$\mathcal{E}\left[1/P(x)\right] = \mathcal{E}\left[f(x)\right] = 3. \tag{2.78}$$

Solution to exercise 2.22 (p.37). For general X,

$$\mathcal{E}\left[1/P(x)\right] = \sum_{x \in \mathcal{A}_X} P(x) 1/P(x) = \sum_{x \in \mathcal{A}_X} 1 = |\mathcal{A}_X|. \tag{2.79}$$

Solution to exercise 2.23 (p.37). $p_a = 0.1$, $p_b = 0.2$, $p_c = 0.7$. $g(a) = 0$, $g(b) = 1$, and $g(c) = 0$.

$$\mathcal{E}\left[g(x)\right] = p_b = 0.2. \tag{2.80}$$

Solution to exercise 2.24 (p.37).

$$P\left(P(x) \in [0.15, 0.5]\right) = p_b = 0.2. \tag{2.81}$$

$$P\left(\left|\log \frac{P(x)}{0.2}\right| > 0.05\right) = p_a + p_c = 0.8. \tag{2.82}$$

Solution to exercise 2.25 (p.37). This type of question can be approached in two ways: either by differentiating the function to be maximized, finding the maximum, and proving it is a global maximum; this strategy is somewhat risky since it is possible for the maximum of a function to be at the boundary of the space, at a place where the derivative is not zero. Alternatively, a carefully chosen inequality can establish the answer. The second method is much neater.

Proof by differentiation (not the recommended method). Since it is slightly easier to differentiate $\ln 1/p$ than $\log_2 1/p$, we temporarily define $H(X)$ to be measured using natural logarithms, thus scaling it down by a factor of $\log_2 e$.

$$H(X) = \sum_i p_i \ln \frac{1}{p_i} \tag{2.83}$$

$$\frac{\partial H(X)}{\partial p_i} = \ln \frac{1}{p_i} - 1 \tag{2.84}$$

we maximize subject to the constraint $\sum_i p_i = 1$ which can be enforced with a Lagrange multiplier:

$$G(\mathbf{p}) \equiv H(X) + \lambda \left(\sum_i p_i - 1\right) \tag{2.85}$$

$$\frac{\partial G(\mathbf{p})}{\partial p_i} = \ln \frac{1}{p_i} - 1 + \lambda. \tag{2.86}$$

At a maximum,

$$\ln \frac{1}{p_i} - 1 + \lambda = 0 \tag{2.87}$$

$$\Rightarrow \ln \frac{1}{p_i} = 1 - \lambda, \tag{2.88}$$

so all the p_i are equal. That this extremum is indeed a maximum is established by finding the curvature:

$$\frac{\partial^2 G(\mathbf{p})}{\partial p_i \partial p_j} = -\frac{1}{p_i} \delta_{ij}, \tag{2.89}$$

which is negative definite. □

Proof using Jensen's inequality (recommended method). First a reminder of the inequality.

> If f is a convex$\smile$ function and x is a random variable then:
>
> $$\mathcal{E}\left[f(x)\right] \geq f\left(\mathcal{E}[x]\right).$$
>
> If f is strictly convex$\smile$ and $\mathcal{E}\left[f(x)\right] = f\left(\mathcal{E}[x]\right)$, then the random variable x is a constant (with probability 1).

The secret of a proof using Jensen's inequality is to choose the right function and the right random variable. We could define

$$f(u) = \log\frac{1}{u} = -\log u \tag{2.90}$$

(which is a convex function) and think of $H(X) = \sum p_i \log \frac{1}{p_i}$ as the mean of $f(u)$ where $u = P(x)$, but this would not get us there – it would give us an inequality in the wrong direction. If instead we define

$$u = 1/P(x) \tag{2.91}$$

then we find:

$$H(X) = -\mathcal{E}\left[f(1/P(x))\right] \leq -f\left(\mathcal{E}[1/P(x)]\right); \tag{2.92}$$

now we know from exercise 2.22 (p.37) that $\mathcal{E}[1/P(x)] = |\mathcal{A}_X|$, so

$$H(X) \leq -f\left(|\mathcal{A}_X|\right) = \log|\mathcal{A}_X|. \tag{2.93}$$

Equality holds only if the random variable $u = 1/P(x)$ is a constant, which means $P(x)$ is a constant for all x. □

Solution to exercise 2.26 (p.37).

$$D_{\mathrm{KL}}(P||Q) = \sum_x P(x)\log\frac{P(x)}{Q(x)}. \tag{2.94}$$

We prove Gibbs' inequality using Jensen's inequality. Let $f(u) = \log 1/u$ and $u = \frac{Q(x)}{P(x)}$. Then

$$\begin{aligned} D_{\mathrm{KL}}(P||Q) &= \mathcal{E}[f(Q(x)/P(x))] & (2.95)\\ &\geq f\left(\sum_x P(x)\frac{Q(x)}{P(x)}\right) = \log\left(\frac{1}{\sum_x Q(x)}\right) = 0, & (2.96)\end{aligned}$$

with equality only if $u = \frac{Q(x)}{P(x)}$ is a constant, that is, if $Q(x) = P(x)$. □

Second solution. In the above proof the expectations were with respect to the probability distribution $P(x)$. A second solution method uses Jensen's inequality with $Q(x)$ instead. We define $f(u) = u\log u$ and let $u = \frac{P(x)}{Q(x)}$. Then

$$\begin{aligned} D_{\mathrm{KL}}(P||Q) &= \sum_x Q(x)\frac{P(x)}{Q(x)}\log\frac{P(x)}{Q(x)} = \sum_x Q(x) f\left(\frac{P(x)}{Q(x)}\right) & (2.97)\\ &\geq f\left(\sum_x Q(x)\frac{P(x)}{Q(x)}\right) = f(1) = 0, & (2.98)\end{aligned}$$

with equality only if $u = \frac{P(x)}{Q(x)}$ is a constant, that is, if $Q(x) = P(x)$. □

Solution to exercise 2.28 (p.38).

$$H(X) = H_2(f) + fH_2(g) + (1-f)H_2(h). \tag{2.99}$$

Solution to exercise 2.29 (p.38). The probability that there are $x-1$ tails and then one head (so we get the first head on the xth toss) is

$$P(x) = (1-f)^{x-1}f. \tag{2.100}$$

If the first toss is a tail, the probability distribution for the future looks just like it did before we made the first toss. Thus we have a recursive expression for the entropy:

$$H(X) = H_2(f) + (1-f)H(X). \tag{2.101}$$

Rearranging,

$$H(X) = H_2(f)/f. \tag{2.102}$$

Solution to exercise 2.34 (p.38). The probability of the number of tails t is

$$P(t) = \left(\frac{1}{2}\right)^t \frac{1}{2} \;\text{ for } t \geq 0. \tag{2.103}$$

The expected number of heads is 1, by definition of the problem. The expected number of tails is

$$\mathcal{E}[t] = \sum_{t=0}^{\infty} t \left(\frac{1}{2}\right)^t \frac{1}{2}, \tag{2.104}$$

which may be shown to be 1 in a variety of ways. For example, since the situation after one tail is thrown is equivalent to the opening situation, we can write down the recurrence relation

$$\mathcal{E}[t] = \frac{1}{2}(1+\mathcal{E}[t]) + \frac{1}{2}0 \;\Rightarrow\; \mathcal{E}[t] = 1. \tag{2.105}$$

The probability distribution of the 'estimator' $\hat{f} = 1/(1+t)$, given that $f = 1/2$, is plotted in figure 2.12. The probability of $\hat{f}$ is simply the probability of the corresponding value of t.

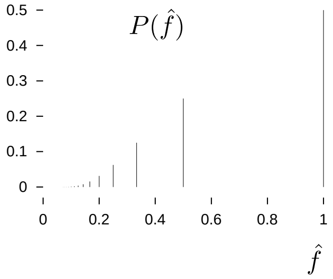


Figure 2.12. The probability distribution of the estimator $\hat{f} = 1/(1+t)$, given that $f = 1/2$.

Solution to exercise 2.35 (p.38).

(a) The mean number of rolls from one six to the next six is six (assuming we start counting rolls after the first of the two sixes). The probability that the next six occurs on the rth roll is the probability of *not* getting a six for $r-1$ rolls multiplied by the probability of then getting a six:

$$P(r_1 = r) = \left(\frac{5}{6}\right)^{r-1} \frac{1}{6}, \;\text{ for } r \in \{1,2,3,\ldots\}. \tag{2.106}$$

This probability distribution of the number of rolls, r, may be called an exponential distribution, since

$$P(r_1 = r) = e^{-\alpha r}/Z, \tag{2.107}$$

where $\alpha = \ln(6/5)$, and Z is a normalizing constant.

(b) The mean number of rolls from the clock until the next six is six.

(c) The mean number of rolls, going back in time, until the most recent six is six.

(d) The mean number of rolls from the six before the clock struck to the six after the clock struck is the sum of the answers to (b) and (c), less one, that is, eleven.

(e) Rather than explaining the difference between (a) and (d), let me give another hint. Imagine that the buses in Poissonville arrive independently at random (a Poisson process), with, on average, one bus every six minutes. Imagine that passengers turn up at bus-stops at a uniform rate, and are scooped up by the bus without delay, so the interval between two buses remains constant. Buses that follow gaps bigger than six minutes become overcrowded. The passengers' representative complains that two-thirds of all passengers found themselves on overcrowded buses. The bus operator claims, 'no, no – only one third of our buses are overcrowded'. Can both these claims be true?

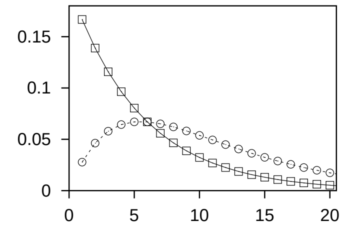


Figure 2.13. The probability distribution of the number of rolls r_1 from one 6 to the next (falling solid line),

$$P(r_1 = r) = \left(\frac{5}{6}\right)^{r-1} \frac{1}{6},$$

and the probability distribution (dashed line) of the number of rolls from the 6 before 1pm to the next 6, r_{tot},

$$P(r_{\text{tot}} = r) = r \left(\frac{5}{6}\right)^{r-1} \left(\frac{1}{6}\right)^2.$$

The probability $P(r_1 > 6)$ is about 1/3; the probability $P(r_{\text{tot}} > 6)$ is about 2/3. The mean of r_1 is 6, and the mean of r_{tot} is 11.

Solution to exercise 2.38 (p.39).

Binomial distribution method. From the solution to exercise 1.2, $p_B = 3f^2(1-f) + f^3$.

Sum rule method. The marginal probabilities of the eight values of $\mathbf{r}$ are illustrated by:

$$P(\mathbf{r} = 000) = 1/2(1-f)^3 + 1/2 f^3, \tag{2.108}$$

$$P(\mathbf{r} = 001) = 1/2 f(1-f)^2 + 1/2 f^2(1-f) = 1/2 f(1-f). \tag{2.109}$$

The posterior probabilities are represented by

$$P(s = 1 \mid \mathbf{r} = 000) = \frac{f^3}{(1-f)^3 + f^3} \tag{2.110}$$

and

$$P(s = 1 \mid \mathbf{r} = 001) = \frac{(1-f)f^2}{f(1-f)^2 + f^2(1-f)} = f. \tag{2.111}$$

The probabilities of error in these representative cases are thus

$$P(\text{error} \mid \mathbf{r} = 000) = \frac{f^3}{(1-f)^3 + f^3} \tag{2.112}$$

and

$$P(\text{error} \mid \mathbf{r} = 001) = f. \tag{2.113}$$

Notice that while the average probability of error of R_3 is about $3f^2$, the probability (given $\mathbf{r}$) that any *particular* bit is wrong is either about f^3 or f.

The average error probability, using the sum rule, is

$$\begin{aligned} P(\text{error}) &= \sum_{\mathbf{r}} P(\mathbf{r}) P(\text{error} \mid \mathbf{r}) \\ &= 2[1/2(1-f)^3 + 1/2 f^3] \frac{f^3}{(1-f)^3 + f^3} + 6[1/2 f(1-f)] f. \end{aligned}$$

The first two terms are for the cases $\mathbf{r} = 000$ and 111; the remaining 6 are for the other outcomes, which share the same probability of occurring and identical error probability, f.

So

$$P(\text{error}) = f^3 + 3f^2(1-f).$$

Solution to exercise 2.39 (p.40). The entropy is 9.7 bits per word.

About Chapter 3

If you are eager to get on to information theory, data compression, and noisy channels, you can skip to Chapter 4. Data compression and data modelling are intimately connected, however, so you'll probably want to come back to this chapter by the time you get to Chapter 6. Before reading Chapter 3, it might be good to look at the following exercises.

▷ Exercise 3.1.[*2*, p.59] A die is selected at random from two twenty-faced dice on which the symbols 1–10 are written with nonuniform frequency as follows.

Symbol	1	2	3	4	5	6	7	8	9	10
Number of faces of die A	6	4	3	2	1	1	1	1	1	0
Number of faces of die B	3	3	2	2	2	2	2	2	1	1

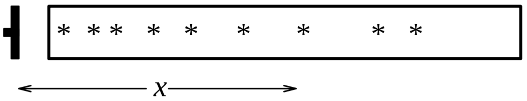
The randomly chosen die is rolled 7 times, with the following outcomes:

$$5, 3, 9, 3, 8, 4, 7.$$

What is the probability that the die is die A?

▷ Exercise 3.2.[*2*, p.59] Assume that there is a third twenty-faced die, die C, on which the symbols 1–20 are written once each. As above, one of the three dice is selected at random and rolled 7 times, giving the outcomes: 3, 5, 4, 8, 3, 9, 7.
What is the probability that the die is (a) die A, (b) die B, (c) die C?

Exercise 3.3.[*3*, p.48] Inferring a decay constant
Unstable particles are emitted from a source and decay at a distance x, a real number that has an exponential probability distribution with characteristic length λ. Decay events can be observed only if they occur in a window extending from $x = 1\,\text{cm}$ to $x = 20\,\text{cm}$. N decays are observed at locations $\{x_1, \ldots, x_N\}$. What is λ?



▷ Exercise 3.4.[*3*, p.55] Forensic evidence
Two people have left traces of their own blood at the scene of a crime. A suspect, Oliver, is tested and found to have type 'O' blood. The blood groups of the two traces are found to be of type 'O' (a common type in the local population, having frequency 60%) and of type 'AB' (a rare type, with frequency 1%). Do these data (type 'O' and 'AB' blood were found at scene) give evidence in favour of the proposition that Oliver was one of the two people present at the crime?

3

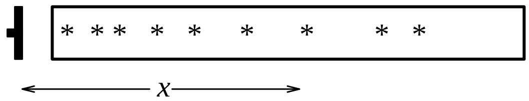
More about Inference

It is not a controversial statement that Bayes' theorem provides the correct language for describing the inference of a message communicated over a noisy channel, as we used it in Chapter 1 (p.6). But strangely, when it comes to other inference problems, the use of Bayes' theorem is not so widespread.

▶ 3.1 A first inference problem

When I was an undergraduate in Cambridge, I was privileged to receive supervisions from Steve Gull. Sitting at his desk in a dishevelled office in St. John's College, I asked him how one ought to answer an old Tripos question (exercise 3.3):

> Unstable particles are emitted from a source and decay at a distance x, a real number that has an exponential probability distribution with characteristic length λ. Decay events can be observed only if they occur in a window extending from $x = 1\,\text{cm}$ to $x = 20\,\text{cm}$. N decays are observed at locations $\{x_1, \ldots, x_N\}$. What is λ?



I had scratched my head over this for some time. My education had provided me with a couple of approaches to solving such inference problems: constructing 'estimators' of the unknown parameters; or 'fitting' the model to the data, or to a processed version of the data.

Since the mean of an unconstrained exponential distribution is λ, it seemed reasonable to examine the sample mean $\bar{x} = \sum_n x_n/N$ and see if an estimator $\hat{\lambda}$ could be obtained from it. It was evident that the estimator $\hat{\lambda} = \bar{x} - 1$ would be appropriate for $\lambda \ll 20\,\text{cm}$, but not for cases where the truncation of the distribution at the right-hand side is significant; with a little ingenuity and the introduction of ad hoc bins, promising estimators for $\lambda \gg 20$ cm could be constructed. But there was no obvious estimator that would work under all conditions.

Nor could I find a satisfactory approach based on fitting the density $P(x \mid \lambda)$ to a histogram derived from the data. I was stuck.

What is the general solution to this problem and others like it? Is it always necessary, when confronted by a new inference problem, to grope in the dark for appropriate 'estimators' and worry about finding the 'best' estimator (whatever that means)?

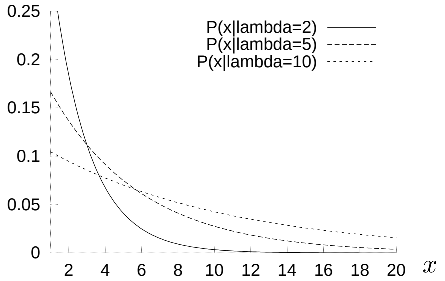


Figure 3.1. The probability density $P(x \mid \lambda)$ as a function of x.

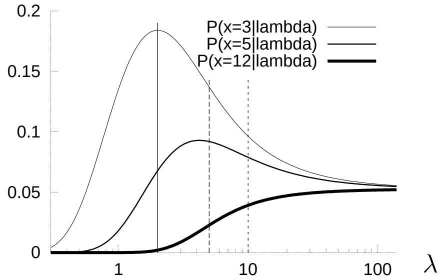


Figure 3.2. The probability density $P(x \mid \lambda)$ as a function of λ, for three different values of x. When plotted this way round, the function is known as the *likelihood* of λ. The marks indicate the three values of λ, $\lambda = 2, 5, 10$, that were used in the preceding figure.

Steve wrote down the probability of one data point, given λ:

$$P(x \mid \lambda) = \begin{cases} \frac{1}{\lambda} e^{-x/\lambda}/Z(\lambda) & 1 < x < 20 \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

where

$$Z(\lambda) = \int_1^{20} \mathrm{d}x \, \frac{1}{\lambda} e^{-x/\lambda} = \left(e^{-1/\lambda} - e^{-20/\lambda} \right). \tag{3.2}$$

This seemed obvious enough. Then he wrote *Bayes' theorem*:

$$\begin{aligned} P(\lambda \mid \{x_1, \ldots, x_N\}) &= \frac{P(\{x\} \mid \lambda) P(\lambda)}{P(\{x\})} & (3.3) \\ &\propto \frac{1}{(\lambda Z(\lambda))^N} \exp\left(-\textstyle\sum_1^N x_n / \lambda \right) P(\lambda). & (3.4) \end{aligned}$$

Suddenly, the straightforward distribution $P(\{x_1, \ldots, x_N\} \mid \lambda)$, defining the probability of the data given the hypothesis λ, was being turned on its head so as to define the probability of a hypothesis given the data. A simple figure showed the probability of a single data point $P(x \mid \lambda)$ as a familiar function of x, for different values of λ (figure 3.1). Each curve was an innocent exponential, normalized to have area 1. Plotting the same function as a function of λ for a fixed value of x, something remarkable happens: a peak emerges (figure 3.2). To help understand these two points of view of the one function, figure 3.3 shows a surface plot of $P(x \mid \lambda)$ as a function of x and λ.

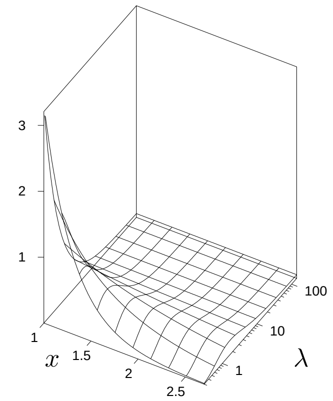


Figure 3.3. The probability density $P(x \mid \lambda)$ as a function of x and λ. Figures 3.1 and 3.2 are vertical sections through this surface.

For a dataset consisting of several points, e.g., the six points $\{x\}_{n=1}^N = \{1.5, 2, 3, 4, 5, 12\}$, the likelihood function $P(\{x\} \mid \lambda)$ is the product of the N functions of λ, $P(x_n \mid \lambda)$ (figure 3.4).

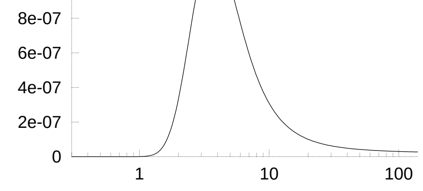


Figure 3.4. The likelihood function in the case of a six-point dataset, $P(\{x\} = \{1.5, 2, 3, 4, 5, 12\} \mid \lambda)$, as a function of λ.

Steve summarized Bayes' theorem as embodying the fact that

> what you know about λ after the data arrive is what you knew before $[P(\lambda)]$, and what the data told you $[P(\{x\}\,|\,\lambda)]$.

Probabilities are used here to quantify degrees of belief. To nip possible confusion in the bud, it must be emphasized that the hypothesis λ that correctly describes the situation is *not* a *stochastic* variable, and the fact that the Bayesian uses a probability distribution P does *not* mean that he thinks of the world as stochastically changing its nature between the states described by the different hypotheses. He uses the notation of probabilities to represent his *beliefs* about the mutually exclusive micro-hypotheses (here, values of λ), of which only one is actually true. That probabilities can denote degrees of belief, given assumptions, seemed reasonable to me.

The posterior probability distribution (3.4) represents the unique and complete solution to the problem. There is no need to invent 'estimators'; nor do we need to invent criteria for comparing alternative estimators with each other. Whereas orthodox statisticians offer twenty ways of solving a problem, and another twenty different criteria for deciding which of these solutions is the best, Bayesian statistics only offers one answer to a well-posed problem.

If you have any difficulty understanding this chapter I recommend ensuring you are happy with exercises 3.1 and 3.2 (p.47) then noting their similarity to exercise 3.3.

Assumptions in inference

Our inference is conditional on our assumptions [for example, the prior $P(\lambda)$]. Critics view such priors as a difficulty because they are 'subjective', but I don't see how it could be otherwise. How can one perform inference without making assumptions? I believe that it is of great value that Bayesian methods force one to make these tacit assumptions explicit.

First, once assumptions are made, the inferences are objective and unique, reproducible with complete agreement by anyone who has the same information and makes the same assumptions. For example, given the assumptions listed above, $\mathcal{H}$, and the data D, everyone will agree about the posterior probability of the decay length λ:

$$P(\lambda\,|\,D,\mathcal{H}) = \frac{P(D\,|\,\lambda,\mathcal{H})P(\lambda\,|\,\mathcal{H})}{P(D\,|\,\mathcal{H})}. \tag{3.5}$$

Second, when the assumptions are explicit, they are easier to criticize, and easier to modify – indeed, we can quantify the sensitivity of our inferences to the details of the assumptions. For example, we can note from the likelihood curves in figure 3.2 that in the case of a single data point at $x = 5$, the likelihood function is less strongly peaked than in the case $x = 3$; the details of the prior $P(\lambda)$ become increasingly important as the sample mean $\bar{x}$ gets closer to the middle of the window, 10.5. In the case $x = 12$, the likelihood function doesn't have a peak at all – such data merely rule out small values of λ, and don't give any information about the relative probabilities of large values of λ. So in this case, the details of the prior at the small–λ end of things are not important, but at the large–λ end, the prior is important.

Third, when we are not sure which of various alternative assumptions is the most appropriate for a problem, we can treat this question as another inference task. Thus, given data D, we can compare alternative assumptions $\mathcal{H}$ using Bayes' theorem:

$$P(\mathcal{H}\,|\,D,I) = \frac{P(D\,|\,\mathcal{H},I)P(\mathcal{H}\,|\,I)}{P(D\,|\,I)}, \tag{3.6}$$

where I denotes the highest assumptions, which we are not questioning.

Fourth, we can take into account our uncertainty regarding such assumptions when we make subsequent predictions. Rather than choosing one particular assumption $\mathcal{H}^*$, and working out our predictions about some quantity $\mathbf{t}$, $P(\mathbf{t} \,|\, D, \mathcal{H}^*, I)$, we obtain predictions that take into account our uncertainty about $\mathcal{H}$ by using the sum rule:

$$P(\mathbf{t} \,|\, D, I) = \sum_{\mathcal{H}} P(\mathbf{t} \,|\, D, \mathcal{H}, I) P(\mathcal{H} \,|\, D, I). \tag{3.7}$$

This is another contrast with orthodox statistics, in which it is conventional to 'test' a default model, and then, if the test 'accepts the model' at some 'significance level', to use exclusively that model to make predictions.

Steve thus persuaded me that

> probability theory reaches parts that ad hoc methods cannot reach.

Let's look at a few more examples of simple inference problems.

▶ 3.2 The bent coin

A bent coin is tossed F times; we observe a sequence $\mathbf{s}$ of heads and tails (which we'll denote by the symbols $\mathtt{a}$ and $\mathtt{b}$). We wish to know the bias of the coin, and predict the probability that the next toss will result in a head. We first encountered this task in example 2.7 (p.30), and we will encounter it again in Chapter 6, when we discuss adaptive data compression. It is also the original inference problem studied by Thomas Bayes in his essay published in 1763.

As in exercise 2.8 (p.30), we will assume a uniform prior distribution and obtain a posterior distribution by multiplying by the likelihood. A critic might object, 'where did this prior come from?' I will not claim that the uniform prior is in any way fundamental; indeed we'll give examples of nonuniform priors later. The prior is a subjective assumption. One of the themes of this book is:

> you can't do inference – or data compression – without making assumptions.

We give the name $\mathcal{H}_1$ to our assumptions. [We'll be introducing an alternative set of assumptions in a moment.] The probability, given $p_{\mathtt{a}}$, that F tosses result in a sequence $\mathbf{s}$ that contains $\{F_{\mathtt{a}}, F_{\mathtt{b}}\}$ counts of the two outcomes is

$$P(\mathbf{s} \,|\, p_{\mathtt{a}}, F, \mathcal{H}_1) = p_{\mathtt{a}}^{F_{\mathtt{a}}} (1 - p_{\mathtt{a}})^{F_{\mathtt{b}}}. \tag{3.8}$$

[For example, $P(\mathbf{s} = \mathtt{aaba} \,|\, p_{\mathtt{a}}, F = 4, \mathcal{H}_1) = p_{\mathtt{a}} p_{\mathtt{a}} (1 - p_{\mathtt{a}}) p_{\mathtt{a}}$.] Our first model assumes a uniform prior distribution for $p_{\mathtt{a}}$,

$$P(p_{\mathtt{a}} \,|\, \mathcal{H}_1) = 1, \quad p_{\mathtt{a}} \in [0, 1] \tag{3.9}$$

and $p_{\mathtt{b}} \equiv 1 - p_{\mathtt{a}}$.

Inferring unknown parameters

Given a string of length F of which $F_{\mathtt{a}}$ are $\mathtt{a}$s and $F_{\mathtt{b}}$ are $\mathtt{b}$s, we are interested in (a) inferring what $p_{\mathtt{a}}$ might be; (b) predicting whether the next character is an a or a b. [Predictions are always expressed as probabilities. So 'predicting whether the next character is an a' is the same as computing the probability that the next character is an a.]

Assuming $\mathcal{H}_1$ to be true, the posterior probability of p_{a}, given a string $\mathbf{s}$ of length F that has counts $\{F_{\mathsf{a}}, F_{\mathsf{b}}\}$, is, by Bayes' theorem,

$$P(p_{\mathsf{a}} \,|\, \mathbf{s}, F, \mathcal{H}_1) \;=\; \frac{P(\mathbf{s} \,|\, p_{\mathsf{a}}, F, \mathcal{H}_1) P(p_{\mathsf{a}} \,|\, \mathcal{H}_1)}{P(\mathbf{s} \,|\, F, \mathcal{H}_1)}. \tag{3.10}$$

The factor $P(\mathbf{s} \,|\, p_{\mathsf{a}}, F, \mathcal{H}_1)$, which, as a function of p_{a}, is known as the likelihood function, was given in equation (3.8); the prior $P(p_{\mathsf{a}} \,|\, \mathcal{H}_1)$ was given in equation (3.9). Our inference of p_{a} is thus:

$$P(p_{\mathsf{a}} \,|\, \mathbf{s}, F, \mathcal{H}_1) \;=\; \frac{p_{\mathsf{a}}^{F_{\mathsf{a}}}(1-p_{\mathsf{a}})^{F_{\mathsf{b}}}}{P(\mathbf{s} \,|\, F, \mathcal{H}_1)}. \tag{3.11}$$

The normalizing constant is given by the beta integral

$$P(\mathbf{s} \,|\, F, \mathcal{H}_1) = \int_0^1 \mathrm{d}p_{\mathsf{a}}\, p_{\mathsf{a}}^{F_{\mathsf{a}}}(1-p_{\mathsf{a}})^{F_{\mathsf{b}}} = \frac{\Gamma(F_{\mathsf{a}}+1)\Gamma(F_{\mathsf{b}}+1)}{\Gamma(F_{\mathsf{a}}+F_{\mathsf{b}}+2)} = \frac{F_{\mathsf{a}}! F_{\mathsf{b}}!}{(F_{\mathsf{a}}+F_{\mathsf{b}}+1)!}. \tag{3.12}$$

Exercise 3.5.[2, p.59] Sketch the posterior probability $P(p_{\mathsf{a}} \,|\, \mathbf{s} = \mathtt{aba}, F = 3)$. What is the most probable value of p_{a} (i.e., the value that maximizes the posterior probability density)? What is the mean value of p_{a} under this distribution?

Answer the same questions for the posterior probability $P(p_{\mathsf{a}} \,|\, \mathbf{s} = \mathtt{bbb}, F = 3)$.

From inferences to predictions

Our prediction about the next toss, the probability that the next toss is an a, is obtained by integrating over p_{a}. This has the effect of taking into account our uncertainty about p_{a} when making predictions. By the sum rule,

$$P(\mathsf{a} \,|\, \mathbf{s}, F) \;=\; \int \mathrm{d}p_{\mathsf{a}}\, P(\mathsf{a} \,|\, p_{\mathsf{a}}) P(p_{\mathsf{a}} \,|\, \mathbf{s}, F). \tag{3.13}$$

The probability of an a given p_{a} is simply p_{a}, so

$$P(\mathsf{a} \,|\, \mathbf{s}, F) = \int \mathrm{d}p_{\mathsf{a}}\, p_{\mathsf{a}} \frac{p_{\mathsf{a}}^{F_{\mathsf{a}}}(1-p_{\mathsf{a}})^{F_{\mathsf{b}}}}{P(\mathbf{s} \,|\, F)} \tag{3.14}$$

$$= \int \mathrm{d}p_{\mathsf{a}}\, \frac{p_{\mathsf{a}}^{F_{\mathsf{a}}+1}(1-p_{\mathsf{a}})^{F_{\mathsf{b}}}}{P(\mathbf{s} \,|\, F)} \tag{3.15}$$

$$= \left[\frac{(F_{\mathsf{a}}+1)!\, F_{\mathsf{b}}!}{(F_{\mathsf{a}}+F_{\mathsf{b}}+2)!}\right] \Big/ \left[\frac{F_{\mathsf{a}}!\, F_{\mathsf{b}}!}{(F_{\mathsf{a}}+F_{\mathsf{b}}+1)!}\right] \;=\; \frac{F_{\mathsf{a}}+1}{F_{\mathsf{a}}+F_{\mathsf{b}}+2}, \tag{3.16}$$

which is known as *Laplace's rule.*

▶ 3.3 The bent coin and model comparison

Imagine that a scientist introduces another theory for our data. He asserts that the source is not really a bent coin but is really a perfectly formed die with one face painted heads ('a') and the other five painted tails ('b'). Thus the parameter p_{a}, which in the original model, $\mathcal{H}_1$, could take any value between 0 and 1, is according to the new hypothesis, $\mathcal{H}_0$, not a free parameter at all; rather, it is equal to 1/6. [This hypothesis is termed $\mathcal{H}_0$ so that the suffix of each model indicates its number of free parameters.]

How can we compare these two models in the light of data? We wish to infer how probable $\mathcal{H}_1$ is relative to $\mathcal{H}_0$.

Model comparison as inference

In order to perform model comparison, we write down Bayes' theorem again, but this time with a different argument on the left-hand side. We wish to know how probable $\mathcal{H}_1$ is given the data. By Bayes' theorem,

$$P(\mathcal{H}_1 \,|\, \mathbf{s}, F) = \frac{P(\mathbf{s} \,|\, F, \mathcal{H}_1) P(\mathcal{H}_1)}{P(\mathbf{s} \,|\, F)}. \tag{3.17}$$

Similarly, the posterior probability of $\mathcal{H}_0$ is

$$P(\mathcal{H}_0 \,|\, \mathbf{s}, F) = \frac{P(\mathbf{s} \,|\, F, \mathcal{H}_0) P(\mathcal{H}_0)}{P(\mathbf{s} \,|\, F)}. \tag{3.18}$$

The normalizing constant in both cases is $P(\mathbf{s} \,|\, F)$, which is the total probability of getting the observed data. If $\mathcal{H}_1$ and $\mathcal{H}_0$ are the only models under consideration, this probability is given by the sum rule:

$$P(\mathbf{s} \,|\, F) = P(\mathbf{s} \,|\, F, \mathcal{H}_1) P(\mathcal{H}_1) + P(\mathbf{s} \,|\, F, \mathcal{H}_0) P(\mathcal{H}_0). \tag{3.19}$$

To evaluate the posterior probabilities of the hypotheses we need to assign values to the prior probabilities $P(\mathcal{H}_1)$ and $P(\mathcal{H}_0)$; in this case, we might set these to 1/2 each. And we need to evaluate the data-dependent terms $P(\mathbf{s} \,|\, F, \mathcal{H}_1)$ and $P(\mathbf{s} \,|\, F, \mathcal{H}_0)$. We can give names to these quantities. The quantity $P(\mathbf{s} \,|\, F, \mathcal{H}_1)$ is a measure of how much the data favour $\mathcal{H}_1$, and we call it the *evidence* for model $\mathcal{H}_1$. We already encountered this quantity in equation (3.10) where it appeared as the normalizing constant of the first inference we made – the inference of p_{a} given the data.

How model comparison works: The evidence for a model is usually the normalizing constant of an earlier Bayesian inference.

We evaluated the normalizing constant for model $\mathcal{H}_1$ in (3.12). The evidence for model $\mathcal{H}_0$ is very simple because this model has no parameters to infer. Defining p_0 to be 1/6, we have

$$P(\mathbf{s} \,|\, F, \mathcal{H}_0) = p_0^{F_{\mathsf{a}}} (1 - p_0)^{F_{\mathsf{b}}}. \tag{3.20}$$

Thus the posterior probability ratio of model $\mathcal{H}_1$ to model $\mathcal{H}_0$ is

$$\begin{aligned} \frac{P(\mathcal{H}_1 \,|\, \mathbf{s}, F)}{P(\mathcal{H}_0 \,|\, \mathbf{s}, F)} &= \frac{P(\mathbf{s} \,|\, F, \mathcal{H}_1) P(\mathcal{H}_1)}{P(\mathbf{s} \,|\, F, \mathcal{H}_0) P(\mathcal{H}_0)} & (3.21) \\ &= \frac{F_{\mathsf{a}}! F_{\mathsf{b}}!}{(F_{\mathsf{a}} + F_{\mathsf{b}} + 1)!} \Big/ p_0^{F_{\mathsf{a}}} (1 - p_0)^{F_{\mathsf{b}}}. & (3.22) \end{aligned}$$

Some values of this posterior probability ratio are illustrated in table 3.5. The first five lines illustrate that some outcomes favour one model, and some favour the other. No outcome is completely incompatible with either model. With small amounts of data (six tosses, say) it is typically not the case that one of the two models is overwhelmingly more probable than the other. But with more data, the evidence against $\mathcal{H}_0$ given by any data set with the ratio $F_{\mathsf{a}}\colon F_{\mathsf{b}}$ differing from 1: 5 mounts up. You can't predict in advance how much data are needed to be pretty sure which theory is true. It depends what p_{a} is.

The simpler model, $\mathcal{H}_0$, since it has no adjustable parameters, is able to lose out by the biggest margin. The odds may be hundreds to one against it. The more complex model can never lose out by a large margin; there's no data set that is actually *unlikely* given model $\mathcal{H}_1$.

F	Data $(F_\mathtt{a}, F_\mathtt{b})$	$\dfrac{P(\mathcal{H}_1 \,\vert\, \mathbf{s}, F)}{P(\mathcal{H}_0 \,\vert\, \mathbf{s}, F)}$	
6	$(5, 1)$	222.2	
6	$(3, 3)$	2.67	
6	$(2, 4)$	0.71	$= 1/1.4$
6	$(1, 5)$	0.356	$= 1/2.8$
6	$(0, 6)$	0.427	$= 1/2.3$
20	$(10, 10)$	96.5	
20	$(3, 17)$	0.2	$= 1/5$
20	$(0, 20)$	1.83	

Table 3.5. Outcome of model comparison between models $\mathcal{H}_1$ and $\mathcal{H}_0$ for the 'bent coin'. Model $\mathcal{H}_0$ states that $p_\mathtt{a} = 1/6$, $p_\mathtt{b} = 5/6$.

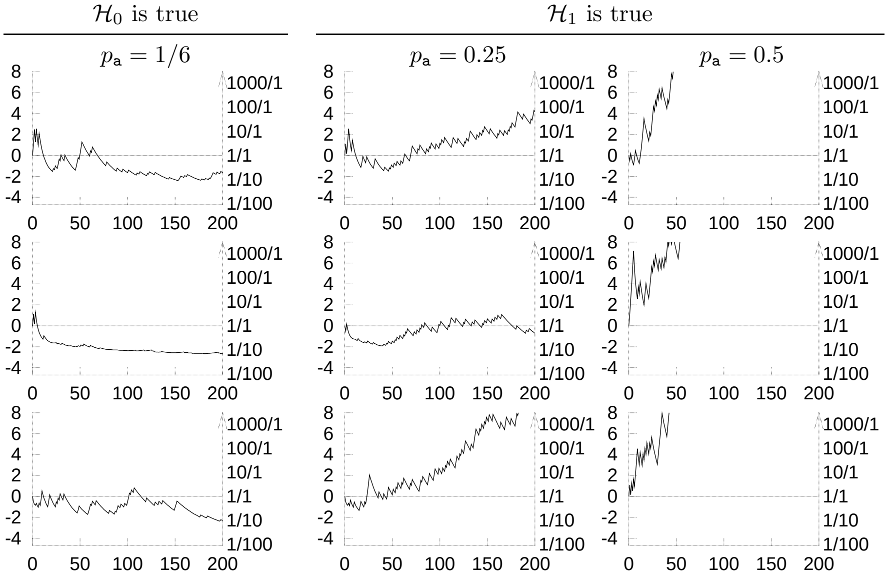


Figure 3.6. Typical behaviour of the evidence in favour of $\mathcal{H}_1$ as bent coin tosses accumulate under three different conditions (columns 1, 2, 3). Horizontal axis is the number of tosses, F. The vertical axis on the left is $\ln \frac{P(\mathbf{s} \,\vert\, F, \mathcal{H}_1)}{P(\mathbf{s} \,\vert\, F, \mathcal{H}_0)}$; the right-hand vertical axis shows the values of $\frac{P(\mathbf{s} \,\vert\, F, \mathcal{H}_1)}{P(\mathbf{s} \,\vert\, F, \mathcal{H}_0)}$.
The three rows show independent simulated experiments.
(See also figure 3.8, p.60.)

▷ Exercise 3.6.[2] Show that after F tosses have taken place, the biggest value that the log evidence ratio

$$\log \frac{P(\mathbf{s} \,\vert\, F, \mathcal{H}_1)}{P(\mathbf{s} \,\vert\, F, \mathcal{H}_0)} \tag{3.23}$$

can have scales *linearly* with F if $\mathcal{H}_1$ is more probable, but the log evidence in favour of $\mathcal{H}_0$ can grow at most as $\log F$.

▷ Exercise 3.7.[3, p.60] Putting your sampling theory hat on, assuming $F_\mathtt{a}$ has not yet been measured, compute a plausible range that the log evidence ratio might lie in, as a function of F and the true value of $p_\mathtt{a}$, and sketch it as a function of F for $p_\mathtt{a} = p_0 = 1/6$, $p_\mathtt{a} = 0.25$, and $p_\mathtt{a} = 1/2$. [Hint: sketch the log evidence as a function of the random variable $F_\mathtt{a}$ and work out the mean and standard deviation of $F_\mathtt{a}$.]

Typical behaviour of the evidence

Figure 3.6 shows the log evidence ratio as a function of the number of tosses, F, in a number of simulated experiments. In the left-hand experiments, $\mathcal{H}_0$ was true. In the right-hand ones, $\mathcal{H}_1$ was true, and the value of $p_\mathtt{a}$ was either 0.25 or 0.5.

We will discuss model comparison more in a later chapter.

▶ 3.4 An example of legal evidence

The following example illustrates that there is more to Bayesian inference than the priors.

> Two people have left traces of their own blood at the scene of a crime. A suspect, Oliver, is tested and found to have type 'O' blood. The blood groups of the two traces are found to be of type 'O' (a common type in the local population, having frequency 60%) and of type 'AB' (a rare type, with frequency 1%). Do these data (type 'O' and 'AB' blood were found at scene) give evidence in favour of the proposition that Oliver was one of the two people present at the crime?

A careless lawyer might claim that the fact that the suspect's blood type was found at the scene is positive evidence for the theory that he was present. But this is not so.

Denote the proposition 'the suspect and one unknown person were present' by S. The alternative, $\bar{S}$, states 'two unknown people from the population were present'. The prior in this problem is the prior probability ratio between the propositions S and $\bar{S}$. This quantity is important to the final verdict and would be based on all other available information in the case. Our task here is just to evaluate the contribution made by the data D, that is, the likelihood ratio, $P(D \,|\, S, \mathcal{H})/P(D \,|\, \bar{S}, \mathcal{H})$. In my view, a jury's task should generally be to multiply together carefully evaluated likelihood ratios from each independent piece of admissible evidence with an equally carefully reasoned prior probability. [This view is shared by many statisticians but learned British appeal judges recently disagreed and actually overturned the verdict of a trial because the jurors *had* been taught to use Bayes' theorem to handle complicated DNA evidence.]

The probability of the data given S is the probability that one unknown person drawn from the population has blood type AB:

$$P(D \,|\, S, \mathcal{H}) = p_{\rm AB} \tag{3.24}$$

(since given S, we already know that one trace will be of type O). The probability of the data given $\bar{S}$ is the probability that two unknown people drawn from the population have types O and AB:

$$P(D \,|\, \bar{S}, \mathcal{H}) = 2\, p_{\rm O}\, p_{\rm AB}. \tag{3.25}$$

In these equations $\mathcal{H}$ denotes the assumptions that two people were present and left blood there, and that the probability distribution of the blood groups of unknown people in an explanation is the same as the population frequencies.

Dividing, we obtain the likelihood ratio:

$$\frac{P(D \,|\, S, \mathcal{H})}{P(D \,|\, \bar{S}, \mathcal{H})} = \frac{1}{2p_{\rm O}} = \frac{1}{2 \times 0.6} = 0.83. \tag{3.26}$$

Thus the data in fact provide weak evidence *against* the supposition that Oliver was present.

This result may be found surprising, so let us examine it from various points of view. First consider the case of another suspect, Alberto, who has type AB. Intuitively, the data do provide evidence in favour of the theory S'

that this suspect was present, relative to the null hypothesis $\bar{S}$. And indeed the likelihood ratio in this case is:

$$\frac{P(D \mid S', \mathcal{H})}{P(D \mid \bar{S}, \mathcal{H})} = \frac{1}{2\,p_{\rm AB}} = 50. \tag{3.27}$$

Now let us change the situation slightly; imagine that 99% of people are of blood type O, and the rest are of type AB. Only these two blood types exist in the population. The data at the scene are the same as before. Consider again how these data influence our beliefs about Oliver, a suspect of type O, and Alberto, a suspect of type AB. Intuitively, we still believe that the presence of the rare AB blood provides positive evidence that Alberto was there. But does the fact that type O blood was detected at the scene favour the hypothesis that Oliver was present? If this were the case, that would mean that regardless of who the suspect is, the data make it more probable they were present; everyone in the population would be under greater suspicion, which would be absurd. The data may be *compatible* with any suspect of either blood type being present, but if they provide evidence *for* some theories, they must also provide evidence *against* other theories.

Here is another way of thinking about this: imagine that instead of two people's blood stains there are ten, and that in the entire local population of one hundred, there are ninety type O suspects and ten type AB suspects. Consider a particular type O suspect, Oliver: without any other information, and before the blood test results come in, there is a one in 10 chance that he was at the scene, since we know that 10 out of the 100 suspects were present. We now get the results of blood tests, and find that *nine* of the ten stains are of type AB, and *one* of the stains is of type O. Does this make it more likely that Oliver was there? No, there is now only a one in ninety chance that he was there, since we know that only one person present was of type O.

Maybe the intuition is aided finally by writing down the formulae for the general case where $n_{\rm O}$ blood stains of individuals of type O are found, and $n_{\rm AB}$ of type AB, a total of N individuals in all, and unknown people come from a large population with fractions $p_{\rm O}, p_{\rm AB}$. (There may be other blood types too.) The task is to evaluate the likelihood ratio for the two hypotheses: S, 'the type O suspect (Oliver) and $N-1$ unknown others left N stains'; and $\bar{S}$, 'N unknowns left N stains'. The probability of the data under hypothesis $\bar{S}$ is just the probability of getting $n_{\rm O}, n_{\rm AB}$ individuals of the two types when N individuals are drawn at random from the population:

$$P(n_{\rm O}, n_{\rm AB} \mid \bar{S}) = \frac{N!}{n_{\rm O}!\, n_{\rm AB}!} p_{\rm O}^{n_{\rm O}} p_{\rm AB}^{n_{\rm AB}}. \tag{3.28}$$

In the case of hypothesis S, we need the distribution of the $N-1$ other individuals:

$$P(n_{\rm O}, n_{\rm AB} \mid S) = \frac{(N-1)!}{(n_{\rm O}-1)!\, n_{\rm AB}!} p_{\rm O}^{n_{\rm O}-1} p_{\rm AB}^{n_{\rm AB}}. \tag{3.29}$$

The likelihood ratio is:

$$\frac{P(n_{\rm O}, n_{\rm AB} \mid S)}{P(n_{\rm O}, n_{\rm AB} \mid \bar{S})} = \frac{n_{\rm O}/N}{p_{\rm O}}. \tag{3.30}$$

This is an instructive result. The likelihood ratio, i.e. the contribution of these data to the question of whether Oliver was present, depends simply on a comparison of the frequency of his blood type in the observed data with the background frequency in the population. There is no dependence on the counts of the other types found at the scene, or their frequencies in the population.

If there are more type O stains than the average number expected under hypothesis $\bar{S}$, then the data give evidence in favour of the presence of Oliver. Conversely, if there are fewer type O stains than the expected number under $\bar{S}$, then the data reduce the probability of the hypothesis that he was there. In the special case $n_{\mathrm{O}}/N = p_{\mathrm{O}}$, the data contribute no evidence either way, regardless of the fact that the data are compatible with the hypothesis S.

▶ 3.5 Exercises

Exercise 3.8.[2, p.60] The three doors, normal rules.

On a game show, a contestant is told the rules as follows:

> There are three doors, labelled 1, 2, 3. A single prize has been hidden behind one of them. You get to select one door. Initially your chosen door will *not* be opened. Instead, the gameshow host will open one of the other two doors, and *he will do so in such a way as not to reveal the prize.* For example, if you first choose door 1, he will then open one of doors 2 and 3, and it is guaranteed that he will choose which one to open so that the prize will not be revealed.
>
> At this point, you will be given a fresh choice of door: you can either stick with your first choice, or you can switch to the other closed door. All the doors will then be opened and you will receive whatever is behind your final choice of door.

Imagine that the contestant chooses door 1 first; then the gameshow host opens door 3, revealing nothing behind the door, as promised. Should the contestant (a) stick with door 1, or (b) switch to door 2, or (c) does it make no difference?

Exercise 3.9.[2, p.61] The three doors, earthquake scenario.

Imagine that the game happens again and just as the gameshow host is about to open one of the doors a violent earthquake rattles the building and one of the three doors flies open. It happens to be door 3, and it happens not to have the prize behind it. The contestant had initially chosen door 1.

Repositioning his toupée, the host suggests, ‘OK, since you chose door 1 initially, door 3 is a valid door for me to open, according to the rules of the game; I’ll let door 3 stay open. Let’s carry on as if nothing happened.’

Should the contestant stick with door 1, or switch to door 2, or does it make no difference? Assume that the prize was placed randomly, that the gameshow host does not know where it is, and that the door flew open because its latch was broken by the earthquake.

[A similar alternative scenario is a gameshow whose *confused host* forgets the rules, and where the prize is, and opens one of the unchosen doors at random. He opens door 3, and the prize is not revealed. Should the contestant choose what’s behind door 1 or door 2? Does the optimal decision for the contestant depend on the contestant’s beliefs about whether the gameshow host is confused or not?]

▷ Exercise 3.10.[2] Another example in which the emphasis is not on priors. You visit a family whose three children are all at the local school. You don’t know anything about the sexes of the children. While walking clumsily round the home, you stumble through one of the three unlabelled bedroom doors that you know belong, one each, to the three children, and find that the bedroom contains girlie stuff in sufficient quantities to convince you that the child who lives in that bedroom is a girl. Later, you sneak a look at a letter addressed to the parents, which reads 'From the Headmaster: we are sending this letter to all parents who have male children at the school to inform them about the following boyish matters. . . '.

These two sources of evidence establish that at least one of the three children is a girl, and that at least one of the children is a boy. What are the probabilities that there are (a) two girls and one boy; (b) two boys and one girl?

▷ Exercise 3.11.[*2, p.61*] Mrs S is found stabbed in her family garden. Mr S behaves strangely after her death and is considered as a suspect. On investigation of police and social records it is found that Mr S had beaten up his wife on at least nine previous occasions. The prosecution advances this data as evidence in favour of the hypothesis that Mr S is guilty of the murder. 'Ah no,' says Mr S's highly paid lawyer, '*statistically*, only one in a thousand wife-beaters actually goes on to murder his wife.[1] So the wife-beating is not strong evidence at all. In fact, given the wife-beating evidence alone, it's extremely *unlikely* that he would be the murderer of his wife – only a 1/1000 chance. You should therefore find him innocent.'

Is the lawyer right to imply that the history of wife-beating does not point to Mr S's being the murderer? Or is the lawyer a slimy trickster? If the latter, what is wrong with his argument?

[Having received an indignant letter from a lawyer about the preceding paragraph, I'd like to add an extra inference exercise at this point: *Does my suggestion that Mr. S.'s lawyer may have been a slimy trickster imply that I believe* all *lawyers are slimy tricksters?* (Answer: No.)]

▷ Exercise 3.12.[*2*] A bag contains one counter, known to be either white or black. A white counter is put in, the bag is shaken, and a counter is drawn out, which proves to be white. What is now the chance of drawing a white counter? [Notice that the state of the bag, after the operations, is exactly identical to its state before.]

▷ Exercise 3.13.[*2, p.62*] You move into a new house; the phone is connected, and you're pretty sure that the phone number is `740511`, but not as sure as you would like to be. As an experiment, you pick up the phone and dial `740511`; you obtain a 'busy' signal. Are you now more sure of your phone number? If so, how much?

▷ Exercise 3.14.[*1*] In a game, two coins are tossed. If either of the coins comes up heads, you have won a prize. To claim the prize, you must point to one of your coins that is a head and say 'look, that coin's a head, I've won'. You watch Fred play the game. He tosses the two coins, and he

[1] In the U.S.A., it is estimated that 2 million women are abused each year by their partners. In 1994, 4739 women were victims of homicide; of those, 1326 women (28%) were slain by husbands and boyfriends.
(Sources: `http://www.umn.edu/mincava/papers/factoid.htm`,
`http://www.gunfree.inter.net/vpc/womenfs.htm`)

points to a coin and says 'look, that coin's a head, I've won'. What is the probability that the *other* coin is a head?

▷ Exercise 3.15.[2, p.63] A statistical statement appeared in *The Guardian* on Friday January 4, 2002:

> When spun on edge 250 times, a Belgian one-euro coin came up heads 140 times and tails 110. 'It looks very suspicious to me', said Barry Blight, a statistics lecturer at the London School of Economics. 'If the coin were unbiased the chance of getting a result as extreme as that would be less than 7%'.

But *do* these data give evidence that the coin is biased rather than fair? [Hint: see equation (3.22).]

▶ 3.6 Solutions

Solution to exercise 3.1 (p.47). Let the data be D. Assuming equal prior probabilities,

$$\frac{P(A\,|\,D)}{P(B\,|\,D)} = \frac{1}{2}\frac{3}{2}\frac{1}{1}\frac{3}{2}\frac{1}{2}\frac{2}{2}\frac{1}{2} = \frac{9}{32} \tag{3.31}$$

and $P(A\,|\,D) = 9/41$.

Solution to exercise 3.2 (p.47). The probability of the data given each hypothesis is:

$$P(D\,|\,A) = \frac{3}{20}\frac{1}{20}\frac{2}{20}\frac{1}{20}\frac{3}{20}\frac{1}{20}\frac{1}{20} = \frac{18}{20^7}; \tag{3.32}$$

$$P(D\,|\,B) = \frac{2}{20}\frac{2}{20}\frac{2}{20}\frac{2}{20}\frac{2}{20}\frac{1}{20}\frac{2}{20} = \frac{64}{20^7}; \tag{3.33}$$

$$P(D\,|\,C) = \frac{1}{20}\frac{1}{20}\frac{1}{20}\frac{1}{20}\frac{1}{20}\frac{1}{20}\frac{1}{20} = \frac{1}{20^7}. \tag{3.34}$$

So

$$P(A\,|\,D) = \frac{18}{18+64+1} = \frac{18}{83}; \qquad P(B\,|\,D) = \frac{64}{83}; \qquad P(C\,|\,D) = \frac{1}{83}. \tag{3.35}$$

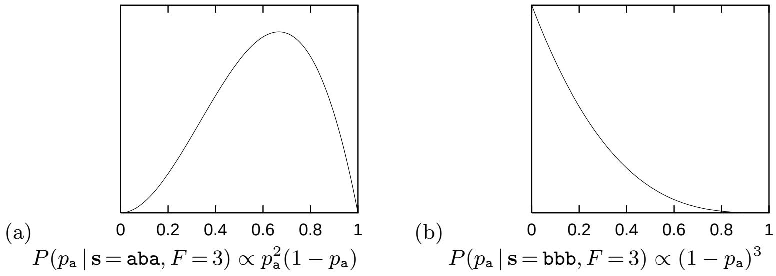


Figure 3.7. Posterior probability for the bias p_a of a bent coin given two different data sets.

Solution to exercise 3.5 (p.52).

(a) $P(p_a\,|\,s = aba, F = 3) \propto p_a^2(1 - p_a)$. The most probable value of p_a (i.e., the value that maximizes the posterior probability density) is 2/3. The mean value of p_a is 3/5.

See figure 3.7a.

(b) $P(p_{\mathtt{a}} \,|\, \mathbf{s} = \mathtt{bbb}, F = 3) \propto (1 - p_{\mathtt{a}})^3$. The most probable value of $p_{\mathtt{a}}$ (i.e., the value that maximizes the posterior probability density) is 0. The mean value of $p_{\mathtt{a}}$ is 1/5.

See figure 3.7b.

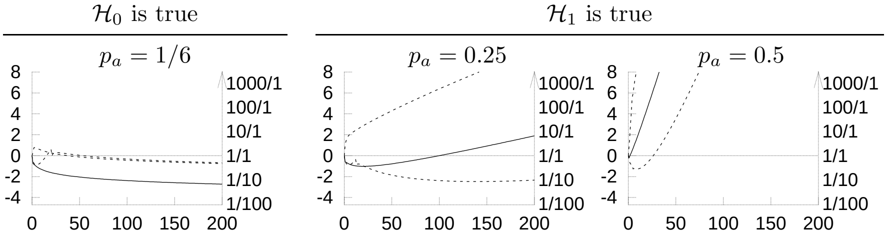


Figure 3.8. Range of plausible values of the log evidence in favour of $\mathcal{H}_1$ as a function of F. The vertical axis on the left is $\log \frac{P(\mathbf{s} \,|\, F, \mathcal{H}_1)}{P(\mathbf{s} \,|\, F, \mathcal{H}_0)}$; the right-hand vertical axis shows the values of $\frac{P(\mathbf{s} \,|\, F, \mathcal{H}_1)}{P(\mathbf{s} \,|\, F, \mathcal{H}_0)}$.
The solid line shows the log evidence if the random variable F_a takes on its mean value, $F_a = p_a F$. The dotted lines show (approximately) the log evidence if F_a is at its 2.5th or 97.5th percentile.
(See also figure 3.6, p.54.)

Solution to exercise 3.7 (p.54). The curves in figure 3.8 were found by finding the mean and standard deviation of F_a, then setting F_a to the mean ± two standard deviations to get a 95% plausible range for F_a, and computing the three corresponding values of the log evidence ratio.

Solution to exercise 3.8 (p.57). Let $\mathcal{H}_i$ denote the hypothesis that the prize is behind door i. We make the following assumptions: the three hypotheses $\mathcal{H}_1$, $\mathcal{H}_2$ and $\mathcal{H}_3$ are equiprobable *a priori*, i.e.,

$$P(\mathcal{H}_1) = P(\mathcal{H}_2) = P(\mathcal{H}_3) = \frac{1}{3}. \tag{3.36}$$

The datum we receive, after choosing door 1, is one of $D = 3$ and $D = 2$ (meaning door 3 or 2 is opened, respectively). We assume that these two possible outcomes have the following probabilities. If the prize is behind door 1 then the host has a free choice; in this case we assume that the host selects at random between $D = 2$ and $D = 3$. Otherwise the choice of the host is forced and the probabilities are 0 and 1.

$$\left| \begin{array}{l} P(D=2 \,|\, \mathcal{H}_1) = 1/2 \\ P(D=3 \,|\, \mathcal{H}_1) = 1/2 \end{array} \right| \begin{array}{l} P(D=2 \,|\, \mathcal{H}_2) = 0 \\ P(D=3 \,|\, \mathcal{H}_2) = 1 \end{array} \left| \begin{array}{l} P(D=2 \,|\, \mathcal{H}_3) = 1 \\ P(D=3 \,|\, \mathcal{H}_3) = 0 \end{array} \right| \tag{3.37}$$

Now, using Bayes' theorem, we evaluate the posterior probabilities of the hypotheses:

$$P(\mathcal{H}_i \,|\, D=3) = \frac{P(D=3 \,|\, \mathcal{H}_i) P(\mathcal{H}_i)}{P(D=3)} \tag{3.38}$$

$$\left| \; P(\mathcal{H}_1 \,|\, D=3) = \tfrac{(1/2)(1/3)}{P(D=3)} \; \right| \; P(\mathcal{H}_2 \,|\, D=3) = \tfrac{(1)(1/3)}{P(D=3)} \; \left| \; P(\mathcal{H}_3 \,|\, D=3) = \tfrac{(0)(1/3)}{P(D=3)} \; \right| \tag{3.39}$$

The denominator $P(D = 3)$ is (1/2) because it is the normalizing constant for this posterior distribution. So

$$\left| \; P(\mathcal{H}_1 \,|\, D=3) \;=\; 1/3 \; \right| \; P(\mathcal{H}_2 \,|\, D=3) \;=\; 2/3 \; \left| \; P(\mathcal{H}_3 \,|\, D=3) \;=\; 0. \; \right| \tag{3.40}$$

So the contestant should switch to door 2 in order to have the biggest chance of getting the prize.

Many people find this outcome surprising. There are two ways to make it more intuitive. One is to play the game thirty times with a friend and keep track of the frequency with which switching gets the prize. Alternatively, you can perform a thought experiment in which the game is played with a million doors. The rules are now that the contestant chooses one door, then the game show host opens 999,998 doors in such a way as not to reveal the prize, leaving the *contestant's* selected door and *one other door* closed. The contestant may now stick or switch. Imagine the contestant confronted by a million doors, of which doors 1 and 234,598 have not been opened, door 1 having been the contestant's initial guess. Where do you think the prize is?

Solution to exercise 3.9 (p.57). If door 3 is opened by an earthquake, the inference comes out differently – even though visually the scene looks the same. The nature of the data, and the probability of the data, are both now different. The possible data outcomes are, firstly, that any number of the doors might have opened. We could label the eight possible outcomes $\mathbf{d} = (0,0,0), (0,0,1), (0,1,0), (1,0,0), (0,1,1), \ldots, (1,1,1)$. Secondly, it might be that the prize is visible after the earthquake has opened one or more doors. So the data D consists of the value of $\mathbf{d}$, and a statement of whether the prize was revealed. It is hard to say what the probabilities of these outcomes are, since they depend on our beliefs about the reliability of the door latches and the properties of earthquakes, but it is possible to extract the desired posterior probability without naming the values of $P(\mathbf{d} \,|\, \mathcal{H}_i)$ for each $\mathbf{d}$. All that matters are the relative values of the quantities $P(D \,|\, \mathcal{H}_1)$, $P(D \,|\, \mathcal{H}_2)$, $P(D \,|\, \mathcal{H}_3)$, for the value of D that actually occurred. [This is the *likelihood principle*, which we met in section 2.3.] The value of D that actually occurred is '$\mathbf{d} = (0,0,1)$, and no prize visible'. First, it is clear that $P(D \,|\, \mathcal{H}_3) = 0$, since the datum that no prize is visible is incompatible with $\mathcal{H}_3$. Now, assuming that the contestant selected door 1, how does the probability $P(D \,|\, \mathcal{H}_1)$ compare with $P(D \,|\, \mathcal{H}_2)$? Assuming that earthquakes are not sensitive to decisions of game show contestants, these two quantities have to be equal, by symmetry. We don't know how likely it is that door 3 falls off its hinges, but however likely it is, it's just as likely to do so whether the prize is behind door 1 or door 2. So, if $P(D \,|\, \mathcal{H}_1)$ and $P(D \,|\, \mathcal{H}_2)$ are equal, we obtain:

$$\left| \begin{array}{r} P(\mathcal{H}_1|D) = \frac{P(D|\mathcal{H}_1)(1/3)}{P(D)} \\ = 1/2 \end{array} \right| \begin{array}{r} P(\mathcal{H}_2|D) = \frac{P(D|\mathcal{H}_2)(1/3)}{P(D)} \\ = 1/2 \end{array} \left| \begin{array}{r} P(\mathcal{H}_3|D) = \frac{P(D|\mathcal{H}_3)(1/3)}{P(D)} \\ = 0. \end{array} \right| \tag{3.41}$$

The two possible hypotheses are now equally likely.

If we assume that the host knows where the prize is and might be acting deceptively, then the answer might be further modified, because we have to view the host's words as part of the data.

Confused? It's well worth making sure you understand these two gameshow problems. Don't worry, I slipped up on the second problem, the first time I met it.

There is a general rule which helps immensely when you have a confusing probability problem:

> Always write down the probability of everything.
> *(Steve Gull)*

From this joint probability, any desired inference can be mechanically obtained (figure 3.9).

Which doors opened by earthquake	Where the prize is		
	door 1	door 2	door 3
none	$\frac{p_{\text{none}}}{3}$	$\frac{p_{\text{none}}}{3}$	$\frac{p_{\text{none}}}{3}$
1			
2			
3	$\frac{p_3}{3}$	$\frac{p_3}{3}$	$\frac{p_3}{3}$
1,2			
1,3			
2,3			
1,2,3	$\frac{p_{1,2,3}}{3}$	$\frac{p_{1,2,3}}{3}$	$\frac{p_{1,2,3}}{3}$

Figure 3.9. The probability of everything, for the second three-door problem, assuming an earthquake has just occurred. Here, p_3 is the probability that door 3 alone is opened by an earthquake.

Solution to exercise 3.11 (p.58). The statistic quoted by the lawyer indicates the probability that a randomly selected wife-beater will also murder his wife. The probability that the husband was the murderer, *given that the wife has been murdered*, is a completely different quantity.

To deduce the latter, we need to make further assumptions about the probability that the wife is murdered by someone else. If she lives in a neighbourhood with frequent random murders, then this probability is large and the posterior probability that the husband did it (in the absence of other evidence) may not be very large. But in more peaceful regions, it may well be that the most likely person to have murdered you, if you are found murdered, is one of your closest relatives.

Let's work out some illustrative numbers with the help of the statistics on page 58. Let $m = 1$ denote the proposition that a woman has been murdered; $h = 1$, the proposition that the husband did it; and $b = 1$, the proposition that he beat her in the year preceding the murder. The statement 'someone else did it' is denoted by $h = 0$. We need to define $P(h \mid m = 1)$, $P(b \mid h = 1, m = 1)$, and $P(b = 1 \mid h = 0, m = 1)$ in order to compute the posterior probability $P(h = 1 \mid b = 1, m = 1)$. From the statistics, we can read out $P(h = 1 \mid m = 1) = 0.28$. And if two million women out of 100 million are beaten, then $P(b = 1 \mid h = 0, m = 1) = 0.02$. Finally, we need a value for $P(b \mid h = 1, m = 1)$: if a man murders his wife, how likely is it that this is the first time he laid a finger on her? I expect it's pretty unlikely; so maybe $P(b = 1 \mid h = 1, m = 1)$ is 0.9 or larger.

By Bayes' theorem, then,

$$P(h = 1 \mid b = 1, m = 1) = \frac{.9 \times .28}{.9 \times .28 + .02 \times .72} \simeq 95\%. \tag{3.42}$$

One way to make obvious the sliminess of the lawyer on p.58 is to construct arguments, with the same logical structure as his, that are clearly wrong. For example, the lawyer could say 'Not only was Mrs. S murdered, she was murdered between 4.02pm and 4.03pm. *Statistically*, only one in a *million* wife-beaters actually goes on to murder his wife between 4.02pm and 4.03pm. So the wife-beating is not strong evidence at all. In fact, given the wife-beating evidence alone, it's extremely unlikely that he would murder his wife in this way – only a 1/1,000,000 chance.'

Solution to exercise 3.13 (p.58). There are two hypotheses. $\mathcal{H}_0$: your number is 740511; $\mathcal{H}_1$: it is another number. The data, D, are 'when I dialed 740511, I got a busy signal'. What is the probability of D, given each hypothesis? If your number is 740511, then we expect a busy signal with certainty:

$$P(D \mid \mathcal{H}_0) = 1.$$

On the other hand, if $\mathcal{H}_1$ is true, then the probability that the number dialled returns a busy signal is smaller than 1, since various other outcomes were also possible (a ringing tone, or a number-unobtainable signal, for example). The value of this probability $P(D \mid \mathcal{H}_1)$ will depend on the probability α that a random phone number similar to your own phone number would be a valid phone number, and on the probability β that you get a busy signal when you dial a valid phone number.

I estimate from the size of my phone book that Cambridge has about 75 000 valid phone numbers, all of length six digits. The probability that a random six-digit number is valid is therefore about $75\,000/10^6 = 0.075$. If we exclude numbers beginning with 0, 1, and 9 from the random choice, the probability α is about $75\,000/700\,000 \simeq 0.1$. If we assume that telephone numbers are clustered then a misremembered number might be more likely to be valid than a randomly chosen number; so the probability, α, that our guessed number would be valid, assuming $\mathcal{H}_1$ is true, might be bigger than

0.1. Anyway, α must be somewhere between 0.1 and 1. We can carry forward this uncertainty in the probability and see how much it matters at the end.

The probability β that you get a busy signal when you dial a valid phone number is equal to the fraction of phones you think are in use or off-the-hook when you make your tentative call. This fraction varies from town to town and with the time of day. In Cambridge, during the day, I would guess that about 1% of phones are in use. At 4am, maybe 0.1%, or fewer.

The probability $P(D \,|\, \mathcal{H}_1)$ is the product of α and β, that is, about $0.1 \times 0.01 = 10^{-3}$. According to our estimates, there's about a one-in-a-thousand chance of getting a busy signal when you dial a random number; or one-in-a-hundred, if valid numbers are strongly clustered; or one-in-10^4, if you dial in the wee hours.

How do the data affect your beliefs about your phone number? The posterior probability ratio is the likelihood ratio times the prior probability ratio:

$$\frac{P(\mathcal{H}_0 \,|\, D)}{P(\mathcal{H}_1 \,|\, D)} = \frac{P(D \,|\, \mathcal{H}_0)}{P(D \,|\, \mathcal{H}_1)} \frac{P(\mathcal{H}_0)}{P(\mathcal{H}_1)}. \tag{3.43}$$

The likelihood ratio is about 100-to-1 or 1000-to-1, so the posterior probability ratio is swung by a factor of 100 or 1000 in favour of $\mathcal{H}_0$. If the prior probability of $\mathcal{H}_0$ was 0.5 then the posterior probability is

$$P(\mathcal{H}_0 \,|\, D) = \frac{1}{1 + \frac{P(\mathcal{H}_1 \,|\, D)}{P(\mathcal{H}_0 \,|\, D)}} \simeq 0.99 \text{ or } 0.999. \tag{3.44}$$

Solution to exercise 3.15 (p.59). We compare the models $\mathcal{H}_0$ – the coin is fair – and $\mathcal{H}_1$ – the coin is biased, with the prior on its bias set to the uniform distribution $P(p|\mathcal{H}_1) = 1$. [The use of a uniform prior seems reasonable to me, since I know that some coins, such as American pennies, have severe biases when spun on edge; so the situations $p = 0.01$ or $p = 0.1$ or $p = 0.95$ would not surprise me.]

> When I mention $\mathcal{H}_0$ – the coin is fair – a pedant would say, 'how absurd to even consider that the coin is fair – any coin is surely biased to some extent'. And of course I would agree. So will pedants kindly understand $\mathcal{H}_0$ as meaning 'the coin is fair to within one part in a thousand, i.e., $p \in 0.5 \pm 0.001$'.

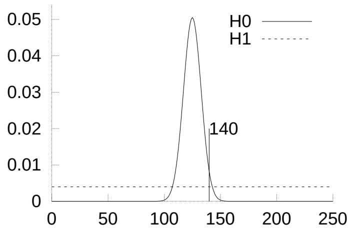


Figure 3.10. The probability distribution of the number of heads given the two hypotheses, that the coin is fair, and that it is biased, with the prior distribution of the bias being uniform. The outcome ($D = 140$ heads) gives weak evidence in favour of $\mathcal{H}_0$, the hypothesis that the coin is fair.

The likelihood ratio is:

$$\frac{P(D|\mathcal{H}_1)}{P(D|\mathcal{H}_0)} = \frac{\frac{140!110!}{251!}}{1/2^{250}} = 0.48. \tag{3.45}$$

Thus the data give scarcely any evidence either way; in fact they give weak evidence (two to one) in favour of $\mathcal{H}_0$!

'No, no', objects the believer in bias, 'your silly uniform prior doesn't represent *my* prior beliefs about the bias of biased coins – I was *expecting* only a small bias'. To be as generous as possible to the $\mathcal{H}_1$, let's see how well it could fare if the prior were presciently set. Let us allow a prior of the form

$$P(p|\mathcal{H}_1, \alpha) = \frac{1}{Z(\alpha)} p^{\alpha-1}(1-p)^{\alpha-1}, \quad \text{where } Z(\alpha) = \Gamma(\alpha)^2/\Gamma(2\alpha) \tag{3.46}$$

(a Beta distribution, with the original uniform prior reproduced by setting $\alpha = 1$). By tweaking α, the likelihood ratio for $\mathcal{H}_1$ over $\mathcal{H}_0$,

$$\frac{P(D|\mathcal{H}_1, \alpha)}{P(D|\mathcal{H}_0)} = \frac{\Gamma(140{+}\alpha)\,\Gamma(110{+}\alpha)\,\Gamma(2\alpha)2^{250}}{\Gamma(250{+}2\alpha)\,\Gamma(\alpha)^2}, \tag{3.47}$$

can be increased a little. It is shown for several values of α in figure 3.11. Even the most favourable choice of α ($\alpha \simeq 50$) can yield a likelihood ratio of only two to one in favour of $\mathcal{H}_1$.

In conclusion, the data are not 'very suspicious'. They can be construed as giving at most two-to-one evidence in favour of one or other of the two hypotheses.

> Are these wimpy likelihood ratios the fault of over-restrictive priors? Is there any way of producing a 'very suspicious' conclusion? The prior that is best-matched to the data, in terms of likelihood, is the prior that sets p to $f \equiv 140/250$ with probability one. Let's call this model $\mathcal{H}_*$. The likelihood ratio is $P(D|\mathcal{H}_*)/P(D|\mathcal{H}_0) = 2^{250} f^{140}(1-f)^{110} = 6.1$. So the strongest evidence that these data can possibly muster against the hypothesis that there is no bias is six-to-one.

α	$\dfrac{P(D\|\mathcal{H}_1,\alpha)}{P(D\|\mathcal{H}_0)}$
.37	.25
1.0	.48
2.7	.82
7.4	1.3
20	1.8
55	1.9
148	1.7
403	1.3
1096	1.1

Figure 3.11. Likelihood ratio for various choices of the prior distribution's hyperparameter α.

While we are noticing the absurdly misleading answers that 'sampling theory' statistics produces, such as the p-value of 7% in the exercise we just solved, let's stick the boot in. If we make a tiny change to the data set, increasing the number of heads in 250 tosses from 140 to 141, we find that the p-value goes below the mystical value of 0.05 (the p-value is 0.0497). The sampling theory statistician would happily squeak 'the probability of getting a result as extreme as 141 heads is smaller than 0.05 – we thus reject the null hypothesis at a significance level of 5%'. The correct answer is shown for several values of α in figure 3.12. The values worth highlighting from this table are, first, the likelihood ratio when $\mathcal{H}_1$ uses the standard uniform prior, which is 1:0.61 in favour of the *null hypothesis* $\mathcal{H}_0$. Second, the most favourable choice of α, from the point of view of $\mathcal{H}_1$, can only yield a likelihood ratio of about 2.3:1 in favour of $\mathcal{H}_1$.

Be warned! A p-value of 0.05 is often interpreted as implying that the odds are stacked about twenty-to-one *against* the null hypothesis. But the truth in this case is that the evidence either slightly *favours* the null hypothesis, or disfavours it by at most 2.3 to one, depending on the choice of prior.

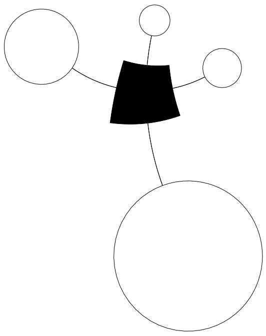
The p-values and 'significance levels' of classical statistics should be treated with *extreme caution*. Shun them! Here ends the sermon.

α	$\dfrac{P(D'\|\mathcal{H}_1,\alpha)}{P(D'\|\mathcal{H}_0)}$
.37	.32
1.0	.61
2.7	1.0
7.4	1.6
20	2.2
55	2.3
148	1.9
403	1.4
1096	1.2

Figure 3.12. Likelihood ratio for various choices of the prior distribution's hyperparameter α, when the data are $D' = 141$ heads in 250 trials.

Part I

Data Compression



About Chapter 4

In this chapter we discuss how to measure the information content of the outcome of a random experiment.

This chapter has some tough bits. If you find the mathematical details hard, skim through them and keep going – you'll be able to enjoy Chapters 5 and 6 without this chapter's tools.

Before reading Chapter 4, you should have read Chapter 2 and worked on exercises 2.21–2.25 and 2.16 (pp.36–37), and exercise 4.1 below.

The following exercise is intended to help you think about how to measure information content.

Notation	
$x \in \mathcal{A}$	x is a *member* of the set $\mathcal{A}$
$\mathcal{S} \subset \mathcal{A}$	$\mathcal{S}$ is a *subset* of the set $\mathcal{A}$
$\mathcal{S} \subseteq \mathcal{A}$	$\mathcal{S}$ is a subset of, or equal to, the set $\mathcal{A}$
$\mathcal{V} = \mathcal{B} \cup \mathcal{A}$	$\mathcal{V}$ is the *union* of the sets $\mathcal{B}$ and $\mathcal{A}$
$\mathcal{V} = \mathcal{B} \cap \mathcal{A}$	$\mathcal{V}$ is the *intersection* of the sets $\mathcal{B}$ and $\mathcal{A}$
$\lvert\mathcal{A}\rvert$	number of elements in set $\mathcal{A}$

Exercise 4.1.[*2*, p.69] – *Please work on this problem before reading Chapter 4.*

You are given 12 balls, all equal in weight except for one that is either heavier or lighter. You are also given a two-pan balance to use. In each use of the balance you may put any number of the 12 balls on the left pan, and the same number on the right pan, and push a button to initiate the weighing; there are three possible outcomes: either the weights are equal, or the balls on the left are heavier, or the balls on the left are lighter. Your task is to design a strategy to determine which is the odd ball *and* whether it is heavier or lighter than the others *in as few uses of the balance as possible.*

While thinking about this problem, you may find it helpful to consider the following questions:

(a) How can one measure *information*?

(b) When you have identified the odd ball and whether it is heavy or light, how much information have you gained?

(c) Once you have designed a strategy, draw a tree showing, for each of the possible outcomes of a weighing, what weighing you perform next. At each node in the tree, how much information have the outcomes so far given you, and how much information remains to be gained?

(d) How much information is gained when you learn (i) the state of a flipped coin; (ii) the states of two flipped coins; (iii) the outcome when a four-sided die is rolled?

(e) How much information is gained on the first step of the weighing problem if 6 balls are weighed against the other 6? How much is gained if 4 are weighed against 4 on the first step, leaving out 4 balls?

4

The Source Coding Theorem

▶ 4.1 How to measure the information content of a random variable?

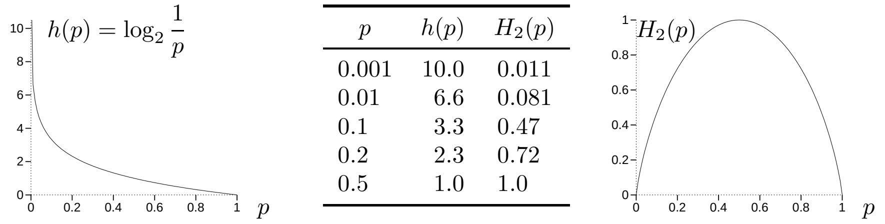
In the next few chapters, we'll be talking about probability distributions and random variables. Most of the time we can get by with sloppy notation, but occasionally, we will need precise notation. Here is the notation that we established in Chapter 2.

An ensemble X is a triple $(x, \mathcal{A}_X, \mathcal{P}_X)$, where the *outcome* x is the value of a random variable, which takes on one of a set of possible values, $\mathcal{A}_X = \{a_1, a_2, \ldots, a_i, \ldots, a_I\}$, having probabilities $\mathcal{P}_X = \{p_1, p_2, \ldots, p_I\}$, with $P(x\!=\!a_i) = p_i$, $p_i \geq 0$ and $\sum_{a_i \in \mathcal{A}_X} P(x\!=\!a_i) = 1$.

How can we measure the information content of an outcome $x = a_i$ from such an ensemble? In this chapter we examine the assertions

1. that the *Shannon information content*,
$$h(x\!=\!a_i) \equiv \log_2 \frac{1}{p_i}, \tag{4.1}$$
is a sensible measure of the information content of the outcome $x = a_i$, and

2. that the *entropy* of the ensemble,
$$H(X) = \sum_i p_i \log_2 \frac{1}{p_i}, \tag{4.2}$$
is a sensible measure of the ensemble's average information content.



p	$h(p)$	$H_2(p)$
0.001	10.0	0.011
0.01	6.6	0.081
0.1	3.3	0.47
0.2	2.3	0.72
0.5	1.0	1.0

Figure 4.1. The Shannon information content $h(p) = \log_2 \frac{1}{p}$ and the binary entropy function $H_2(p) = H(p, 1\!-\!p) = p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{(1-p)}$ as a function of p.

Figure 4.1 shows the Shannon information content of an outcome with probability p, as a function of p. The less probable an outcome is, the greater its Shannon information content. Figure 4.1 also shows the binary entropy function,

$$H_2(p) = H(p, 1\!-\!p) = p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{(1-p)}, \tag{4.3}$$

which is the entropy of the ensemble X whose alphabet and probability distribution are $\mathcal{A}_X = \{a, b\}, \mathcal{P}_X = \{p, (1-p)\}$.

Information content of independent random variables

Why should $\log 1/p_i$ have anything to do with the information content? Why not some other function of p_i? We'll explore this question in detail shortly, but first, notice a nice property of this particular function $h(x) = \log 1/p(x)$.

Imagine learning the value of two *independent* random variables, x and y. The definition of independence is that the probability distribution is separable into a *product*:

$$P(x, y) = P(x)P(y). \tag{4.4}$$

Intuitively, we might want any measure of the 'amount of information gained' to have the property of *additivity* – that is, for independent random variables x and y, the information gained when we learn x and y should equal the sum of the information gained if x alone were learned and the information gained if y alone were learned.

The Shannon information content of the outcome x, y is

$$h(x, y) = \log \frac{1}{P(x, y)} = \log \frac{1}{P(x)P(y)} = \log \frac{1}{P(x)} + \log \frac{1}{P(y)} \tag{4.5}$$

so it does indeed satisfy

$$h(x, y) = h(x) + h(y), \text{ if } x \text{ and } y \text{ are independent.} \tag{4.6}$$

Exercise 4.2.$^{[1, \text{ p.86}]}$ Show that, if x and y are independent, the entropy of the outcome x, y satisfies

$$H(X, Y) = H(X) + H(Y). \tag{4.7}$$

In words, entropy is additive for independent variables.

We now explore these ideas with some examples; then, in section 4.4 and in Chapters 5 and 6, we prove that the Shannon information content and the entropy are related to the number of bits needed to describe the outcome of an experiment.

The weighing problem: designing informative experiments

Have you solved the weighing problem (exercise 4.1, p.66) yet? Are you sure? Notice that in three uses of the balance – which reads either 'left heavier', 'right heavier', or 'balanced' – the number of conceivable outcomes is $3^3 = 27$, whereas the number of possible states of the world is 24: the odd ball could be any of twelve balls, and it could be heavy or light. So in principle, the problem might be solvable in three weighings – but not in two, since $3^2 < 24$.

If you know how you can determine the odd weight *and* whether it is heavy or light in *three* weighings, then you may read on. If you haven't found a strategy that always gets there in three weighings, I encourage you to think about exercise 4.1 some more.

Why is your strategy optimal? What is it about your series of weighings that allows useful information to be gained as quickly as possible? The answer is that at each step of an optimal procedure, the three outcomes ('left heavier', 'right heavier', and 'balance') are *as close as possible to equiprobable.* An optimal solution is shown in figure 4.2.

Suboptimal strategies, such as weighing balls 1–6 against 7–12 on the first step, do not achieve all outcomes with equal probability: these two sets of balls can never balance, so the only possible outcomes are 'left heavy' and 'right heavy'. Such a binary outcome rules out only half of the possible hypotheses,

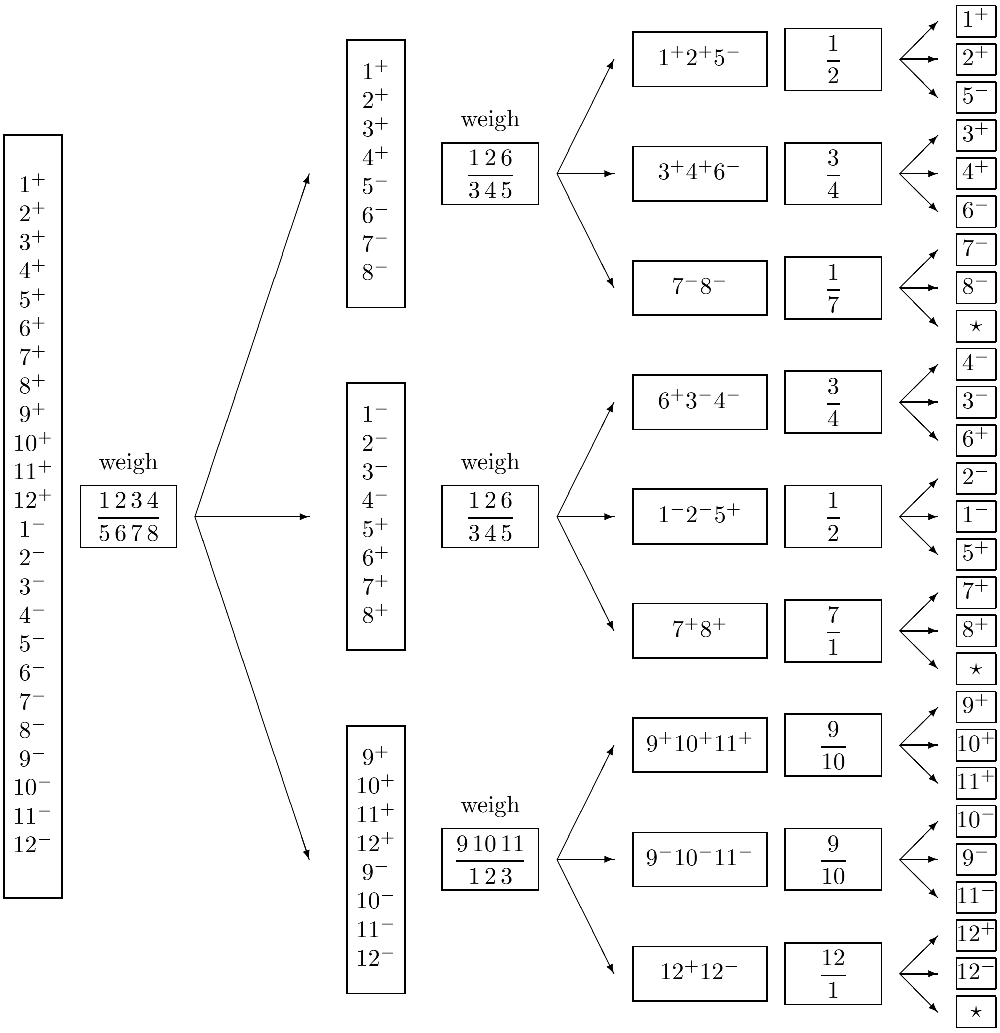


Figure 4.2. An optimal solution to the weighing problem. At each step there are two boxes: the left box shows which hypotheses are still possible; the right box shows the balls involved in the next weighing. The 24 hypotheses are written $1^+, \ldots, 12^-$, with, e.g., 1^+ denoting that 1 is the odd ball and it is heavy. Weighings are written by listing the names of the balls on the two pans, separated by a line; for example, in the first weighing, balls 1, 2, 3, and 4 are put on the left-hand side and 5, 6, 7, and 8 on the right. In each triplet of arrows the upper arrow leads to the situation when the left side is heavier, the middle arrow to the situation when the right side is heavier, and the lower arrow to the situation when the outcome is balanced. The three points labelled $\star$ correspond to impossible outcomes.

so a strategy that uses such outcomes must sometimes take longer to find the right answer.

The insight that the outcomes should be as near as possible to equiprobable makes it easier to search for an optimal strategy. The first weighing must divide the 24 possible hypotheses into three groups of eight. Then the second weighing must be chosen so that there is a 3:3:2 split of the hypotheses.

Thus we might conclude:

> the outcome of a random experiment is guaranteed to be most informative if the probability distribution over outcomes is uniform.

This conclusion agrees with the property of the entropy that you proved when you solved exercise 2.25 (p.37): the entropy of an ensemble X is biggest if all the outcomes have equal probability $p_i = 1/|\mathcal{A}_X|$.

Guessing games

In the game of twenty questions, one player thinks of an object, and the other player attempts to guess what the object is by asking questions that have yes/no answers, for example, 'is it alive?', or 'is it human?' The aim is to identify the object with as few questions as possible. What is the best strategy for playing this game? For simplicity, imagine that we are playing the rather dull version of twenty questions called 'sixty-three'.

Example 4.3. The game 'sixty-three'. What's the smallest number of yes/no questions needed to identify an integer x between 0 and 63?

Intuitively, the best questions successively divide the 64 possibilities into equal sized sets. Six questions suffice. One reasonable strategy asks the following questions:

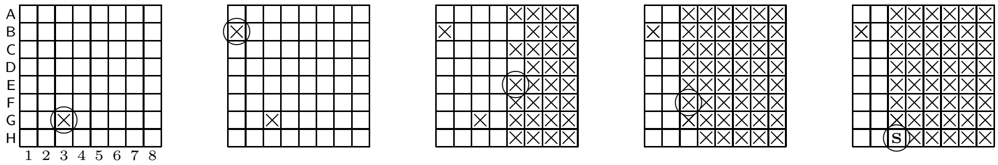
1: is $x \geq 32$?
2: is $x \bmod 32 \geq 16$?
3: is $x \bmod 16 \geq 8$?
4: is $x \bmod 8 \geq 4$?
5: is $x \bmod 4 \geq 2$?
6: is $x \bmod 2 = 1$?

[The notation $x \bmod 32$, pronounced 'x modulo 32', denotes the remainder when x is divided by 32; for example, $35 \bmod 32 = 3$ and $32 \bmod 32 = 0$.]

The answers to these questions, if translated from $\{\text{yes}, \text{no}\}$ to $\{1, 0\}$, give the binary expansion of x, for example $35 \Rightarrow$ 100011. □

What are the Shannon information contents of the outcomes in this example? If we assume that all values of x are equally likely, then the answers to the questions are independent and each has Shannon information content $\log_2(1/0.5) = 1\,\text{bit}$; the total Shannon information gained is always six bits. Furthermore, the number x that we learn from these questions is a six-bit binary number. Our questioning strategy defines a way of encoding the random variable x as a binary file.

So far, the Shannon information content makes sense: it measures the length of a binary file that encodes x. However, we have not yet studied ensembles where the outcomes have unequal probabilities. Does the Shannon information content make sense there too?



move #	1	2	32	48	49
question	G3	B1	E5	F3	H3
outcome	$x = \mathtt{n}$	$x = \mathtt{n}$	$x = \mathtt{n}$	$x = \mathtt{n}$	$x = \mathtt{y}$
$P(x)$	$\frac{63}{64}$	$\frac{62}{63}$	$\frac{32}{33}$	$\frac{16}{17}$	$\frac{1}{16}$
$h(x)$	0.0227	0.0230	0.0443	0.0874	4.0
Total info.	0.0227	0.0458	1.0	2.0	6.0

Figure 4.3. A game of `submarine`. The submarine is hit on the 49th attempt.

The game of submarine: how many bits can one bit convey?

In the game of battleships, each player hides a fleet of ships in a sea represented by a square grid. On each turn, one player attempts to hit the other's ships by firing at one square in the opponent's sea. The response to a selected square such as 'G3' is either 'miss', 'hit', or 'hit and destroyed'.

In a boring version of battleships called `submarine`, each player hides just one submarine in one square of an eight-by-eight grid. Figure 4.3 shows a few pictures of this game in progress: the circle represents the square that is being fired at, and the ×s show squares in which the outcome was a miss, $x = \mathtt{n}$; the submarine is hit (outcome $x = \mathtt{y}$ shown by the symbol $\mathtt{s}$) on the 49th attempt.

Each shot made by a player defines an ensemble. The two possible outcomes are $\{\mathtt{y}, \mathtt{n}\}$, corresponding to a hit and a miss, and their probabilities depend on the state of the board. At the beginning, $P(\mathtt{y}) = 1/64$ and $P(\mathtt{n}) = 63/64$. At the second shot, if the first shot missed, $P(\mathtt{y}) = 1/63$ and $P(\mathtt{n}) = 62/63$. At the third shot, if the first two shots missed, $P(\mathtt{y}) = 1/62$ and $P(\mathtt{n}) = 61/62$.

The Shannon information gained from an outcome x is $h(x) = \log(1/P(x))$. If we are lucky, and hit the submarine on the first shot, then

$$h(x) = h_{(1)}(\mathtt{y}) = \log_2 64 = 6\,\text{bits}. \tag{4.8}$$

Now, it might seem a little strange that one binary outcome can convey six bits. But we have learnt the hiding place, which could have been any of 64 squares; so we have, by one lucky binary question, indeed learnt six bits.

What if the first shot misses? The Shannon information that we gain from this outcome is

$$h(x) = h_{(1)}(\mathtt{n}) = \log_2 \frac{64}{63} = 0.0227\,\text{bits}. \tag{4.9}$$

Does this make sense? It is not so obvious. Let's keep going. If our second shot also misses, the Shannon information content of the second outcome is

$$h_{(2)}(\mathtt{n}) = \log_2 \frac{63}{62} = 0.0230\,\text{bits}. \tag{4.10}$$

If we miss thirty-two times (firing at a new square each time), the total Shannon information gained is

$$\begin{aligned} &\log_2 \frac{64}{63} + \log_2 \frac{63}{62} + \cdots + \log_2 \frac{33}{32} \\ &= \; 0.0227 + 0.0230 + \cdots + 0.0430 \; = \; 1.0\,\text{bits}. \end{aligned} \tag{4.11}$$

Why this round number? Well, what have we learnt? We now know that the submarine is not in any of the 32 squares we fired at; learning that fact is just like playing a game of `sixty-three` (p. 70), asking as our first question 'is x one of the thirty-two numbers corresponding to these squares I fired at?', and receiving the answer 'no'. This answer rules out half of the hypotheses, so it gives us one bit.

After 48 unsuccessful shots, the information gained is 2 bits: the unknown location has been narrowed down to one quarter of the original hypothesis space.

What if we hit the submarine on the 49th shot, when there were 16 squares left? The Shannon information content of this outcome is

$$h_{(49)}(\mathrm{y}) = \log_2 16 = 4.0\,\text{bits}. \tag{4.12}$$

The total Shannon information content of all the outcomes is

$$\begin{aligned}&\log_2 \frac{64}{63} + \log_2 \frac{63}{62} + \cdots + \log_2 \frac{17}{16} + \log_2 \frac{16}{1}\\ &= \;0.0227 + 0.0230 + \cdots + 0.0874 + 4.0 \;=\; 6.0\,\text{bits}. \end{aligned}\tag{4.13}$$

So once we know where the submarine is, the total Shannon information content gained is 6 bits.

This result holds regardless of when we hit the submarine. If we hit it when there are n squares left to choose from – n was 16 in equation (4.13) – then the total information gained is:

$$\begin{aligned}&\log_2 \frac{64}{63} + \log_2 \frac{63}{62} + \cdots + \log_2 \frac{n+1}{n} + \log_2 \frac{n}{1}\\ &= \;\log_2 \left[\frac{64}{63} \times \frac{63}{62} \times \cdots \times \frac{n+1}{n} \times \frac{n}{1}\right] \;=\; \log_2 \frac{64}{1} \;=\; 6\,\text{bits}. \end{aligned}\tag{4.14}$$

What have we learned from the examples so far? I think the `submarine` example makes quite a convincing case for the claim that the Shannon information content is a sensible measure of information content. And the game of `sixty-three` shows that the Shannon information content can be intimately connected to the size of a file that encodes the outcomes of a random experiment, thus suggesting a possible connection to data compression.

In case you're not convinced, let's look at one more example.

The Wenglish language

Wenglish is a language similar to English. Wenglish sentences consist of words drawn at random from the Wenglish dictionary, which contains $2^{15} = 32{,}768$ words, all of length 5 characters. Each word in the Wenglish dictionary was constructed at random by picking five letters from the probability distribution over `a`...`z` depicted in figure 2.1.

1	`aaail`
2	`aaaiu`
3	`aaald`
	⋮
129	`abati`
	⋮
2047	`azpan`
2048	`aztdn`
	⋮
	⋮
16 384	`odrcr`
	⋮
	⋮
32 737	`zatnt`
	⋮
32 768	`zxast`

Figure 4.4. The Wenglish dictionary.

Some entries from the dictionary are shown in alphabetical order in figure 4.4. Notice that the number of words in the dictionary (32,768) is much smaller than the total number of possible words of length 5 letters, $26^5 \simeq 12{,}000{,}000$.

Because the probability of the letter `z` is about 1/1000, only 32 of the words in the dictionary begin with the letter `z`. In contrast, the probability of the letter `a` is about 0.0625, and 2048 of the words begin with the letter `a`. Of those 2048 words, two start `az`, and 128 start `aa`.

Let's imagine that we are reading a Wenglish document, and let's discuss the Shannon information content of the characters as we acquire them. If we are given the text one word at a time, the Shannon information content of each five-character word is $\log 32{,}768 = 15$ bits, since Wenglish uses all its words with equal probability. The average information content per character is therefore 3 bits.

Now let's look at the information content if we read the document one character at a time. If, say, the first letter of a word is `a`, the Shannon information content is $\log 1/0.0625 \simeq 4$ bits. If the first letter is `z`, the Shannon information content is $\log 1/0.001 \simeq 10$ bits. The information content is thus highly variable at the first character. The total information content of the 5 characters in a word, however, is exactly 15 bits; so the letters that follow an initial `z` have lower average information content per character than the letters that follow an initial `a`. A rare initial letter such as `z` indeed conveys more information about what the word is than a common initial letter.

Similarly, in English, if rare characters occur at the start of the word (e.g. `xyl...`), then often we can identify the whole word immediately; whereas words that start with common characters (e.g. `pro...`) require more characters before we can identify them.

▶ 4.2 Data compression

The preceding examples justify the idea that the Shannon information content of an outcome is a natural measure of its information content. Improbable outcomes do convey more information than probable outcomes. We now discuss the information content of a source by considering how many bits are needed to describe the outcome of an experiment.

If we can show that we can compress data from a particular source into a file of L bits per source symbol and recover the data reliably, then we will say that the average information content of that source is at most L bits per symbol.

Example: compression of text files

A file is composed of a sequence of bytes. A byte is composed of 8 bits and can have a decimal value between 0 and 255. A typical text file is composed of the ASCII character set (decimal values 0 to 127). This character set uses only seven of the eight bits in a byte.

Here we use the word 'bit' with its meaning, 'a symbol with two values', not to be confused with the unit of information content.

▷ Exercise 4.4.[*1*, p.86] By how much could the size of a file be reduced given that it is an ASCII file? How would you achieve this reduction?

Intuitively, it seems reasonable to assert that an ASCII file contains 7/8 as much information as an arbitrary file of the same size, since we already know one out of every eight bits before we even look at the file. This is a simple example of redundancy. Most sources of data have further redundancy: English text files use the ASCII characters with non-equal frequency; certain pairs of letters are more probable than others; and entire words can be predicted given the context and a semantic understanding of the text.

Some simple data compression methods that define measures of information content

One way of measuring the information content of a random variable is simply to count the number of *possible* outcomes, $|\mathcal{A}_X|$. (The number of elements in a set $\mathcal{A}$ is denoted by $|\mathcal{A}|$.) If we gave a binary name to each outcome, the length of each name would be $\log_2 |\mathcal{A}_X|$ bits, if $|\mathcal{A}_X|$ happened to be a power of 2. We thus make the following definition.

The raw bit content of X is

$$H_0(X) = \log_2 |\mathcal{A}_X|. \tag{4.15}$$

$H_0(X)$ is a lower bound for the number of binary questions that are always guaranteed to identify an outcome from the ensemble X. It is an additive quantity: the raw bit content of an ordered pair x, y, having $|\mathcal{A}_X||\mathcal{A}_Y|$ possible outcomes, satisfies

$$H_0(X,Y) = H_0(X) + H_0(Y). \tag{4.16}$$

This measure of information content does not include any probabilistic element, and the encoding rule it corresponds to does not 'compress' the source data, it simply maps each outcome to a constant-length binary string.

Exercise 4.5.$^{[2,\ \text{p.86}]}$ Could there be a compressor that maps an outcome x to a binary code $c(x)$, and a decompressor that maps c back to x, such that *every possible outcome* is compressed into a binary code of length *shorter* than $H_0(X)$ bits?

Even though a simple counting argument shows that it is impossible to make a reversible compression program that reduces the size of *all* files, amateur compression enthusiasts frequently announce that they have invented a program that can do this – indeed that they can further compress compressed files by putting them through their compressor several times. Stranger yet, patents have been granted to these modern-day alchemists. See the `comp.compression` frequently asked questions for further reading.[1]

There are only two ways in which a 'compressor' can actually compress files:

1. A *lossy* compressor compresses some files, but maps some files to the *same* encoding. We'll assume that the user requires perfect recovery of the source file, so the occurrence of one of these confusable files leads to a failure (though in applications such as image compression, lossy compression is viewed as satisfactory). We'll denote by δ the probability that the source string is one of the confusable files, so a lossy compressor has a probability δ of failure. If δ can be made very small then a lossy compressor may be practically useful.

2. A *lossless* compressor maps all files to different encodings; if it shortens some files, it necessarily *makes others longer.* We try to design the compressor so that the probability that a file is lengthened is very small, and the probability that it is shortened is large.

In this chapter we discuss a simple lossy compressor. In subsequent chapters we discuss lossless compression methods.

▶ 4.3 Information content defined in terms of lossy compression

Whichever type of compressor we construct, we need somehow to take into account the *probabilities* of the different outcomes. Imagine comparing the information contents of two text files – one in which all 128 ASCII characters

[1] `http://sunsite.org.uk/public/usenet/news-faqs/comp.compression/`

are used with equal probability, and one in which the characters are used with their frequencies in English text. Can we define a measure of information content that distinguishes between these two files? Intuitively, the latter file contains less information per character because it is more predictable.

One simple way to use our knowledge that some symbols have a smaller probability is to imagine recoding the observations into a smaller alphabet – thus losing the ability to encode some of the more improbable symbols – and then measuring the raw bit content of the new alphabet. For example, we might take a risk when compressing English text, guessing that the most infrequent characters won't occur, and make a reduced ASCII code that omits the characters { !, @, #, %, ^, *, ~, <, >, /, \, _, {, }, [,], | }, thereby reducing the size of the alphabet by seventeen. The larger the risk we are willing to take, the smaller our final alphabet becomes.

We introduce a parameter δ that describes the risk we are taking when using this compression method: δ is the probability that there will be no name for an outcome x.

Example 4.6. Let

$$\begin{aligned} \mathcal{A}_X &= \{ \texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}, \texttt{e}, \texttt{f}, \texttt{g}, \texttt{h} \}, \\ \text{and} \quad \mathcal{P}_X &= \{ \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{3}{16}, \tfrac{1}{64}, \tfrac{1}{64}, \tfrac{1}{64}, \tfrac{1}{64} \}. \end{aligned} \tag{4.17}$$

The raw bit content of this ensemble is 3 bits, corresponding to 8 binary names. But notice that $P(x \in \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}\}) = 15/16$. So if we are willing to run a risk of $\delta = 1/16$ of not having a name for x, then we can get by with four names – half as many names as are needed if every $x \in \mathcal{A}_X$ has a name.

Table 4.5 shows binary names that could be given to the different outcomes in the cases $\delta = 0$ and $\delta = 1/16$. When $\delta = 0$ we need 3 bits to encode the outcome; when $\delta = 1/16$ we need only 2 bits.

$\delta = 0$		$\delta = 1/16$	
x	$c(x)$	x	$c(x)$
a	000	a	00
b	001	b	01
c	010	c	10
d	011	d	11
e	100	e	–
f	101	f	–
g	110	g	–
h	111	h	–

Table 4.5. Binary names for the outcomes, for two failure probabilities δ.

Let us now formalize this idea. To make a compression strategy with risk δ, we make the smallest possible subset S_δ such that the probability that x is not in S_δ is less than or equal to δ, i.e., $P(x \notin S_\delta) \leq \delta$. For each value of δ we can then define a new measure of information content – the log of the size of this smallest subset S_δ. [In ensembles in which several elements have the same probability, there may be several smallest subsets that contain different elements, but all that matters is their sizes (which are equal), so we will not dwell on this ambiguity.]

The smallest δ-sufficient subset S_δ is the smallest subset of $\mathcal{A}_X$ satisfying

$$P(x \in S_\delta) \geq 1 - \delta. \tag{4.18}$$

The subset S_δ can be constructed by ranking the elements of $\mathcal{A}_X$ in order of decreasing probability and adding successive elements starting from the most probable elements until the total probability is $\geq (1-\delta)$.

We can make a data compression code by assigning a binary name to each element of the smallest sufficient subset. This compression scheme motivates the following measure of information content:

The essential bit content of X is:

$$H_\delta(X) = \log_2 |S_\delta|. \tag{4.19}$$

Note that $H_0(X)$ is the special case of $H_\delta(X)$ with $\delta = 0$ (if $P(x) > 0$ for all $x \in \mathcal{A}_X$). [Caution: do not confuse $H_0(X)$ and $H_\delta(X)$ with the function $H_2(p)$ displayed in figure 4.1.]

Figure 4.6 shows $H_\delta(X)$ for the ensemble of example 4.6 as a function of δ.

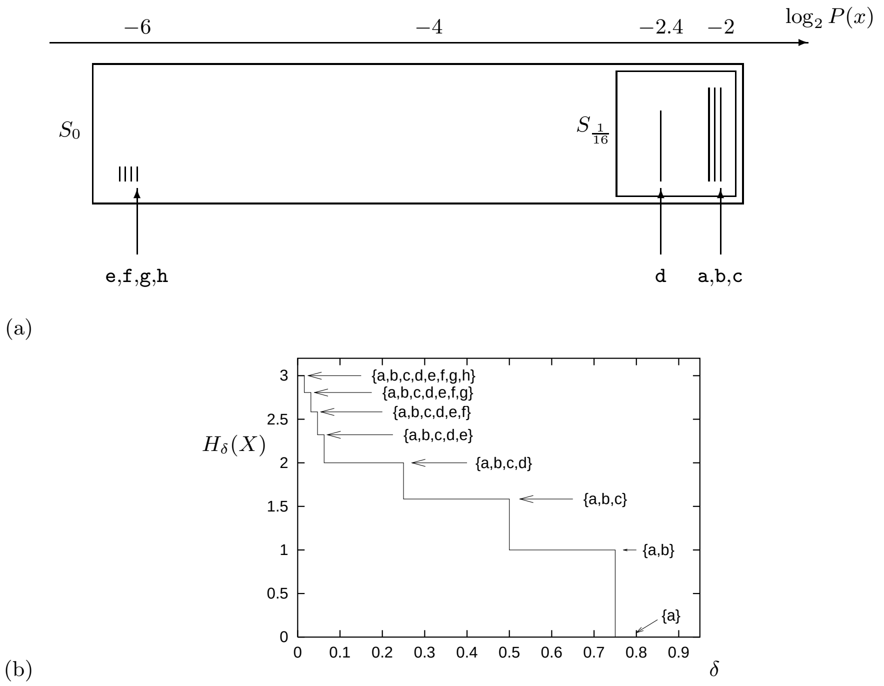


Figure 4.6. (a) The outcomes of X (from example 4.6 (p.75)), ranked by their probability. (b) The essential bit content $H_\delta(X)$. The labels on the graph show the smallest sufficient set as a function of δ. Note $H_0(X) = 3$ bits and $H_{1/16}(X) = 2$ bits.

Extended ensembles

Is this compression method any more useful if we compress *blocks* of symbols from a source?

We now turn to examples where the outcome $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ is a string of N independent identically distributed random variables from a single ensemble X. We will denote by X^N the ensemble $(X_1, X_2, \ldots, X_N)$. Remember that entropy is additive for independent variables (exercise 4.2 (p.68)), so $H(X^N) = NH(X)$.

Example 4.7. Consider a string of N flips of a bent coin, $\mathbf{x} = (x_1, x_2, \ldots, x_N)$, where $x_n \in \{\mathtt{0}, \mathtt{1}\}$, with probabilities $p_0 = 0.9$, $p_1 = 0.1$. The most probable strings $\mathbf{x}$ are those with most 0s. If $r(\mathbf{x})$ is the number of 1s in $\mathbf{x}$ then

$$P(\mathbf{x}) = p_0^{N-r(\mathbf{x})} p_1^{r(\mathbf{x})}. \tag{4.20}$$

To evaluate $H_\delta(X^N)$ we must find the smallest sufficient subset S_δ. This subset will contain all $\mathbf{x}$ with $r(\mathbf{x}) = 0, 1, 2, \ldots,$ up to some $r_{\max}(\delta) - 1$, and some of the $\mathbf{x}$ with $r(\mathbf{x}) = r_{\max}(\delta)$. Figures 4.7 and 4.8 show graphs of $H_\delta(X^N)$ against δ for the cases $N = 4$ and $N = 10$. The steps are the values of δ at which $|S_\delta|$ changes by 1, and the cusps where the slope of the staircase changes are the points where $r_{\max}$ changes by 1.

Exercise 4.8.[2, p.86] What are the mathematical shapes of the curves between the cusps?

For the examples shown in figures 4.6–4.8, $H_\delta(X^N)$ depends strongly on the value of δ, so it might not seem a fundamental or useful definition of information content. But we will consider what happens as N, the number of independent variables in X^N, increases. We will find the remarkable result that $H_\delta(X^N)$ becomes almost independent of δ – and for all δ it is very close to $NH(X)$, where $H(X)$ is the entropy of one of the random variables.

Figure 4.9 illustrates this asymptotic tendency for the binary ensemble of example 4.7. As N increases, $\frac{1}{N}H_\delta(X^N)$ becomes an increasingly flat function,

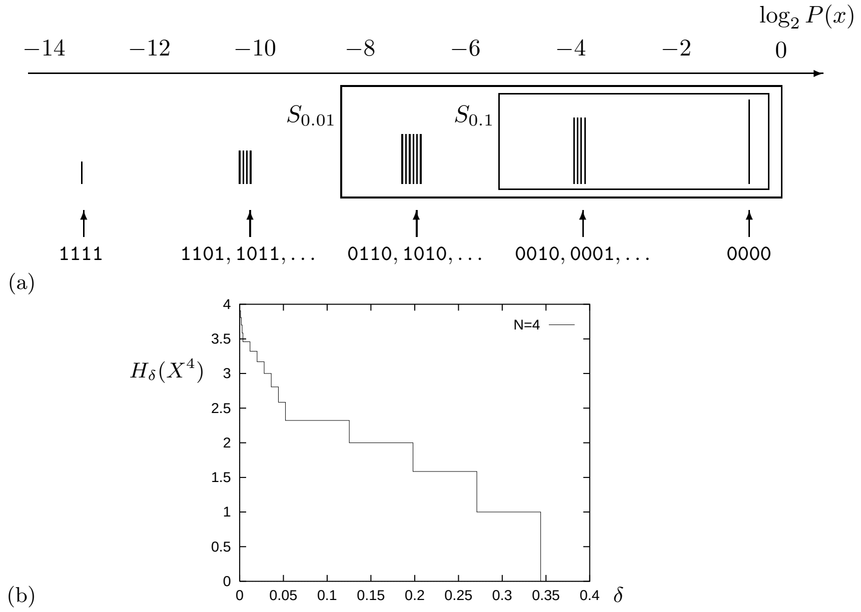


Figure 4.7. (a) The sixteen outcomes of the ensemble X^4 with $p_1 = 0.1$, ranked by probability. (b) The essential bit content $H_\delta(X^4)$. The upper schematic diagram indicates the strings' probabilities by the vertical lines' lengths (not to scale).

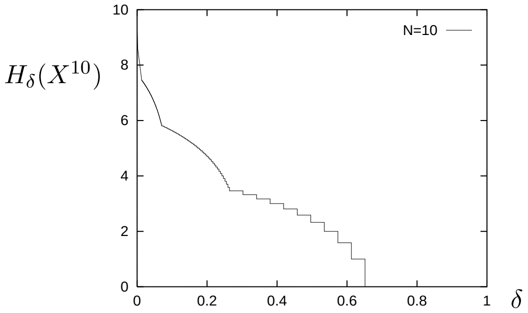


Figure 4.8. $H_\delta(X^N)$ for $N = 10$ binary variables with $p_1 = 0.1$.

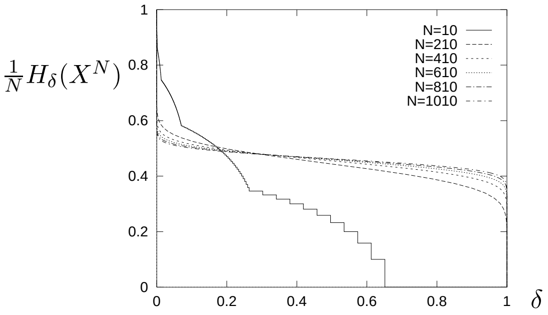


Figure 4.9. $\frac{1}{N}H_\delta(X^N)$ for $N = 10, 210, \ldots, 1010$ binary variables with $p_1 = 0.1$.

$\mathbf{x}$	$\log_2(P(\mathbf{x}))$
...1...................1.....1....1.1.......1........1...........1.....................1.......11...	-50.1
......................1.....1.....1.......1....1.........1.....................................1....	-37.3
........1....1..1...1....11..1.1.........11.........................1...1.1..1...1................1.	-65.9
1.1...1................1.......................11.1..1............................1.....1..1..11....	-56.4
...11...........1...1.....1.1......1..........1....1...1.....1............1.........................	-53.2
..............1......1.........1.1.......1..........1............1...1......................1.......	-43.7
.....1........1.......1...1............1............1...........1......1..11........................	-46.8
.....1..1..1...............111...................1...............1.........1.1...1...1.............1	-56.4
.........1..........1.....1......1..........1....1..1...	-37.3
......1........................1..............1.....1..1.1.1..1...................................1.	-43.7
1.......................1..........1...1...................1....1....1........1..11..1.1...1........	-56.4
...........11.1.........1................1......1.....................1.............................	-37.3
.1..........1...1.1.............1.......11...........1.1...1..............1.............11..........	-56.4
......1...1..1.....1..11.1.1.1...1.....................1............1.............1..1..............	-59.5
............11.1......1....1..1............................1.......1..............1.......1.........	-46.8
....	-15.2
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111	-332.1

Figure 4.10. The top 15 strings are samples from X^{100}, where $p_1 = 0.1$ and $p_0 = 0.9$. The bottom two are the most and least probable strings in this ensemble. The final column shows the log-probabilities of the random strings, which may be compared with the entropy $H(X^{100}) = 46.9$ bits.

except for tails close to $\delta = 0$ and 1. As long as we are allowed a tiny probability of error δ, compression down to NH bits is possible. Even if we are allowed a large probability of error, we still can compress only down to NH bits. This is the source coding theorem.

Theorem 4.1 Shannon's source coding theorem. *Let X be an ensemble with entropy $H(X) = H$ bits. Given $\epsilon > 0$ and $0 < \delta < 1$, there exists a positive integer N_0 such that for $N > N_0$,*

$$\left| \frac{1}{N} H_\delta(X^N) - H \right| < \epsilon. \tag{4.21}$$

▶ 4.4 Typicality

Why does increasing N help? Let's examine long strings from X^N. Table 4.10 shows fifteen samples from X^N for $N = 100$ and $p_1 = 0.1$. The probability of a string $\mathbf{x}$ that contains r 1s and $N-r$ 0s is

$$P(\mathbf{x}) = p_1^r (1 - p_1)^{N-r}. \tag{4.22}$$

The number of strings that contain r 1s is

$$n(r) = \binom{N}{r}. \tag{4.23}$$

So the number of 1s, r, has a binomial distribution:

$$P(r) = \binom{N}{r} p_1^r (1 - p_1)^{N-r}. \tag{4.24}$$

These functions are shown in figure 4.11. The mean of r is Np_1, and its standard deviation is $\sqrt{Np_1(1 - p_1)}$ (p.1). If N is 100 then

$$r \sim Np_1 \pm \sqrt{Np_1(1 - p_1)} \simeq 10 \pm 3. \tag{4.25}$$

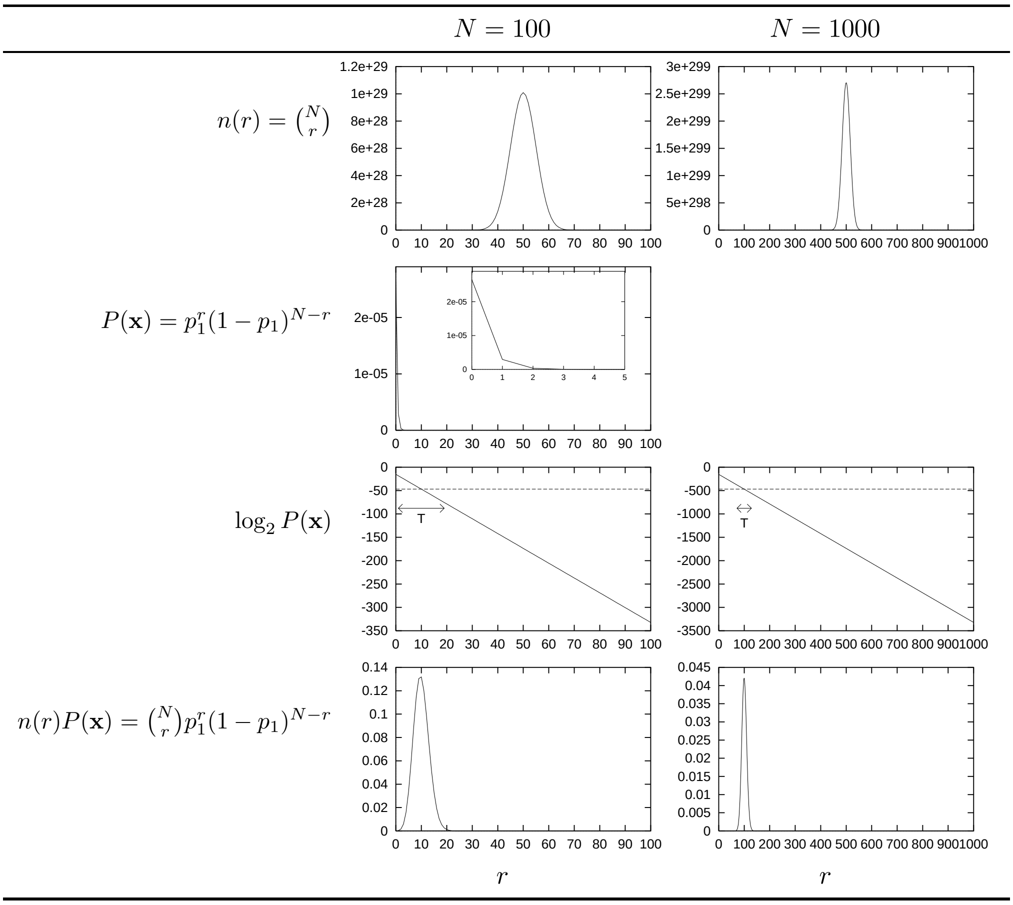


Figure 4.11. Anatomy of the typical set T. For $p_1 = 0.1$ and $N = 100$ and $N = 1000$, these graphs show $n(r)$, the number of strings containing r 1s; the probability $P(\mathbf{x})$ of a single string that contains r 1s; the same probability on a log scale; and the total probability $n(r)P(\mathbf{x})$ of all strings that contain r 1s. The number r is on the horizontal axis. The plot of $\log_2 P(\mathbf{x})$ also shows by a dotted line the mean value of $\log_2 P(\mathbf{x}) = -NH_2(p_1)$, which equals -46.9 when $N = 100$ and -469 when $N = 1000$. The typical set includes only the strings that have $\log_2 P(\mathbf{x})$ close to this value. The range marked T shows the set $T_{N\beta}$ (as defined in section 4.4) for $N = 100$ and $\beta = 0.29$ (left) and $N = 1000$, $\beta = 0.09$ (right).

If $N = 1000$ then

$$r \sim 100 \pm 10. \tag{4.26}$$

Notice that as N gets bigger, the probability distribution of r becomes more concentrated, in the sense that while the range of possible values of r grows as N, the standard deviation of r grows only as $\sqrt{N}$. That r is most likely to fall in a small range of values implies that the outcome $\mathbf{x}$ is also most likely to fall in a corresponding small subset of outcomes that we will call the *typical set*.

Definition of the typical set

Let us define typicality for an arbitrary ensemble X with alphabet $\mathcal{A}_X$. Our definition of a typical string will involve the string's probability. A long string of N symbols will usually contain about $p_1 N$ occurrences of the first symbol, $p_2 N$ occurrences of the second, etc. Hence the probability of this string is roughly

$$P(\mathbf{x})_{\text{typ}} = P(x_1)P(x_2)P(x_3)\ldots P(x_N) \simeq p_1^{(p_1 N)} p_2^{(p_2 N)} \ldots p_I^{(p_I N)} \tag{4.27}$$

so that the information content of a typical string is

$$\log_2 \frac{1}{P(\mathbf{x})} \simeq N \sum_i p_i \log_2 \frac{1}{p_i} = NH. \tag{4.28}$$

So the random variable $\log_2 1/P(\mathbf{x})$, which is the information content of $\mathbf{x}$, is very likely to be close in value to NH. We build our definition of typicality on this observation.

We define the typical elements of $\mathcal{A}_X^N$ to be those elements that have probability close to 2^{-NH}. (Note that the typical set, unlike the smallest sufficient subset, does *not* include the most probable elements of $\mathcal{A}_X^N$, but we will show that these most probable elements contribute negligible probability.)

We introduce a parameter β that defines how close the probability has to be to 2^{-NH} for an element to be 'typical'. We call the set of typical elements the typical set, $T_{N\beta}$:

$$T_{N\beta} \equiv \left\{ \mathbf{x} \in \mathcal{A}_X^N : \left| \frac{1}{N} \log_2 \frac{1}{P(\mathbf{x})} - H \right| < \beta \right\}. \tag{4.29}$$

We will show that whatever value of β we choose, the typical set contains almost all the probability as N increases.

This important result is sometimes called the *'asymptotic equipartition' principle*.

'Asymptotic equipartition' principle. For an ensemble of N independent identically distributed (i.i.d.) random variables $X^N \equiv (X_1, X_2, \ldots, X_N)$, with N sufficiently large, the outcome $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ is almost certain to belong to a subset of $\mathcal{A}_X^N$ having only $2^{NH(X)}$ members, each having probability 'close to' $2^{-NH(X)}$.

Notice that if $H(X) < H_0(X)$ then $2^{NH(X)}$ is a *tiny* fraction of the number of possible outcomes $|\mathcal{A}_X^N| = |\mathcal{A}_X|^N = 2^{NH_0(X)}$.

> The term equipartition is chosen to describe the idea that the members of the typical set have *roughly equal* probability. [This should not be taken too literally, hence my use of quotes around 'asymptotic equipartition'; see page 83.]
>
> A second meaning for equipartition, in thermal physics, is the idea that each degree of freedom of a classical system has equal average energy, $\frac{1}{2}kT$. This second meaning is not intended here.

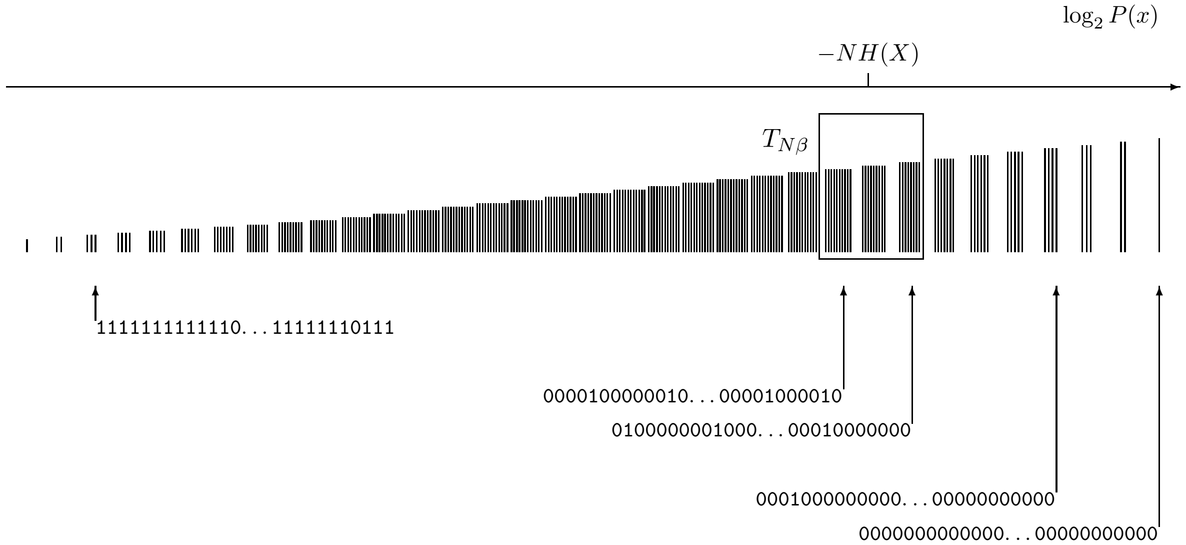


Figure 4.12. Schematic diagram showing all strings in the ensemble X^N ranked by their probability, and the typical set $T_{N\beta}$.

The 'asymptotic equipartition' principle is equivalent to:

Shannon's source coding theorem (verbal statement). N i.i.d. random variables each with entropy $H(X)$ can be compressed into more than $NH(X)$ bits with negligible risk of information loss, as $N \rightarrow \infty$; conversely if they are compressed into fewer than $NH(X)$ bits it is virtually certain that information will be lost.

These two theorems are equivalent because we can define a compression algorithm that gives a distinct name of length $NH(X)$ bits to each $\mathbf{x}$ in the typical set.

4.5 Proofs

This section may be skipped if found tough going.

The law of large numbers

Our proof of the source coding theorem uses the law of large numbers.

Mean and variance of a real random variable are $\mathcal{E}[u] = \bar{u} = \sum_u P(u)u$ and $\text{var}(u) = \sigma_u^2 = \mathcal{E}[(u-\bar{u})^2] = \sum_u P(u)(u-\bar{u})^2$.

Technical note: strictly I am assuming here that u is a function $u(x)$ of a sample x from a finite discrete ensemble X. Then the summations $\sum_u P(u)f(u)$ should be written $\sum_x P(x)f(u(x))$. This means that $P(u)$ is a finite sum of delta functions. This restriction guarantees that the mean and variance of u do exist, which is not necessarily the case for general $P(u)$.

Chebyshev's inequality 1. Let t be a non-negative real random variable, and let α be a positive real number. Then

$$P(t \geq \alpha) \leq \frac{\bar{t}}{\alpha}. \tag{4.30}$$

Proof: $P(t \geq \alpha) = \sum_{t \geq \alpha} P(t)$. We multiply each term by $t/\alpha \geq 1$ and obtain: $P(t \geq \alpha) \leq \sum_{t \geq \alpha} P(t)t/\alpha$. We add the (non-negative) missing terms and obtain: $P(t \geq \alpha) \leq \sum_t P(t)t/\alpha = \bar{t}/\alpha$. □

Chebyshev's inequality 2. Let x be a random variable, and let α be a positive real number. Then

$$P\left((x-\bar{x})^2 \geq \alpha\right) \;\leq\; \sigma_x^2/\alpha. \tag{4.31}$$

Proof: Take $t = (x-\bar{x})^2$ and apply the previous proposition. □

Weak law of large numbers. Take x to be the average of N independent random variables $h_1, \ldots, h_N$, having common mean $\bar{h}$ and common variance σ_h^2: $x = \frac{1}{N}\sum_{n=1}^{N} h_n$. Then

$$P((x-\bar{h})^2 \geq \alpha) \leq \sigma_h^2/\alpha N. \tag{4.32}$$

Proof: obtained by showing that $\bar{x} = \bar{h}$ and that $\sigma_x^2 = \sigma_h^2/N$. □

We are interested in x being very close to the mean (α very small). No matter how large σ_h^2 is, and no matter how small the required α is, and no matter how small the desired probability that $(x-\bar{h})^2 \geq \alpha$, we can always achieve it by taking N large enough.

Proof of theorem 4.1 (p. 78)

We apply the law of large numbers to the random variable $\frac{1}{N}\log_2\frac{1}{P(\mathbf{x})}$ defined for $\mathbf{x}$ drawn from the ensemble X^N. This random variable can be written as the average of N information contents $h_n = \log_2(1/P(x_n))$, each of which is a random variable with mean $H = H(X)$ and variance $\sigma^2 \equiv \mathrm{var}[\log_2(1/P(x_n))]$. (Each term h_n is the Shannon information content of the nth outcome.)

We again define the typical set with parameters N and β thus:

$$T_{N\beta} = \left\{\mathbf{x} \in \mathcal{A}_X^N : \left[\frac{1}{N}\log_2\frac{1}{P(\mathbf{x})} - H\right]^2 < \beta^2\right\}. \tag{4.33}$$

For all $\mathbf{x} \in T_{N\beta}$, the probability of $\mathbf{x}$ satisfies

$$2^{-N(H+\beta)} < P(\mathbf{x}) < 2^{-N(H-\beta)}. \tag{4.34}$$

And by the law of large numbers,

$$P(\mathbf{x} \in T_{N\beta}) \geq 1 - \frac{\sigma^2}{\beta^2 N}. \tag{4.35}$$

We have thus proved the 'asymptotic equipartition' principle. As N increases, the probability that $\mathbf{x}$ falls in $T_{N\beta}$ approaches 1, for any β. How does this result relate to source coding?

We must relate $T_{N\beta}$ to $H_\delta(X^N)$. We will show that for any given δ there is a sufficiently big N such that $H_\delta(X^N) \simeq NH$.

Part 1: $\frac{1}{N}H_\delta(X^N) < H + \epsilon$.

The set $T_{N\beta}$ is not the best subset for compression. So the size of $T_{N\beta}$ gives an upper bound on H_δ. We show how *small* $H_\delta(X^N)$ must be by calculating how big $T_{N\beta}$ could possibly be. We are free to set β to any convenient value. The smallest possible probability that a member of $T_{N\beta}$ can have is $2^{-N(H+\beta)}$, and the total probability contained by $T_{N\beta}$ can't be any bigger than 1. So

$$|T_{N\beta}|\, 2^{-N(H+\beta)} < 1, \tag{4.36}$$

that is, the size of the typical set is bounded by

$$|T_{N\beta}| < 2^{N(H+\beta)}. \tag{4.37}$$

If we set $\beta = \epsilon$ and N_0 such that $\frac{\sigma^2}{\epsilon^2 N_0} \leq \delta$, then $P(T_{N\beta}) \geq 1 - \delta$, and the set $T_{N\beta}$ becomes a witness to the fact that $H_\delta(X^N) \leq \log_2|T_{N\beta}| < N(H+\epsilon)$.

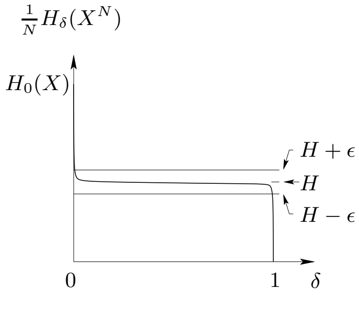
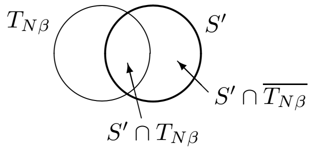


Figure 4.13. Schematic illustration of the two parts of the theorem. Given any δ and ϵ, we show that for large enough N, $\frac{1}{N}H_\delta(X^N)$ lies (1) below the line $H+\epsilon$ and (2) above the line $H-\epsilon$.

Part 2: $\frac{1}{N} H_\delta(X^N) > H - \epsilon$.

Imagine that someone claims this second part is not so – that, for any N, the smallest δ-sufficient subset S_δ is smaller than the above inequality would allow. We can make use of our typical set to show that they must be mistaken. Remember that we are free to set β to any value we choose. We will set $\beta = \epsilon/2$, so that our task is to prove that a subset S' having $|S'| \leq 2^{N(H-2\beta)}$ and achieving $P(\mathbf{x} \in S') \geq 1 - \delta$ cannot exist (for N greater than an N_0 that we will specify).

So, let us consider the probability of falling in this rival smaller subset S'. The probability of the subset S' is



$$P(\mathbf{x} \in S') \;=\; P(\mathbf{x} \in S' \cap T_{N\beta}) + P(\mathbf{x} \in S' \cap \overline{T_{N\beta}}), \tag{4.38}$$

where $\overline{T_{N\beta}}$ denotes the complement $\{\mathbf{x} \notin T_{N\beta}\}$. The maximum value of the first term is found if $S' \cap T_{N\beta}$ contains $2^{N(H-2\beta)}$ outcomes all with the maximum probability, $2^{-N(H-\beta)}$. The maximum value the second term can have is $P(\mathbf{x} \notin T_{N\beta})$. So:

$$P(\mathbf{x} \in S') \;\leq\; 2^{N(H-2\beta)}\, 2^{-N(H-\beta)} + \frac{\sigma^2}{\beta^2 N} = 2^{-N\beta} + \frac{\sigma^2}{\beta^2 N}. \tag{4.39}$$

We can now set $\beta = \epsilon/2$ and N_0 such that $P(\mathbf{x} \in S') < 1 - \delta$, which shows that S' cannot satisfy the definition of a sufficient subset S_δ. Thus *any* subset S' with size $|S'| \leq 2^{N(H-\epsilon)}$ has probability less than $1 - \delta$, so by the definition of H_δ, $H_\delta(X^N) > N(H - \epsilon)$.

Thus for large enough N, the function $\frac{1}{N} H_\delta(X^N)$ is essentially a constant function of δ, for $0 < \delta < 1$, as illustrated in figures 4.9 and 4.13. □

▶ 4.6 Comments

The source coding theorem (p.78) has two parts, $\frac{1}{N} H_\delta(X^N) < H + \epsilon$, and $\frac{1}{N} H_\delta(X^N) > H - \epsilon$. Both results are interesting.

The first part tells us that even if the probability of error δ is extremely small, the number of bits per symbol $\frac{1}{N} H_\delta(X^N)$ needed to specify a long N-symbol string $\mathbf{x}$ with vanishingly small error probability does not have to exceed $H + \epsilon$ bits. We need to have only a tiny tolerance for error, and the number of bits required drops significantly from $H_0(X)$ to $(H + \epsilon)$.

What happens if we are yet more tolerant to compression errors? Part 2 tells us that even if δ is very close to 1, so that errors are made most of the time, the average number of bits per symbol needed to specify $\mathbf{x}$ must still be at least $H - \epsilon$ bits. These two extremes tell us that regardless of our specific allowance for error, the number of bits per symbol needed to specify $\mathbf{x}$ is H bits; no more and no less.

Caveat regarding 'asymptotic equipartition'

I put the words 'asymptotic equipartition' in quotes because it is important not to think that the elements of the typical set $T_{N\beta}$ really do have roughly the same probability as each other. They are similar in probability only in the sense that their values of $\log_2 \frac{1}{P(\mathbf{x})}$ are within $2N\beta$ of each other. Now, as β is decreased, how does N have to increase, if we are to keep our bound on the mass of the typical set, $P(\mathbf{x} \in T_{N\beta}) \geq 1 - \frac{\sigma^2}{\beta^2 N}$, constant? N must grow as $1/\beta^2$, so, if we write β in terms of N as $\alpha/\sqrt{N}$, for some constant α, then the most probable string in the typical set will be of order $2^{\alpha\sqrt{N}}$ times greater than the least probable string in the typical set. As β decreases, N increases, and this ratio $2^{\alpha\sqrt{N}}$ grows exponentially. Thus we have 'equipartition' only in a weak sense!

Why did we introduce the typical set?

The best choice of subset for block compression is (by definition) S_δ, not a typical set. So why did we bother introducing the typical set? The answer is, *we can count the typical set.* We know that all its elements have 'almost identical' probability (2^{-NH}), and we know the whole set has probability almost 1, so the typical set must have roughly 2^{NH} elements. Without the help of the typical set (which is very similar to S_δ) it would have been hard to count how many elements there are in S_δ.

▶ 4.7 Exercises

Weighing problems

▷ Exercise 4.9.[*1*] While some people, when they first encounter the weighing problem with 12 balls and the three-outcome balance (exercise 4.1 (p.66)), think that weighing six balls against six balls is a good first weighing, others say 'no, weighing six against six conveys *no* information at all'. Explain to the second group why they are both right and wrong. Compute the information gained about *which is the odd ball*, and the information gained about *which is the odd ball and whether it is heavy or light.*

▷ Exercise 4.10.[*2*] Solve the weighing problem for the case where there are 39 balls of which one is known to be odd.

▷ Exercise 4.11.[*2*] You are given 16 balls, all of which are equal in weight except for one that is either heavier or lighter. You are also given a bizarre two-pan balance that can report only two outcomes: 'the two sides balance' or 'the two sides do not balance'. Design a strategy to determine which is the odd ball in as few uses of the balance as possible.

▷ Exercise 4.12.[*2*] You have a two-pan balance; your job is to weigh out bags of flour with integer weights 1 to 40 pounds inclusive. How many weights do you need? [You are allowed to put weights on either pan. You're only allowed to put one flour bag on the balance at a time.]

Exercise 4.13.[*4*, p.86] (a) Is it possible to solve exercise 4.1 (p.66) (the weighing problem with 12 balls and the three-outcome balance) using a sequence of three *fixed* weighings, such that the balls chosen for the second weighing do not depend on the outcome of the first, and the third weighing does not depend on the first or second?

(b) Find a solution to the general N-ball weighing problem in which exactly one of N balls is odd. Show that in W weighings, an odd ball can be identified from among $N = (3^W - 3)/2$ balls.

Exercise 4.14.[*3*] You are given 12 balls and the three-outcome balance of exercise 4.1; this time, *two* of the balls are odd; each odd ball may be heavy or light, and we don't know which. We want to identify the odd balls and in which direction they are odd.

(a) *Estimate* how many weighings are required by the optimal strategy. And what if there are three odd balls?

(b) How do your answers change if it is known that all the regular balls weigh 100 g, that light balls weigh 99 g, and heavy ones weigh 110 g?

Source coding with a lossy compressor, with loss δ

▷ Exercise 4.15.[*2*, p.87] Let $\mathcal{P}_X = \{0.2, 0.8\}$. Sketch $\frac{1}{N} H_\delta(X^N)$ as a function of δ for $N = 1, 2$ and 1000.

▷ Exercise 4.16.[*2*] Let $\mathcal{P}_Y = \{0.5, 0.5\}$. Sketch $\frac{1}{N} H_\delta(Y^N)$ as a function of δ for $N = 1, 2, 3$ and 100.

▷ Exercise 4.17.[*2*, p.87] (For physics students.) Discuss the relationship between the proof of the 'asymptotic equipartition' principle and the equivalence (for large systems) of the Boltzmann entropy and the Gibbs entropy.

Distributions that don't obey the law of large numbers

The law of large numbers, which we used in this chapter, shows that the mean of a set of N i.i.d. random variables has a probability distribution that becomes narrower, with width $\propto 1/\sqrt{N}$, as N increases. However, we have proved this property only for discrete random variables, that is, for real numbers taking on a *finite* set of possible values. While many random variables with continuous probability distributions also satisfy the law of large numbers, there are important distributions that do not. Some continuous distributions do not have a mean or variance.

▷ Exercise 4.18.[*3*, p.88] Sketch the Cauchy distribution

$$P(x) = \frac{1}{Z} \frac{1}{x^2 + 1}, \quad x \in (-\infty, \infty). \tag{4.40}$$

What is its normalizing constant Z? Can you evaluate its mean or variance?

Consider the sum $z = x_1 + x_2$, where x_1 and x_2 are independent random variables from a Cauchy distribution. What is $P(z)$? What is the probability distribution of the mean of x_1 and x_2, $\bar{x} = (x_1 + x_2)/2$? What is the probability distribution of the mean of N samples from this Cauchy distribution?

Other asymptotic properties

Exercise 4.19.[*3*] Chernoff bound. We derived the weak law of large numbers from Chebyshev's inequality (4.30) by letting the random variable t in the inequality $P(t \geq \alpha) \leq \bar{t}/\alpha$ be a function, $t = (x - \bar{x})^2$, of the random variable x we were interested in.

Other useful inequalities can be obtained by using other functions. The Chernoff bound, which is useful for bounding the tails of a distribution, is obtained by letting $t = \exp(sx)$.

Show that

$$P(x \geq a) \leq e^{-sa} g(s), \quad \text{for any } s > 0 \tag{4.41}$$

and

$$P(x \leq a) \leq e^{-sa} g(s), \quad \text{for any } s < 0 \tag{4.42}$$

where $g(s)$ is the moment-generating function of x,

$$g(s) = \sum_x P(x)\, e^{sx}. \tag{4.43}$$

Curious functions related to $p \log 1/p$

Exercise 4.20.[4, p.89] This exercise has no purpose at all; it's included for the enjoyment of those who like mathematical curiosities.

Sketch the function

$$f(x) = x^{x^{x^{x^{\cdot^{\cdot^{\cdot}}}}}} \tag{4.44}$$

for $x \geq 0$. Hint: Work out the inverse function to f – that is, the function $g(y)$ such that if $x = g(y)$ then $y = f(x)$ – it's closely related to $p \log 1/p$.

▶ 4.8 Solutions

Solution to exercise 4.2 (p.68). Let $P(x, y) = P(x)P(y)$. Then

$$\begin{aligned} H(X,Y) &= \sum_{xy} P(x)P(y) \log \frac{1}{P(x)P(y)} & (4.45)\\ &= \sum_{xy} P(x)P(y) \log \frac{1}{P(x)} + \sum_{xy} P(x)P(y) \log \frac{1}{P(y)} & (4.46)\\ &= \sum_{x} P(x) \log \frac{1}{P(x)} + \sum_{y} P(y) \log \frac{1}{P(y)} & (4.47)\\ &= H(X) + H(Y). & (4.48) \end{aligned}$$

Solution to exercise 4.4 (p.73). An ASCII file can be reduced in size by a factor of 7/8. This reduction could be achieved by a block code that maps 8-byte blocks into 7-byte blocks by copying the 56 information-carrying bits into 7 bytes, and ignoring the last bit of every character.

Solution to exercise 4.5 (p.74). The pigeon-hole principle states: you can't put 16 pigeons into 15 holes without using one of the holes twice.

Similarly, you can't give $\mathcal{A}_X$ outcomes unique binary names of some length l shorter than $\log_2 |\mathcal{A}_X|$ bits, because there are only 2^l such binary names, and $l < \log_2 |\mathcal{A}_X|$ implies $2^l < |\mathcal{A}_X|$, so at least two different inputs to the compressor would compress to the same output file.

Solution to exercise 4.8 (p.76). Between the cusps, all the changes in probability are equal, and the number of elements in T changes by one at each step. So H_δ varies logarithmically with $(-\delta)$.

Solution to exercise 4.13 (p.84). This solution was found by Dyson and Lyness in 1946 and presented in the following elegant form by John Conway in 1999. Be warned: the symbols A, B, and C are used to name the balls, to name the pans of the balance, to name the outcomes, and to name the possible states of the odd ball!

(a) Label the 12 balls by the sequences

```
AAB  ABA  ABB  ABC  BBC  BCA  BCB  BCC  CAA  CAB  CAC  CCA
```

and in the

```
1st              AAB ABA ABB ABC           BBC BCA BCB BCC
2nd weighings put AAB CAA CAB CAC in pan A, ABA ABB ABC BBC in pan B.
3rd              ABA BCA CAA CCA           AAB ABB BCB CAB
```

Now in a given weighing, a pan will either end up in the

- Canonical position (C) that it assumes when the pans are balanced, or
- Above that position (A), or
- Below it (B),

so the three weighings determine for each pan a sequence of three of these letters.

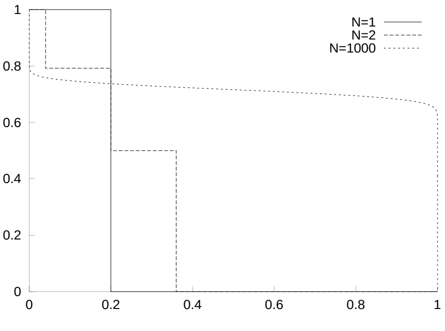
If both sequences are CCC, then there's no odd ball. Otherwise, for *just one* of the two pans, the sequence is among the 12 above, and names the odd ball, whose weight is Above or Below the proper one according as the pan is A or B.

(b) In W weighings the odd ball can be identified from among

$$N = (3^W - 3)/2 \tag{4.49}$$

balls in the same way, by labelling them with all the non-constant sequences of W letters from A, B, C whose first change is A-to-B or B-to-C or C-to-A, and at the wth weighing putting those whose wth letter is A in pan A and those whose wth letter is B in pan B.

Solution to exercise 4.15 (p.85). The curves $\frac{1}{N}H_\delta(X^N)$ as a function of δ for $N = 1, 2$ and 1000 are shown in figure 4.14. Note that $H_2(0.2) = 0.72$ bits.



$N = 1$		
δ	$\frac{1}{N}H_\delta(\mathbf{X})$	$2^{H_\delta(\mathbf{X})}$
0–0.2	1	2
0.2–1	0	1

$N = 2$		
δ	$\frac{1}{N}H_\delta(\mathbf{X})$	$2^{H_\delta(\mathbf{X})}$
0–0.04	1	4
0.04–0.2	0.79	3
0.2–0.36	0.5	2
0.36–1	0	1

Figure 4.14. $\frac{1}{N}H_\delta(\mathbf{X})$ (vertical axis) against δ (horizontal), for $N = 1, 2, 100$ binary variables with $p_1 = 0.4$.

Solution to exercise 4.17 (p.85). The Gibbs entropy is $k_\mathrm{B} \sum_i p_i \ln \frac{1}{p_i}$, where i runs over all states of the system. This entropy is equivalent (apart from the factor of k_B) to the Shannon entropy of the ensemble.

Whereas the Gibbs entropy can be defined for any ensemble, the Boltzmann entropy is only defined for *microcanonical* ensembles, which have a probability distribution that is uniform over a set of accessible states. The Boltzmann entropy is defined to be $S_\mathrm{B} = k_\mathrm{B} \ln \Omega$ where Ω is the number of accessible states of the microcanonical ensemble. This is equivalent (apart from the factor of k_B) to the perfect information content H_0 of that constrained ensemble. The Gibbs entropy of a microcanonical ensemble is trivially equal to the Boltzmann entropy.

We now consider a thermal distribution (the *canonical* ensemble), where the probability of a state $\mathbf{x}$ is

$$P(\mathbf{x}) = \frac{1}{Z} \exp\left(-\frac{E(\mathbf{x})}{k_{\rm B}T}\right). \tag{4.50}$$

With this canonical ensemble we can associate a corresponding microcanonical ensemble, an ensemble with total energy fixed to the mean energy of the canonical ensemble (fixed to within some precision ϵ). Now, fixing the total energy to a precision ϵ is equivalent to fixing the value of $\ln {}^1\!/\!P(\mathbf{x})$ to within $\epsilon k_{\rm B}T$. Our definition of the typical set $T_{N\beta}$ was precisely that it consisted of all elements that have a value of $\log P(\mathbf{x})$ very close to the mean value of $\log P(\mathbf{x})$ under the canonical ensemble, $-NH(X)$. Thus the microcanonical ensemble is equivalent to a uniform distribution over the typical set of the canonical ensemble.

Our proof of the 'asymptotic equipartition' principle thus proves – for the case of a system whose energy is separable into a sum of independent terms – that the Boltzmann entropy of the microcanonical ensemble is very close (for large N) to the Gibbs entropy of the canonical ensemble, if the energy of the microcanonical ensemble is constrained to equal the mean energy of the canonical ensemble.

Solution to exercise 4.18 (p.85). The normalizing constant of the Cauchy distribution

$$P(x) = \frac{1}{Z}\frac{1}{x^2+1}$$

is

$$Z = \int_{-\infty}^{\infty} {\rm d}x \, \frac{1}{x^2+1} = \left[\tan^{-1} x\right]_{-\infty}^{\infty} = \frac{\pi}{2} - \frac{-\pi}{2} = \pi. \tag{4.51}$$

The mean and variance of this distribution are both undefined. (The distribution is symmetrical about zero, but this does not imply that its mean is zero. The mean is the value of a divergent integral.) The sum $z = x_1 + x_2$, where x_1 and x_2 both have Cauchy distributions, has probability density given by the convolution

$$P(z) = \frac{1}{\pi^2}\int_{-\infty}^{\infty} {\rm d}x_1 \, \frac{1}{x_1^2+1}\frac{1}{(z-x_1)^2+1}, \tag{4.52}$$

which after a considerable labour using standard methods gives

$$P(z) = \frac{1}{\pi^2} 2 \frac{\pi}{z^2+4} = \frac{2}{\pi}\frac{1}{z^2+2^2}, \tag{4.53}$$

which we recognize as a Cauchy distribution with width parameter 2 (where the original distribution has width parameter 1). This implies that the mean of the two points, $\bar{x} = (x_1+x_2)/2 = z/2$, has a Cauchy distribution with width parameter 1. Generalizing, the mean of N samples from a Cauchy distribution is Cauchy-distributed with the *same parameters* as the individual samples. The probability distribution of the mean does *not* become narrower as $1/\sqrt{N}$.

The central-limit theorem does not apply to the Cauchy distribution, because it does not have a finite variance.

An alternative neat method for getting to equation (4.53) makes use of the Fourier transform of the Cauchy distribution, which is a biexponential $e^{-|\omega|}$. Convolution in real space corresponds to multiplication in Fourier space, so the Fourier transform of z is simply $e^{-|2\omega|}$. Reversing the transform, we obtain equation (4.53).

Solution to exercise 4.20 (p.86). The function $f(x)$ has inverse function

$$g(y) = y^{1/y}. \tag{4.54}$$

Note

$$\log g(y) = 1/y \log y. \tag{4.55}$$

I obtained a tentative graph of $f(x)$ by plotting $g(y)$ with y along the vertical axis and $g(y)$ along the horizontal axis. The resulting graph suggests that $f(x)$ is single valued for $x \in (0, 1)$, and looks surprisingly well-behaved and ordinary; for $x \in (1, e^{1/e})$, $f(x)$ is two-valued. $f(\sqrt{2})$ is equal both to 2 and 4. For $x > e^{1/e}$ (which is about 1.44), $f(x)$ is infinite. However, it might be argued that this approach to sketching $f(x)$ is only partly valid, if we define f as the limit of the sequence of functions x, x^x, $x^{x^x}, \ldots$; this sequence does not have a limit for $0 \leq x \leq (1/e)^e \simeq 0.07$ on account of a pitchfork bifurcation at $x = (1/e)^e$; and for $x \in (1, e^{1/e})$, the sequence's limit is single-valued – the lower of the two values sketched in the figure.

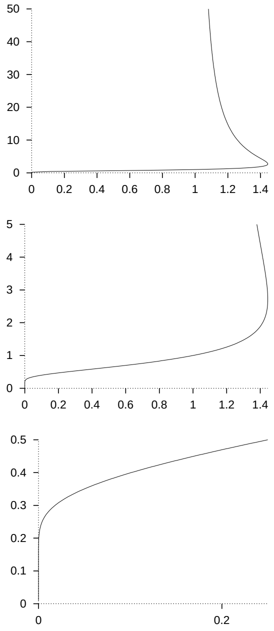


Figure 4.15. $f(x) = x^{x^{x^{x^{x^{\cdot^{\cdot^{\cdot}}}}}}}$, shown at three different scales.

About Chapter 5

In the last chapter, we saw a proof of the fundamental status of the entropy as a measure of average information content. We defined a data compression scheme using *fixed length block codes*, and proved that as N increases, it is possible to encode N i.i.d. variables $\mathbf{x} = (x_1, \ldots, x_N)$ into a block of $N(H(X)+\epsilon)$ bits with vanishing probability of error, whereas if we attempt to encode X^N into $N(H(X)-\epsilon)$ bits, the probability of error is virtually 1.

We thus verified the *possibility* of data compression, but the block coding defined in the proof did not give a practical algorithm. In this chapter and the next, we study practical data compression algorithms. Whereas the last chapter's compression scheme used large blocks of *fixed* size and was *lossy*, in the next chapter we discuss *variable-length* compression schemes that are practical for small block sizes and that are *not lossy*.

Imagine a rubber glove filled with water. If we compress two fingers of the glove, some other part of the glove has to expand, because the total volume of water is constant. (Water is essentially incompressible.) Similarly, when we shorten the codewords for some outcomes, there must be other codewords that get longer, if the scheme is not lossy. In this chapter we will discover the information-theoretic equivalent of water volume.

Before reading Chapter 5, you should have worked on exercise 2.26 (p.37).

We will use the following notation for intervals:

$$
\begin{array}{ll}
x \in [1,2) & \text{means that } x \geq 1 \text{ and } x < 2; \\
x \in (1,2] & \text{means that } x > 1 \text{ and } x \leq 2.
\end{array}
$$

5

Symbol Codes

In this chapter, we discuss *variable-length symbol codes*, which encode one source symbol at a time, instead of encoding huge strings of N source symbols. These codes are *lossless:* unlike the last chapter's block codes, they are guaranteed to compress and decompress without any errors; but there is a chance that the codes may sometimes produce encoded strings longer than the original source string.

The idea is that we can achieve compression, on average, by assigning *shorter* encodings to the more probable outcomes and *longer* encodings to the less probable.

The key issues are:

What are the implications if a symbol code is *lossless*? If some codewords are shortened, by how much do other codewords have to be lengthened?

Making compression practical. How can we ensure that a symbol code is easy to decode?

Optimal symbol codes. How should we assign codelengths to achieve the best compression, and what is the best achievable compression?

We again verify the fundamental status of the Shannon information content and the entropy, proving:

Source coding theorem (symbol codes). There exists a variable-length encoding C of an ensemble X such that the average length of an encoded symbol, $L(C, X)$, satisfies $L(C, X) \in [H(X), H(X) + 1)$.

The average length is equal to the entropy $H(X)$ only if the codelength for each outcome is equal to its Shannon information content.

We will also define a constructive procedure, the Huffman coding algorithm, that produces optimal symbol codes.

Notation for alphabets. $\mathcal{A}^N$ denotes the set of ordered N-tuples of elements from the set $\mathcal{A}$, i.e., all strings of length N. The symbol $\mathcal{A}^+$ will denote the set of all strings of finite length composed of elements from the set $\mathcal{A}$.

Example 5.1. $\{0, 1\}^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$.

Example 5.2. $\{0, 1\}^+ = \{0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$.

▶ 5.1 Symbol codes

A (binary) symbol code C for an ensemble X is a mapping from the range of x, $\mathcal{A}_X = \{a_1, \ldots, a_I\}$, to $\{\mathtt{0}, \mathtt{1}\}^+$. $c(x)$ will denote the *codeword* corresponding to x, and $l(x)$ will denote its length, with $l_i = l(a_i)$.

The *extended code* C^+ is a mapping from $\mathcal{A}_X^+$ to $\{\mathtt{0}, \mathtt{1}\}^+$ obtained by concatenation, without punctuation, of the corresponding codewords:

$$c^+(x_1 x_2 \ldots x_N) = c(x_1)c(x_2)\ldots c(x_N). \tag{5.1}$$

[The term ‘mapping’ here is a synonym for ‘function’.]

Example 5.3. A symbol code for the ensemble X defined by

$$\begin{array}{rcl} \mathcal{A}_X & = & \{\ \mathtt{a},\ \mathtt{b},\ \mathtt{c},\ \mathtt{d}\ \}, \\ \mathcal{P}_X & = & \{\ 1/2, 1/4, 1/8, 1/8\ \}, \end{array} \tag{5.2}$$

is C_0, shown in the margin.

Using the extended code, we may encode acdbac as

$$c^+(\mathtt{acdbac}) = \mathtt{100000100001010010000010}. \tag{5.3}$$

C_0:

a_i	$c(a_i)$	l_i
a	1000	4
b	0100	4
c	0010	4
d	0001	4

There are basic requirements for a useful symbol code. First, any encoded string must have a unique decoding. Second, the symbol code must be easy to decode. And third, the code should achieve as much compression as possible.

Any encoded string must have a unique decoding

A code $C(X)$ is uniquely decodeable if, under the extended code C^+, no two distinct strings have the same encoding, i.e.,

$$\forall\, \mathbf{x}, \mathbf{y} \in \mathcal{A}_X^+,\ \ \mathbf{x} \neq \mathbf{y} \ \ \Rightarrow\ \ c^+(\mathbf{x}) \neq c^+(\mathbf{y}). \tag{5.4}$$

The code C_0 defined above is an example of a uniquely decodeable code.

The symbol code must be easy to decode

A symbol code is easiest to decode if it is possible to identify the end of a codeword as soon as it arrives, which means that no codeword can be a *prefix* of another codeword. [A word c is a *prefix* of another word d if there exists a tail string t such that the concatenation ct is identical to d. For example, 1 is a prefix of 101, and so is 10.]

We will show later that we don’t lose any performance if we constrain our symbol code to be a prefix code.

A symbol code is called a prefix code if no codeword is a prefix of any other codeword.

A prefix code is also known as an *instantaneous* or *self-punctuating* code, because an encoded string can be decoded from left to right without looking ahead to subsequent codewords. The end of a codeword is immediately recognizable. A prefix code is uniquely decodeable.

Prefix codes are also known as ‘prefix-free codes’ or ‘prefix condition codes’.

Prefix codes correspond to trees, as illustrated in the margin of the next page.

Example 5.4. The code $C_1 = \{\texttt{0}, \texttt{101}\}$ is a prefix code because `0` is not a prefix of `101`, nor is `101` a prefix of `0`.

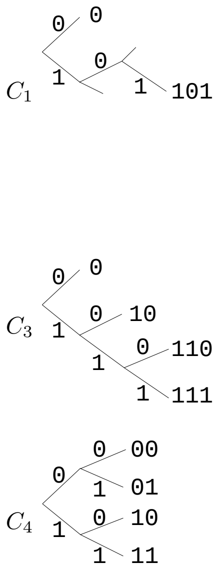
Example 5.5. Let $C_2 = \{\texttt{1}, \texttt{101}\}$. This code is not a prefix code because `1` is a prefix of `101`.

Example 5.6. The code $C_3 = \{\texttt{0}, \texttt{10}, \texttt{110}, \texttt{111}\}$ is a prefix code.

Example 5.7. The code $C_4 = \{\texttt{00}, \texttt{01}, \texttt{10}, \texttt{11}\}$ is a prefix code.

Exercise 5.8.$^{[1, \text{ p.104}]}$ Is C_2 uniquely decodeable?

Example 5.9. Consider exercise 4.1 (p.66) and figure 4.2 (p.69). Any weighing strategy that identifies the odd ball and whether it is heavy or light can be viewed as assigning a *ternary* code to each of the 24 possible states. This code is a prefix code.



Prefix codes can be represented on binary trees. *Complete* prefix codes correspond to binary trees with no unused branches. C_1 is an incomplete code.

The code should achieve as much compression as possible

The expected length $L(C, X)$ of a symbol code C for ensemble X is

$$L(C, X) = \sum_{x \in \mathcal{A}_X} P(x)\, l(x). \tag{5.5}$$

We may also write this quantity as

$$L(C, X) = \sum_{i=1}^{I} p_i l_i \tag{5.6}$$

where $I = |\mathcal{A}_X|$.

Example 5.10. Let

$$\begin{array}{rrcl} & \mathcal{A}_X & = & \{\ \texttt{a},\ \texttt{b},\ \texttt{c},\ \texttt{d}\ \}, \\ \text{and} & \mathcal{P}_X & = & \{\ 1/2, 1/4, 1/8, 1/8\ \}, \end{array} \tag{5.7}$$

and consider the code C_3. The entropy of X is 1.75 bits, and the expected length $L(C_3, X)$ of this code is also 1.75 bits. The sequence of symbols $\mathbf{x} = (\texttt{acdbac})$ is encoded as $c^+(\mathbf{x}) = \texttt{0110111100110}$. C_3 is a prefix code and is therefore uniquely decodeable. Notice that the codeword lengths satisfy $l_i = \log_2(1/p_i)$, or equivalently, $p_i = 2^{-l_i}$.

C_3:

a_i	$c(a_i)$	p_i	$h(p_i)$	l_i
a	0	$1/2$	1.0	1
b	10	$1/4$	2.0	2
c	110	$1/8$	3.0	3
d	111	$1/8$	3.0	3

Example 5.11. Consider the fixed length code for the same ensemble X, C_4. The expected length $L(C_4, X)$ is 2 bits.

Example 5.12. Consider C_5. The expected length $L(C_5, X)$ is 1.25 bits, which is less than $H(X)$. But the code is not uniquely decodeable. The sequence $\mathbf{x} = (\texttt{acdbac})$ encodes as `000111000`, which can also be decoded as (`cabdca`).

	C_4	C_5
a	00	0
b	01	1
c	10	00
d	11	11

Example 5.13. Consider the code C_6. The expected length $L(C_6, X)$ of this code is 1.75 bits. The sequence of symbols $\mathbf{x} = (\texttt{acdbac})$ is encoded as $c^+(\mathbf{x}) = \texttt{0011111010011}$.

Is C_6 a prefix code? It is not, because $c(\texttt{a}) = \texttt{0}$ is a prefix of both $c(\texttt{b})$ and $c(\texttt{c})$.

C_6:

a_i	$c(a_i)$	p_i	$h(p_i)$	l_i
a	0	$1/2$	1.0	1
b	01	$1/4$	2.0	2
c	011	$1/8$	3.0	3
d	111	$1/8$	3.0	3

Is C_6 uniquely decodeable? This is not so obvious. If you think that it might *not* be uniquely decodeable, try to prove it so by finding a pair of strings $\mathbf{x}$ and $\mathbf{y}$ that have the same encoding. [The definition of unique decodeability is given in equation (5.4).]

C_6 certainly isn't *easy* to decode. When we receive '`00`', it is possible that $\mathbf{x}$ could start '`aa`', '`ab`' or '`ac`'. Once we have received '`001111`', the second symbol is still ambiguous, as $\mathbf{x}$ could be '`abd`...' or '`acd`...'. But eventually a unique decoding crystallizes, once the next `0` appears in the encoded stream.

C_6 *is* in fact uniquely decodeable. Comparing with the prefix code C_3, we see that the codewords of C_6 are the reverse of C_3's. That C_3 is uniquely decodeable proves that C_6 is too, since any string from C_6 is identical to a string from C_3 read backwards.

▶ 5.2 What limit is imposed by unique decodeability?

We now ask, given a list of positive integers $\{l_i\}$, does there exist a uniquely decodeable code with those integers as its codeword lengths? At this stage, we ignore the probabilities of the different symbols; once we understand unique decodeability better, we'll reintroduce the probabilities and discuss how to make an *optimal* uniquely decodeable symbol code.

In the examples above, we have observed that if we take a code such as $\{\texttt{00}, \texttt{01}, \texttt{10}, \texttt{11}\}$, and shorten one of its codewords, for example $\texttt{00} \rightarrow \texttt{0}$, then we can retain unique decodeability only if we lengthen other codewords. Thus there seems to be a constrained budget that we can spend on codewords, with shorter codewords being more expensive.

Let us explore the nature of this budget. If we build a code purely from codewords of length l equal to three, how many codewords can we have and retain unique decodeability? The answer is $2^l = 8$. Once we have chosen all eight of these codewords, is there any way we could add to the code another codeword of some *other* length and retain unique decodeability? It would seem not.

What if we make a code that includes a length-one codeword, '`0`', with the other codewords being of length three? How many length-three codewords can we have? If we restrict attention to prefix codes, then we can have only four codewords of length three, namely $\{\texttt{100}, \texttt{101}, \texttt{110}, \texttt{111}\}$. What about other codes? Is there any other way of choosing codewords of length 3 that can give more codewords? Intuitively, we think this unlikely. A codeword of length 3 appears to have a cost that is 2^2 times smaller than a codeword of length 1.

Let's define a total budget of size 1, which we can spend on codewords. If we set the cost of a codeword whose length is l to 2^{-l}, then we have a pricing system that fits the examples discussed above. Codewords of length 3 cost $^1/_8$ each; codewords of length 1 cost $1/2$ each. We can spend our budget on any codewords. If we go over our budget then the code will certainly not be uniquely decodeable. If, on the other hand,

$$\sum_i 2^{-l_i} \leq 1, \tag{5.8}$$

then the code may be uniquely decodeable. This inequality is the Kraft inequality.

Kraft inequality. For any uniquely decodeable code $C(X)$ over the binary alphabet $\{0, 1\}$, the codeword lengths must satisfy:

$$\sum_{i=1}^{I} 2^{-l_i} \leq 1, \tag{5.9}$$

where $I = |\mathcal{A}_X|$.

Completeness. If a uniquely decodeable code satisfies the Kraft inequality with equality then it is called a *complete* code.

We want codes that are uniquely decodeable; prefix codes are uniquely decodeable, and are easy to decode. So life would be simpler for us if we could restrict attention to prefix codes. Fortunately, for any source there *is* an optimal symbol code that is also a prefix code.

Kraft inequality and prefix codes. Given a set of codeword lengths that satisfy the Kraft inequality, there exists a uniquely decodeable prefix code with these codeword lengths.

The Kraft inequality might be more accurately referred to as the Kraft–McMillan inequality: Kraft proved that if the inequality is satisfied, then a prefix code exists with the given lengths. McMillan (1956) proved the converse, that unique decodeability implies that the inequality holds.

Proof of the Kraft inequality. Define $S = \sum_i 2^{-l_i}$. Consider the quantity

$$S^N = \left[\sum_i 2^{-l_i}\right]^N = \sum_{i_1=1}^{I}\sum_{i_2=1}^{I}\cdots\sum_{i_N=1}^{I} 2^{-\left(l_{i_1} + l_{i_2} + \cdots l_{i_N}\right)}. \tag{5.10}$$

The quantity in the exponent, $(l_{i_1} + l_{i_2} + \cdots + l_{i_N})$, is the length of the encoding of the string $\mathbf{x} = a_{i_1}a_{i_2}\ldots a_{i_N}$. For every string $\mathbf{x}$ of length N, there is one term in the above sum. Introduce an array A_l that counts how many strings $\mathbf{x}$ have encoded length l. Then, defining $l_{\min} = \min_i l_i$ and $l_{\max} = \max_i l_i$:

$$S^N = \sum_{l=Nl_{\min}}^{Nl_{\max}} 2^{-l} A_l. \tag{5.11}$$

Now assume C is uniquely decodeable, so that for all $\mathbf{x} \neq \mathbf{y}$, $c^+(\mathbf{x}) \neq c^+(\mathbf{y})$. Concentrate on the $\mathbf{x}$ that have encoded length l. There are a total of 2^l distinct bit strings of length l, so it must be the case that $A_l \leq 2^l$. So

$$S^N = \sum_{l=Nl_{\min}}^{Nl_{\max}} 2^{-l} A_l \leq \sum_{l=Nl_{\min}}^{Nl_{\max}} 1 \;\leq\; Nl_{\max}. \tag{5.12}$$

Thus $S^N \leq l_{\max}N$ for all N. Now if S were greater than 1, then as N increases, S^N would be an exponentially growing function, and for large enough N, an exponential always exceeds a polynomial such as $l_{\max}N$. But our result ($S^N \leq l_{\max}N$) is true for *any* N. Therefore $S \leq 1$. □

▷ Exercise 5.14.[*3*, p.104] Prove the result stated above, that for any set of codeword lengths $\{l_i\}$ satisfying the Kraft inequality, there is a prefix code having those lengths.

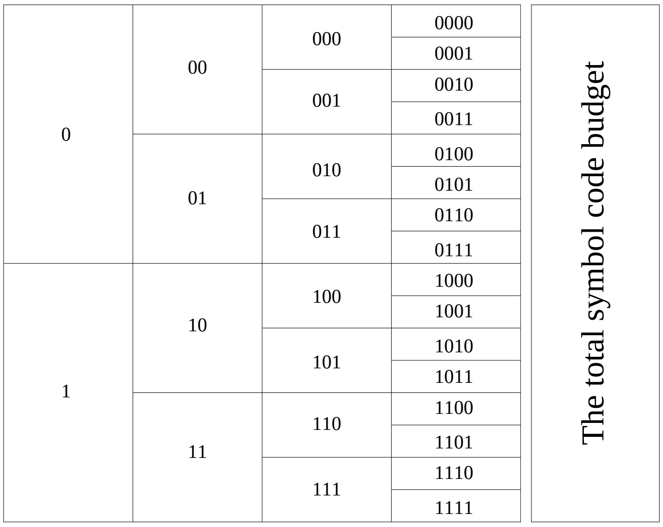


Figure 5.1. The symbol coding budget. The 'cost' 2^{-l} of each codeword (with length l) is indicated by the size of the box it is written in. The total budget available when making a uniquely decodeable code is 1.
You can think of this diagram as showing a *codeword supermarket*, with the codewords arranged in aisles by their length, and the cost of each codeword indicated by the size of its box on the shelf. If the cost of the codewords that you take exceeds the budget then your code will not be uniquely decodeable.

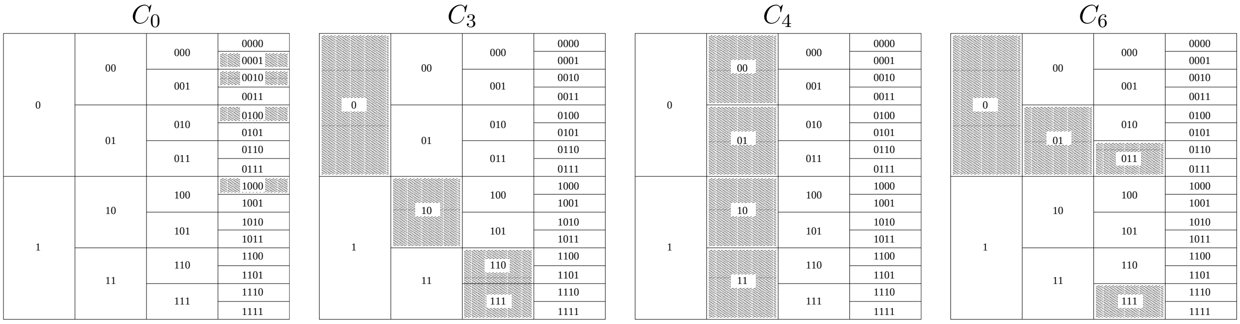


Figure 5.2. Selections of codewords made by codes C_0, C_3, C_4 and C_6 from section 5.1.

A pictorial view of the Kraft inequality may help you solve this exercise. Imagine that we are choosing the codewords to make a symbol code. We can draw the set of all candidate codewords in a supermarket that displays the 'cost' of the codeword by the area of a box (figure 5.1). The total budget available – the '1' on the right-hand side of the Kraft inequality – is shown at one side. Some of the codes discussed in section 5.1 are illustrated in figure 5.2. Notice that the codes that are prefix codes, C_0, C_3, and C_4, have the property that to the right of any selected codeword, there are no other selected codewords – because prefix codes correspond to trees. Notice that a *complete* prefix code corresponds to a *complete* tree having no unused branches.

We are now ready to put back the symbols' probabilities $\{p_i\}$. Given a set of symbol probabilities (the English language probabilities of figure 2.1, for example), how do we make the best symbol code – one with the smallest possible expected length $L(C,X)$? And what is that smallest possible expected length? It's not obvious how to assign the codeword lengths. If we give short codewords to the more probable symbols then the expected length might be reduced; on the other hand, shortening some codewords necessarily causes others to lengthen, by the Kraft inequality.

▶ 5.3 What's the most compression that we can hope for?

We wish to minimize the expected length of a code,

$$L(C,X) \;=\; \sum_i p_i l_i. \tag{5.13}$$

As you might have guessed, the entropy appears as the lower bound on the expected length of a code.

Lower bound on expected length. The expected length $L(C,X)$ of a uniquely decodeable code is bounded below by $H(X)$.

Proof. We define the *implicit probabilities* $q_i \equiv 2^{-l_i}/z$, where $z = \sum_{i'} 2^{-l_{i'}}$, so that $l_i = \log 1/q_i - \log z$. We then use Gibbs' inequality, $\sum_i p_i \log 1/q_i \geq \sum_i p_i \log 1/p_i$, with equality if $q_i = p_i$, and the Kraft inequality $z \leq 1$:

$$\begin{aligned} L(C,X) &= \sum_i p_i l_i = \sum_i p_i \log 1/q_i - \log z && (5.14)\\ &\geq \sum_i p_i \log 1/p_i - \log z && (5.15)\\ &\geq H(X). && (5.16)\end{aligned}$$

The equality $L(C,X) = H(X)$ is achieved only if the Kraft equality $z = 1$ is satisfied, and if the codelengths satisfy $l_i = \log(1/p_i)$. □

This is an important result so let's say it again:

Optimal source codelengths. The expected length is minimized and is equal to $H(X)$ only if the codelengths are equal to the *Shannon information contents*:

$$l_i = \log_2(1/p_i). \tag{5.17}$$

Implicit probabilities defined by codelengths. Conversely, any choice of codelengths $\{l_i\}$ *implicitly* defines a probability distribution $\{q_i\}$,

$$q_i \equiv 2^{-l_i}/z, \tag{5.18}$$

for which those codelengths would be the optimal codelengths. If the code is complete then $z = 1$ and the implicit probabilities are given by $q_i = 2^{-l_i}$.

▶ 5.4 How much can we compress?

So, we can't compress below the entropy. How close can we expect to get to the entropy?

Theorem 5.1 Source coding theorem for symbol codes. *For an ensemble X there exists a prefix code C with expected length satisfying*

$$H(X) \leq L(C, X) < H(X) + 1. \tag{5.19}$$

Proof. We set the codelengths to integers slightly larger than the optimum lengths:

$$l_i = \lceil \log_2(1/p_i) \rceil \tag{5.20}$$

where $\lceil l^* \rceil$ denotes the smallest integer greater than or equal to l^*. [We are not asserting that the *optimal* code necessarily uses these lengths, we are simply choosing these lengths because we can use them to prove the theorem.]

We check that there *is* a prefix code with these lengths by confirming that the Kraft inequality is satisfied.

$$\sum_i 2^{-l_i} = \sum_i 2^{-\lceil \log_2(1/p_i) \rceil} \leq \sum_i 2^{-\log_2(1/p_i)} = \sum_i p_i = 1. \tag{5.21}$$

Then we confirm

$$L(C, X) = \sum_i p_i \lceil \log(1/p_i) \rceil < \sum_i p_i (\log(1/p_i) + 1) = H(X) + 1. \tag{5.22}$$

□

The cost of using the wrong codelengths

If we use a code whose lengths are not equal to the optimal codelengths, the average message length will be larger than the entropy.

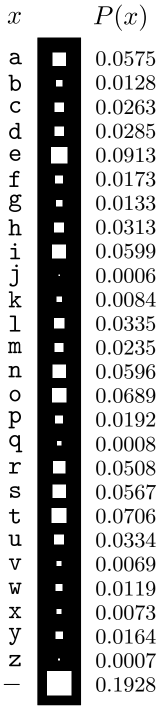
If the true probabilities are $\{p_i\}$ and we use a complete code with lengths l_i, we can view those lengths as defining implicit probabilities $q_i = 2^{-l_i}$. Continuing from equation (5.14), the average length is

$$L(C, X) = H(X) + \sum_i p_i \log p_i / q_i, \tag{5.23}$$

i.e., it exceeds the entropy by the relative entropy $D_{\mathrm{KL}}(\mathbf{p}||\mathbf{q})$ (as defined on p.34).

▶ 5.5 Optimal source coding with symbol codes: Huffman coding

Given a set of probabilities $\mathcal{P}$, how can we design an optimal prefix code? For example, what is the best symbol code for the English language ensemble shown in figure 5.3? When we say 'optimal', let's assume our aim is to minimize the expected length $L(C, X)$.



x	$P(x)$
a	0.0575
b	0.0128
c	0.0263
d	0.0285
e	0.0913
f	0.0173
g	0.0133
h	0.0313
i	0.0599
j	0.0006
k	0.0084
l	0.0335
m	0.0235
n	0.0596
o	0.0689
p	0.0192
q	0.0008
r	0.0508
s	0.0567
t	0.0706
u	0.0334
v	0.0069
w	0.0119
x	0.0073
y	0.0164
z	0.0007
–	0.1928

Figure 5.3. An ensemble in need of a symbol code.

How not to do it

One might try to roughly split the set $\mathcal{A}_X$ in two, and continue bisecting the subsets so as to define a binary tree from the root. This construction has the right spirit, as in the weighing problem, but it is not necessarily optimal; it achieves $L(C, X) \leq H(X) + 2$.

The Huffman coding algorithm

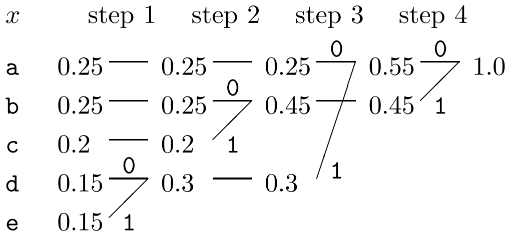
We now present a beautifully simple algorithm for finding an optimal prefix code. The trick is to construct the code *backwards* starting from the tails of the codewords; *we build the binary tree from its leaves.*

1. Take the two least probable symbols in the alphabet. These two symbols will be given the longest codewords, which will have equal length, and differ only in the last digit.
2. Combine these two symbols into a single symbol, and repeat.

Algorithm 5.4. Huffman coding algorithm.

Since each step reduces the size of the alphabet by one, this algorithm will have assigned strings to all the symbols after $|\mathcal{A}_X| - 1$ steps.

Example 5.15. Let $\mathcal{A}_X = \{$ `a`, `b`, `c`, `d`, `e` $\}$
and $\mathcal{P}_X = \{0.25, 0.25, 0.2, 0.15, 0.15\}$.



a_i	p_i	$h(p_i)$	l_i	$c(a_i)$
`a`	0.25	2.0	2	`00`
`b`	0.25	2.0	2	`10`
`c`	0.2	2.3	2	`11`
`d`	0.15	2.7	3	`010`
`e`	0.15	2.7	3	`011`

Table 5.5. Code created by the Huffman algorithm.

The codewords are then obtained by concatenating the binary digits in reverse order: $C = \{$`00`, `10`, `11`, `010`, `011`$\}$. The codelengths selected by the Huffman algorithm (column 4 of table 5.5) are in some cases longer and in some cases shorter than the ideal codelengths, the Shannon information contents $\log_2 {}^1\!/\!_{p_i}$ (column 3). The expected length of the code is $L = 2.30$ bits, whereas the entropy is $H = 2.2855$ bits. □

If at any point there is more than one way of selecting the two least probable symbols then the choice may be made in any manner – the expected length of the code will not depend on the choice.

Exercise 5.16.[*3*, p.105] Prove that there is no better symbol code for a source than the Huffman code.

Example 5.17. We can make a Huffman code for the probability distribution over the alphabet introduced in figure 2.1. The result is shown in figure 5.6. This code has an expected length of 4.15 bits; the entropy of the ensemble is 4.11 bits. Observe the disparities between the assigned codelengths and the ideal codelengths $\log_2 {}^1\!/\!_{p_i}$.

Constructing a binary tree top-down is suboptimal

In previous chapters we studied weighing problems in which we built ternary or binary trees. We noticed that balanced trees – ones in which, at every step, the two possible outcomes were as close as possible to equiprobable – appeared to describe the most efficient experiments. This gave an intuitive motivation for entropy as a measure of information content.

a_i	p_i	$\log_2 \frac{1}{p_i}$	l_i	$c(a_i)$
a	0.0575	4.1	4	0000
b	0.0128	6.3	6	001000
c	0.0263	5.2	5	00101
d	0.0285	5.1	5	10000
e	0.0913	3.5	4	1100
f	0.0173	5.9	6	111000
g	0.0133	6.2	6	001001
h	0.0313	5.0	5	10001
i	0.0599	4.1	4	1001
j	0.0006	10.7	10	1101000000
k	0.0084	6.9	7	1010000
l	0.0335	4.9	5	11101
m	0.0235	5.4	6	110101
n	0.0596	4.1	4	0001
o	0.0689	3.9	4	1011
p	0.0192	5.7	6	111001
q	0.0008	10.3	9	110100001
r	0.0508	4.3	5	11011
s	0.0567	4.1	4	0011
t	0.0706	3.8	4	1111
u	0.0334	4.9	5	10101
v	0.0069	7.2	8	11010001
w	0.0119	6.4	7	1101001
x	0.0073	7.1	7	1010001
y	0.0164	5.9	6	101001
z	0.0007	10.4	10	1101000001
–	0.1928	2.4	2	01

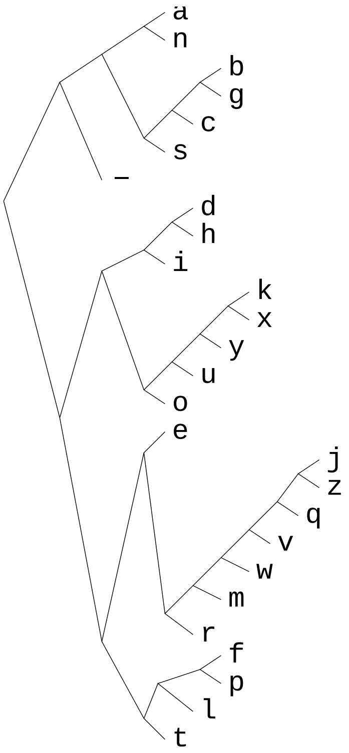


Figure 5.6. Huffman code for the English language ensemble (monogram statistics).

It is not the case, however, that optimal codes can *always* be constructed by a greedy top-down method in which the alphabet is successively divided into subsets that are as near as possible to equiprobable.

Example 5.18. Find the optimal binary symbol code for the ensemble:

$$\begin{aligned} \mathcal{A}_X &= \{\ \mathtt{a},\ \ \mathtt{b},\ \ \mathtt{c},\ \ \mathtt{d},\ \ \mathtt{e},\ \ \mathtt{f},\ \ \mathtt{g}\ \} \\ \mathcal{P}_X &= \{\, 0.01, 0.24, 0.05, 0.20, 0.47, 0.01,\ 0.02\, \} \end{aligned} . \tag{5.24}$$

Notice that a greedy top-down method can split this set into two subsets $\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d}\}$ and $\{\mathtt{e},\mathtt{f},\mathtt{g}\}$ which both have probability 1/2, and that $\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d}\}$ can be divided into subsets $\{\mathtt{a},\mathtt{b}\}$ and $\{\mathtt{c},\mathtt{d}\}$, which have probability 1/4; so a greedy top-down method gives the code shown in the third column of table 5.7, which has expected length 2.53. The Huffman coding algorithm yields the code shown in the fourth column, which has expected length 1.97. □

a_i	p_i	Greedy	Huffman
a	.01	000	000000
b	.24	001	01
c	.05	010	0001
d	.20	011	001
e	.47	10	1
f	.01	110	000001
g	.02	111	00001

Table 5.7. A greedily-constructed code compared with the Huffman code.

▶ 5.6 Disadvantages of the Huffman code

The Huffman algorithm produces an optimal symbol code for an ensemble, but this is not the end of the story. Both the word ‘ensemble’ and the phrase ‘symbol code’ need careful attention.

Changing ensemble

If we wish to communicate a sequence of outcomes from one unchanging ensemble, then a Huffman code may be convenient. But often the appropriate ensemble changes. If for example we are compressing text, then the symbol frequencies will vary with context: in English the letter `u` is much more probable after a `q` than after an `e` (figure 2.3). And furthermore, our knowledge of these context-dependent symbol frequencies will also change as we learn the statistical properties of the text source.

Huffman codes do not handle changing ensemble probabilities with any elegance. One brute-force approach would be to recompute the Huffman code every time the probability over symbols changes. Another attitude is to deny the option of adaptation, and instead run through the entire file in advance and compute a good probability distribution, which will then remain fixed throughout transmission. The code itself must also be communicated in this scenario. Such a technique is not only cumbersome and restrictive, it is also suboptimal, since the initial message specifying the code and the document itself are partially redundant. This technique therefore wastes bits.

The extra bit

An equally serious problem with Huffman codes is the innocuous-looking 'extra bit' relative to the ideal average length of $H(X)$ – a Huffman code achieves a length that satisfies $H(X) \leq L(C, X) < H(X)+1$, as proved in theorem 5.1. A Huffman code thus incurs an overhead of between 0 and 1 bits per symbol. If $H(X)$ were large, then this overhead would be an unimportant fractional increase. But for many applications, the entropy may be as low as one bit per symbol, or even smaller, so the overhead $L(C, X) - H(X)$ may dominate the encoded file length. Consider English text: in some contexts, long strings of characters may be highly predictable. For example, in the context '`strings_of_ch`', one might predict the next nine symbols to be '`aracters_`' with a probability of 0.99 each. A traditional Huffman code would be obliged to use at least one bit per character, making a total cost of nine bits where virtually no information is being conveyed (0.13 bits in total, to be precise). The entropy of English, given a good model, is about one bit per character (Shannon, 1948), so a Huffman code is likely to be highly inefficient.

A traditional patch-up of Huffman codes uses them to compress *blocks* of symbols, for example the 'extended sources' X^N we discussed in Chapter 4. The overhead per block is at most 1 bit so the overhead per symbol is at most $1/N$ bits. For sufficiently large blocks, the problem of the extra bit may be removed – but only at the expenses of (a) losing the elegant instantaneous decodeability of simple Huffman coding; and (b) having to compute the probabilities of all relevant strings and build the associated Huffman tree. One will end up explicitly computing the probabilities and codes for a huge number of strings, most of which will never actually occur. (See exercise 5.29 (p.103).)

Beyond symbol codes

Huffman codes, therefore, although widely trumpeted as 'optimal', have many defects for practical purposes. They *are* optimal *symbol* codes, but for practical purposes *we don't want a symbol code.*

The defects of Huffman codes are rectified by *arithmetic coding*, which dispenses with the restriction that each symbol must translate into an integer number of bits. Arithmetic coding is the main topic of the next chapter.

▶ 5.7 Summary

Kraft inequality. If a code is *uniquely decodeable* its lengths must satisfy

$$\sum_i 2^{-l_i} \leq 1. \tag{5.25}$$

For any lengths satisfying the Kraft inequality, there exists a prefix code with those lengths.

Optimal source codelengths for an ensemble are equal to the Shannon information contents

$$l_i = \log_2 \frac{1}{p_i}, \tag{5.26}$$

and conversely, any choice of codelengths defines *implicit probabilities*

$$q_i = \frac{2^{-l_i}}{z}. \tag{5.27}$$

The relative entropy $D_{\mathrm{KL}}(\mathbf{p}||\mathbf{q})$ measures how many bits per symbol are wasted by using a code whose implicit probabilities are $\mathbf{q}$, when the ensemble's true probability distribution is $\mathbf{p}$.

Source coding theorem for symbol codes. For an ensemble X, there exists a prefix code whose expected length satisfies

$$H(X) \leq L(C, X) < H(X) + 1. \tag{5.28}$$

The Huffman coding algorithm generates an optimal symbol code iteratively. At each iteration, the two least probable symbols are combined.

▶ 5.8 Exercises

▷ Exercise 5.19.[2] Is the code $\{\texttt{00}, \texttt{11}, \texttt{0101}, \texttt{111}, \texttt{1010}, \texttt{100100}, \texttt{0110}\}$ uniquely decodeable?

▷ Exercise 5.20.[2] Is the ternary code $\{\texttt{00}, \texttt{012}, \texttt{0110}, \texttt{0112}, \texttt{100}, \texttt{201}, \texttt{212}, \texttt{22}\}$ uniquely decodeable?

Exercise 5.21.[3, p.106] Make Huffman codes for X^2, X^3 and X^4 where $\mathcal{A}_X = \{0, 1\}$ and $\mathcal{P}_X = \{0.9, 0.1\}$. Compute their expected lengths and compare them with the entropies $H(X^2)$, $H(X^3)$ and $H(X^4)$.

Repeat this exercise for X^2 and X^4 where $\mathcal{P}_X = \{0.6, 0.4\}$.

Exercise 5.22.[2, p.106] Find a probability distribution $\{p_1, p_2, p_3, p_4\}$ such that there are *two* optimal codes that assign different lengths $\{l_i\}$ to the four symbols.

Exercise 5.23.[3] (Continuation of exercise 5.22.) Assume that the four probabilities $\{p_1, p_2, p_3, p_4\}$ are ordered such that $p_1 \geq p_2 \geq p_3 \geq p_4 \geq 0$. Let $\mathcal{Q}$ be the set of all probability vectors $\mathbf{p}$ such that there are *two* optimal codes with different lengths. Give a complete description of $\mathcal{Q}$. Find three probability vectors $\mathbf{q}^{(1)}$, $\mathbf{q}^{(2)}$, $\mathbf{q}^{(3)}$, which are the convex hull of $\mathcal{Q}$, i.e., such that any $\mathbf{p} \in \mathcal{Q}$ can be written as

$$\mathbf{p} = \mu_1 \mathbf{q}^{(1)} + \mu_2 \mathbf{q}^{(2)} + \mu_3 \mathbf{q}^{(3)}, \tag{5.29}$$

where $\{\mu_i\}$ are positive.

▷ Exercise 5.24.[1] Write a short essay discussing how to play the game of twenty questions optimally. [In twenty questions, one player thinks of an object, and the other player has to guess the object using as few binary questions as possible, preferably fewer than twenty.]

▷ Exercise 5.25.[2] Show that, if each probability p_i is equal to an integer power of 2 then there exists a source code whose expected length equals the entropy.

▷ Exercise 5.26.[2, p.106] Make ensembles for which the difference between the entropy and the expected length of the Huffman code is as big as possible.

▷ Exercise 5.27.[2, p.106] A source X has an alphabet of eleven characters

$$\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d},\mathtt{e},\mathtt{f},\mathtt{g},\mathtt{h},\mathtt{i},\mathtt{j},\mathtt{k}\},$$

all of which have equal probability, 1/11.

Find an optimal uniquely decodeable symbol code for this source. How much greater is the expected length of this optimal code than the entropy of X?

▷ Exercise 5.28.[2] Consider the optimal symbol code for an ensemble X with alphabet size I from which all symbols have identical probability $p = 1/I$. I is not a power of 2.

Show that the fraction f^+ of the I symbols that are assigned codelengths equal to

$$l^+ \equiv \lceil \log_2 I \rceil \tag{5.30}$$

satisfies

$$f^+ = 2 - \frac{2^{l^+}}{I} \tag{5.31}$$

and that the expected length of the optimal symbol code is

$$L = l^+ - 1 + f^+. \tag{5.32}$$

By differentiating the excess length $\Delta L \equiv L - H(X)$ with respect to I, show that the excess length is bounded by

$$\Delta L \leq 1 - \frac{\ln(\ln 2)}{\ln 2} - \frac{1}{\ln 2} = 0.086. \tag{5.33}$$

Exercise 5.29.[2] Consider a sparse binary source with $\mathcal{P}_X = \{0.99, 0.01\}$. Discuss how Huffman codes could be used to compress this source *efficiently.* Estimate how many codewords your proposed solutions require.

▷ Exercise 5.30.[2] *Scientific American* carried the following puzzle in 1975.

The poisoned glass. 'Mathematicians are curious birds', the police commissioner said to his wife. 'You see, we had all those partly filled glasses lined up in rows on a table in the hotel kitchen. Only one contained poison, and we wanted to know which one before searching that glass for fingerprints. Our lab could test the liquid in each glass, but the tests take time and money, so we wanted to make as few of them as possible by simultaneously testing mixtures of small samples from groups of glasses. The university sent over a

> mathematics professor to help us. He counted the glasses, smiled and said:
>
> ‘ “Pick any glass you want, Commissioner. We’ll test it first.”
>
> ‘ “But won’t that waste a test?” I asked.
>
> ‘ “No,” he said, “it’s part of the best procedure. We can test one glass first. It doesn’t matter which one.” ’
>
> ‘How many glasses were there to start with?’ the commissioner’s wife asked.
>
> ‘I don’t remember. Somewhere between 100 and 200.’
>
> What was the exact number of glasses?

Solve this puzzle and then explain why the professor was in fact wrong and the commissioner was right. What is in fact the optimal procedure for identifying the one poisoned glass? What is the expected waste relative to this optimum if one followed the professor’s strategy? Explain the relationship to symbol coding.

Exercise 5.31.[*2*, p.106] Assume that a sequence of symbols from the ensemble X introduced at the beginning of this chapter is compressed using the code C_3. Imagine picking one bit at random from the binary encoded sequence $\mathbf{c} = c(x_1)c(x_2)c(x_3)\ldots$. What is the probability that this bit is a 1?

C_3:

a_i	$c(a_i)$	p_i	$h(p_i)$	l_i
`a`	`0`	$1/2$	1.0	1
`b`	`10`	$1/4$	2.0	2
`c`	`110`	$1/8$	3.0	3
`d`	`111`	$1/8$	3.0	3

▷ Exercise 5.32.[*2*, p.107] How should the binary Huffman encoding scheme be modified to make optimal symbol codes in an encoding alphabet with q symbols? (Also known as ‘radix q’.)

Mixture codes

It is a tempting idea to construct a ‘metacode’ from several symbol codes that assign different-length codewords to the alternative symbols, then switch from one code to another, choosing whichever assigns the shortest codeword to the current symbol. Clearly we cannot do this for free. If one wishes to choose between two codes, then it is necessary to lengthen the message in a way that indicates which of the two codes is being used. If we indicate this choice by a single leading bit, it will be found that the resulting code is suboptimal because it is incomplete (that is, it fails the Kraft equality).

Exercise 5.33.[*3*, p.108] Prove that this metacode is incomplete, and explain why this combined code is suboptimal.

▶ 5.9 Solutions

Solution to exercise 5.8 (p.93). Yes, $C_2 = \{\mathtt{1}, \mathtt{101}\}$ is uniquely decodeable, even though it is not a prefix code, because no two different strings can map onto the same string; only the codeword $c(a_2) = \mathtt{101}$ contains the symbol `0`.

Solution to exercise 5.14 (p.95). We wish to prove that for any set of codeword lengths $\{l_i\}$ satisfying the Kraft inequality, there is a prefix code having those lengths. This is readily proved by thinking of the codewords illustrated in figure 5.8 as being in a ‘codeword supermarket’, with size indicating cost. We imagine purchasing codewords one at a time, starting from the shortest codewords (i.e., the biggest purchases), using the budget shown at the right of figure 5.8. We start at one side of the codeword supermarket, say the

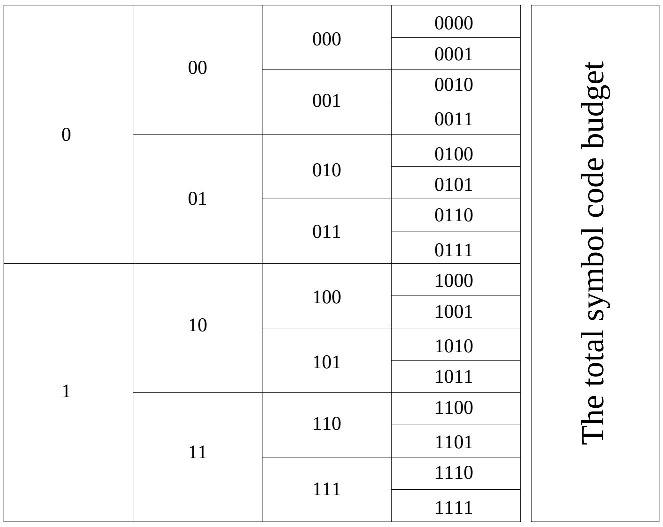


Figure 5.8. The codeword supermarket and the symbol coding budget. The 'cost' 2^{-l} of each codeword (with length l) is indicated by the size of the box it is written in. The total budget available when making a uniquely decodeable code is 1.

symbol	probability		Huffman codewords	Rival code's codewords	Modified rival code
a	p_a	▭	$\boxed{c_{\rm H}(a)}$	$\boxed{c_{\rm R}(a)}$	$\boxed{c_{\rm R}(c)}$
b	p_b	□	$\boxed{c_{\rm H}(b)}$	$\boxed{c_{\rm R}(b)}$	$\boxed{c_{\rm R}(b)}$
c	p_c	▭	$\boxed{c_{\rm H}(c)}$	$\boxed{c_{\rm R}(c)}$	$\boxed{c_{\rm R}(a)}$

Figure 5.9. Proof that Huffman coding makes an optimal symbol code. We assume that the rival code, which is said to be optimal, assigns *unequal* length codewords to the two symbols with smallest probability, a and b. By interchanging codewords a and c of the rival code, where c is a symbol with rival codelength as long as b's, we can make a code better than the rival code. This shows that the rival code was not optimal.

top, and purchase the first codeword of the required length. We advance down the supermarket a distance 2^{-l}, and purchase the next codeword of the next required length, and so forth. Because the codeword lengths are getting longer, and the corresponding intervals are getting shorter, we can always buy an adjacent codeword to the latest purchase, so there is no wasting of the budget. Thus at the Ith codeword we have advanced a distance $\sum_{i=1}^{I} 2^{-l_i}$ down the supermarket; if $\sum 2^{-l_i} \leq 1$, we will have purchased all the codewords without running out of budget.

Solution to exercise 5.16 (p.99). The proof that Huffman coding is optimal depends on proving that the key step in the algorithm – the decision to give the two symbols with smallest probability equal encoded lengths – cannot lead to a larger expected length than any other code. We can prove this by contradiction.

Assume that the two symbols with smallest probability, called a and b, to which the Huffman algorithm would assign equal length codewords, do *not* have equal lengths in *any* optimal symbol code. The optimal symbol code is some other rival code in which these two codewords have unequal lengths l_a and l_b with $l_a < l_b$. Without loss of generality we can assume that this other code is a complete prefix code, because any codelengths of a uniquely decodeable code can be realized by a prefix code.

In this rival code, there must be some other symbol c whose probability p_c is greater than p_a and whose length in the rival code is greater than or equal to l_b, because the code for b must have an adjacent codeword of equal or greater length – a complete prefix code never has a solo codeword of the maximum length.

Consider exchanging the codewords of a and c (figure 5.9), so that a is encoded with the longer codeword that was c's, and c, which is more probable than a, gets the shorter codeword. Clearly this reduces the expected length of the code. The change in expected length is $(p_a - p_c)(l_c - l_a)$. Thus we have contradicted the assumption that the rival code is optimal. Therefore it is valid to give the two symbols with smallest probability equal encoded lengths. Huffman coding produces optimal symbol codes. □

Solution to exercise 5.21 (p.102). A Huffman code for X^2 where $\mathcal{A}_X = \{\texttt{0}, \texttt{1}\}$ and $\mathcal{P}_X = \{0.9, 0.1\}$ is $\{\texttt{00}, \texttt{01}, \texttt{10}, \texttt{11}\} \rightarrow \{\texttt{1}, \texttt{01}, \texttt{000}, \texttt{001}\}$. This code has $L(C, X^2) = 1.29$, whereas the entropy $H(X^2)$ is 0.938.

A Huffman code for X^3 is

$$\{\texttt{000}, \texttt{100}, \texttt{010}, \texttt{001}, \texttt{101}, \texttt{011}, \texttt{110}, \texttt{111}\} \rightarrow \\ \{\texttt{1}, \texttt{011}, \texttt{010}, \texttt{001}, \texttt{00000}, \texttt{00001}, \texttt{00010}, \texttt{00011}\}.$$

This has expected length $L(C, X^3) = 1.598$ whereas the entropy $H(X^3)$ is 1.4069.

A Huffman code for X^4 maps the sixteen source strings to the following codelengths:

$$\{\texttt{0000}, \texttt{1000}, \texttt{0100}, \texttt{0010}, \texttt{0001}, \texttt{1100}, \texttt{0110}, \texttt{0011}, \texttt{0101}, \texttt{1010}, \texttt{1001}, \texttt{1110}, \texttt{1101}, \\ \texttt{1011}, \texttt{0111}, \texttt{1111}\} \rightarrow \{1, 3, 3, 3, 4, 6, 7, 7, 7, 7, 7, 9, 9, 9, 10, 10\}.$$

This has expected length $L(C, X^4) = 1.9702$ whereas the entropy $H(X^4)$ is 1.876.

When $\mathcal{P}_X = \{0.6, 0.4\}$, the Huffman code for X^2 has lengths $\{2, 2, 2, 2\}$; the expected length is 2 bits, and the entropy is 1.94 bits. A Huffman code for X^4 is shown in table 5.10. The expected length is 3.92 bits, and the entropy is 3.88 bits.

a_i	p_i	l_i	$c(a_i)$
0000	0.1296	3	000
0001	0.0864	4	0100
0010	0.0864	4	0110
0100	0.0864	4	0111
1000	0.0864	3	100
1100	0.0576	4	1010
1010	0.0576	4	1100
1001	0.0576	4	1101
0110	0.0576	4	1110
0101	0.0576	4	1111
0011	0.0576	4	0010
1110	0.0384	5	00110
1101	0.0384	5	01010
1011	0.0384	5	01011
0111	0.0384	4	1011
1111	0.0256	5	00111

Table 5.10. Huffman code for X^4 when $p_0 = 0.6$. Column 3 shows the assigned codelengths and column 4 the codewords. Some strings whose probabilities are identical, e.g., the fourth and fifth, receive different codelengths.

Solution to exercise 5.22 (p.102). The set of probabilities $\{p_1, p_2, p_3, p_4\} = \{1/6, 1/6, 1/3, 1/3\}$ gives rise to two different optimal sets of codelengths, because at the second step of the Huffman coding algorithm we can choose any of the three possible pairings. We may either put them in a constant length code $\{\texttt{00}, \texttt{01}, \texttt{10}, \texttt{11}\}$ or the code $\{\texttt{000}, \texttt{001}, \texttt{01}, \texttt{1}\}$. Both codes have expected length 2.

Another solution is $\{p_1, p_2, p_3, p_4\} = \{1/5, 1/5, 1/5, 2/5\}$.

And a third is $\{p_1, p_2, p_3, p_4\} = \{1/3, 1/3, 1/3, 0\}$.

Solution to exercise 5.26 (p.103). Let $p_{\max}$ be the largest probability in $p_1, p_2, \ldots, p_I$. The difference between the expected length L and the entropy H can be no bigger than $\max(p_{\max}, 0.086)$ (Gallager, 1978).

See exercises 5.27–5.28 to understand where the curious 0.086 comes from.

Solution to exercise 5.27 (p.103). Length $-$ entropy $= 0.086$.

Solution to exercise 5.31 (p.104). There are two ways to answer this problem correctly, and one popular way to answer it incorrectly. Let's give the incorrect answer first:

Erroneous answer. "We can pick a random bit by first picking a random source symbol x_i with probability p_i, then picking a random bit from $c(x_i)$. If we define f_i to be the fraction of the bits of $c(x_i)$ that are 1s, we find

$$P(\text{bit is } \texttt{1}) = \sum_i p_i f_i \quad (5.34)$$

$$= 1/2 \times 0 + 1/4 \times 1/2 + 1/8 \times 2/3 + 1/8 \times 1 = 1/3.\text{"} \quad (5.35)$$

	a_i	$c(a_i)$	p_i	l_i
	a	0	1/2	1
C_3:	b	10	1/4	2
	c	110	1/8	3
	d	111	1/8	3

This answer is wrong because it falls for the bus-stop fallacy, which was introduced in exercise 2.35 (p.38): if buses arrive at random, and we are interested in 'the average time from one bus until the next', we must distinguish two possible averages: (a) the average time from a randomly chosen bus until the next; (b) the average time between the bus you just missed and the next bus. The second 'average' is twice as big as the first because, by waiting for a bus at a random time, you bias your selection of a bus in favour of buses that follow a large gap. You're unlikely to catch a bus that comes 10 seconds after a preceding bus! Similarly, the symbols `c` and `d` get encoded into longer-length binary strings than `a`, so when we pick a bit from the compressed string at random, we are more likely to land in a bit belonging to a `c` or a `d` than would be given by the probabilities p_i in the expectation (5.34). All the probabilities need to be scaled up by l_i, and renormalized.

Correct answer in the same style. Every time symbol x_i is encoded, l_i bits are added to the binary string, of which $f_i l_i$ are `1`s. The expected number of `1`s added per symbol is

$$\sum_i p_i f_i l_i; \tag{5.36}$$

and the expected total number of bits added per symbol is

$$\sum_i p_i l_i. \tag{5.37}$$

So the fraction of `1`s in the transmitted string is

$$\begin{aligned} P(\text{bit is } \mathtt{1}) &= \frac{\sum_i p_i f_i l_i}{\sum_i p_i l_i} && (5.38) \\ &= \frac{1/2 \times 0 + 1/4 \times 1 + 1/8 \times 2 + 1/8 \times 3}{7/4} = \frac{7/8}{7/4} = 1/2. \end{aligned}$$

For a general symbol code and a general ensemble, the expectation (5.38) is the correct answer. But in this case, we can use a more powerful argument.

Information-theoretic answer. The encoded string $\mathbf{c}$ is the output of an optimal compressor that compresses samples from X down to an expected length of $H(X)$ bits. We can't expect to compress this data any further. But if the probability $P(\text{bit is } \mathtt{1})$ were not equal to $1/2$ then it *would* be possible to compress the binary string further (using a block compression code, say). Therefore $P(\text{bit is } \mathtt{1})$ must be equal to $1/2$; indeed the probability of any sequence of l bits in the compressed stream taking on any particular value must be 2^{-l}. The output of a perfect compressor is always perfectly random bits.

To put it another way, if the probability $P(\text{bit is } \mathtt{1})$ were not equal to $1/2$, then the information content per bit of the compressed string would be at most $H_2(P(\mathtt{1}))$, which would be less than 1; but this contradicts the fact that we can recover the original data from $\mathbf{c}$, so the information content per bit of the compressed string must be $H(X)/L(C,X) = 1$.

Solution to exercise 5.32 (p.104). The general Huffman coding algorithm for an encoding alphabet with q symbols has one difference from the binary case. The process of combining q symbols into 1 symbol reduces the number of symbols by $q-1$. So if we start with A symbols, we'll only end up with a complete q-ary tree if $A \bmod (q-1)$ is equal to 1. Otherwise, we know that whatever prefix code we make, it must be an incomplete tree with a number of missing leaves equal, modulo $(q-1)$, to $A \bmod (q-1) - 1$. For example, if a ternary tree is built for eight symbols, then there will unavoidably be one missing leaf in the tree.

The optimal q-ary code is made by putting these extra leaves in the longest branch of the tree. This can be achieved by adding the appropriate number of symbols to the original source symbol set, all of these extra symbols having probability zero. The total number of leaves is then equal to $r(q-1)+1$, for some integer r. The symbols are then repeatedly combined by taking the q symbols with smallest probability and replacing them by a single symbol, as in the binary Huffman coding algorithm.

Solution to exercise 5.33 (p.104). We wish to show that a greedy metacode, which picks the code which gives the shortest encoding, is actually suboptimal, because it violates the Kraft inequality.

We'll assume that each symbol x is assigned lengths $l_k(x)$ by each of the candidate codes C_k. Let us assume there are K alternative codes and that we can encode which code is being used with a header of length $\log K$ bits. Then the metacode assigns lengths $l'(x)$ that are given by

$$l'(x) = \log_2 K + \min_k l_k(x). \tag{5.39}$$

We compute the Kraft sum:

$$S = \sum_x 2^{-l'(x)} = \frac{1}{K} \sum_x 2^{-\min_k l_k(x)}. \tag{5.40}$$

Let's divide the set $\mathcal{A}_X$ into non-overlapping subsets $\{\mathcal{A}_k\}_{k=1}^K$ such that subset $\mathcal{A}_k$ contains all the symbols x that the metacode sends via code k. Then

$$S = \frac{1}{K} \sum_k \sum_{x \in \mathcal{A}_k} 2^{-l_k(x)}. \tag{5.41}$$

Now if one sub-code k satisfies the Kraft equality $\sum_{x \in \mathcal{A}_X} 2^{-l_k(x)} = 1$, then it must be the case that

$$\sum_{x \in \mathcal{A}_k} 2^{-l_k(x)} \leq 1, \tag{5.42}$$

with equality only if all the symbols x are in $\mathcal{A}_k$, which would mean that we are only using one of the K codes. So

$$S \leq \frac{1}{K} \sum_{k=1}^K 1 = 1, \tag{5.43}$$

with equality only if equation (5.42) is an equality for all codes k. But it's impossible for all the symbols to be in *all* the non-overlapping subsets $\{\mathcal{A}_k\}_{k=1}^K$, so we can't have equality (5.42) holding for *all* k. So $S < 1$.

Another way of seeing that a mixture code is suboptimal is to consider the binary tree that it defines. Think of the special case of two codes. The first bit we send identifies which code we are using. Now, in a complete code, any subsequent binary string is a valid string. But once we know that we are using, say, code A, we know that what follows can only be a codeword corresponding to a symbol x whose encoding is shorter under code A than code B. So some strings are invalid continuations, and the mixture code is incomplete and suboptimal.

For further discussion of this issue and its relationship to probabilistic modelling read about 'bits back coding' in section 28.3 and in Frey (1998).

About Chapter 6

Before reading Chapter 6, you should have read the previous chapter and worked on most of the exercises in it.

We'll also make use of some Bayesian modelling ideas that arrived in the vicinity of exercise 2.8 (p.30).

6

Stream Codes

In this chapter we discuss two data compression schemes.

Arithmetic coding is a beautiful method that goes hand in hand with the philosophy that compression of data from a source entails probabilistic modelling of that source. As of 1999, the best compression methods for text files use arithmetic coding, and several state-of-the-art image compression systems use it too.

Lempel–Ziv coding is a 'universal' method, designed under the philosophy that we would like a single compression algorithm that will do a reasonable job for *any* source. In fact, for many real life sources, this algorithm's universal properties hold only in the limit of unfeasibly large amounts of data, but, all the same, Lempel–Ziv compression is widely used and often effective.

▶ 6.1 The guessing game

As a motivation for these two compression methods, consider the redundancy in a typical English text file. Such files have redundancy at several levels: for example, they contain the ASCII characters with non-equal frequency; certain consecutive pairs of letters are more probable than others; and entire words can be predicted given the context and a semantic understanding of the text.

To illustrate the redundancy of English, and a curious way in which it could be compressed, we can imagine a guessing game in which an English speaker repeatedly attempts to predict the next character in a text file.

For simplicity, let us assume that the allowed alphabet consists of the 26 upper case letters `A,B,C,..., Z` and a space '`-`'. The game involves asking the subject to guess the next character repeatedly, the only feedback being whether the guess is correct or not, until the character is correctly guessed. After a correct guess, we note the number of guesses that were made when the character was identified, and ask the subject to guess the next character in the same way.

One sentence gave the following result when a human was asked to guess a sentence. The numbers of guesses are listed below each character.

T	H	E	R	E	-	I	S	-	N	O	-	R	E	V	E	R	S	E	-	O	N	-	A	-	M	O	T	O	R	C	Y	C	L	E	-
1	1	1	5	1	1	2	1	1	2	1	1	15	1	17	1	1	1	2	1	3	2	1	2	2	7	1	1	1	1	4	1	1	1	1	1

Notice that in many cases, the next letter is guessed immediately, in one guess. In other cases, particularly at the start of syllables, more guesses are needed.

What do this game and these results offer us? First, they demonstrate the redundancy of English from the point of view of an English speaker. Second, this game might be used in a data compression scheme, as follows.

The string of numbers '1, 1, 1, 5, 1, ...', listed above, was obtained by presenting the text to the subject. The maximum number of guesses that the subject will make for a given letter is twenty-seven, so what the subject is doing for us is performing a time-varying mapping of the twenty-seven letters $\{\texttt{A}, \texttt{B}, \texttt{C}, \ldots, \texttt{Z}, -\}$ onto the twenty-seven numbers $\{1, 2, 3, \ldots, 27\}$, which we can view as symbols in a new alphabet. The total number of symbols has not been reduced, but since he uses some of these symbols much more frequently than others – for example, 1 and 2 – it should be easy to compress this new string of symbols.

How would the *uncompression* of the sequence of numbers '1, 1, 1, 5, 1, ...' work? At uncompression time, we do not have the original string 'THERE...', we have only the encoded sequence. Imagine that our subject has an absolutely identical twin who also plays the guessing game with us, as if we knew the source text. If we stop him whenever he has made a number of guesses equal to the given number, then he will have just guessed the correct letter, and we can then say 'yes, that's right', and move to the next character. Alternatively, if the identical twin is not available, we could design a compression system with the help of just one human as follows. We choose a window length L, that is, a number of characters of context to show the human. For every one of the 27^L possible strings of length L, we ask them, 'What would you predict is the next character?', and 'If that prediction were wrong, what would your next guesses be?'. After tabulating their answers to these 26×27^L questions, we could use two copies of these enormous tables at the encoder and the decoder in place of the two human twins. Such a language model is called an Lth order Markov model.

These systems are clearly unrealistic for practical compression, but they illustrate several principles that we will make use of now.

▶ 6.2 Arithmetic codes

When we discussed variable-length symbol codes, and the optimal Huffman algorithm for constructing them, we concluded by pointing out two practical and theoretical problems with Huffman codes (section 5.6).

These defects are rectified by *arithmetic codes*, which were invented by Elias, by Rissanen and by Pasco, and subsequently made practical by Witten *et al.* (1987). In an arithmetic code, the probabilistic modelling is clearly separated from the encoding operation. The system is rather similar to the guessing game. The human predictor is replaced by a *probabilistic model* of the source. As each symbol is produced by the source, the probabilistic model supplies a *predictive distribution* over all possible values of the next symbol, that is, a list of positive numbers $\{p_i\}$ that sum to one. If we choose to model the source as producing i.i.d. symbols with some known distribution, then the predictive distribution is the same every time; but arithmetic coding can with equal ease handle complex adaptive models that produce context-dependent predictive distributions. The predictive model is usually implemented in a computer program.

The encoder makes use of the model's predictions to create a binary string. The decoder makes use of an identical twin of the model (just as in the guessing game) to interpret the binary string.

Let the source alphabet be $\mathcal{A}_X = \{a_1, \ldots, a_I\}$, and let the Ith symbol a_I have the special meaning 'end of transmission'. The source spits out a sequence $x_1, x_2, \ldots, x_n, \ldots$. The source does *not* necessarily produce i.i.d. symbols. We will assume that a computer program is provided to the encoder that assigns a predictive probability distribution over a_i given the sequence that has occurred thus far, $P(x_n = a_i \mid x_1, \ldots, x_{n-1})$. The receiver has an identical program that produces the same predictive probability distribution $P(x_n = a_i \mid x_1, \ldots, x_{n-1})$.

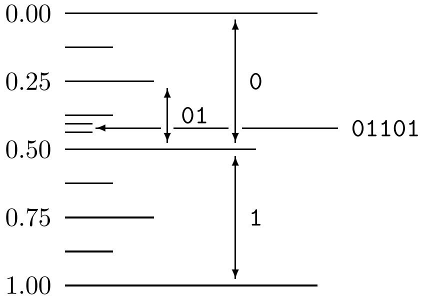


Figure 6.1. Binary strings define real intervals within the real line [0,1). We first encountered a picture like this when we discussed the symbol-code supermarket in Chapter 5.

Concepts for understanding arithmetic coding

Notation for intervals. The interval $[0.01, 0.10)$ is all numbers between 0.01 and 0.10, including $0.01\dot{0} \equiv 0.01000\ldots$ but not $0.10\dot{0} \equiv 0.10000\ldots$.

A binary transmission defines an interval within the real line from 0 to 1. For example, the string `01` is interpreted as a binary real number 0.01..., which corresponds to the interval $[0.01, 0.10)$ in binary, i.e., the interval $[0.25, 0.50)$ in base ten.

The longer string `01101` corresponds to a smaller interval [0.01101, 0.01110). Because `01101` has the first string, `01`, as a prefix, the new interval is a sub-interval of the interval $[0.01, 0.10)$. A one-megabyte binary file (2^{23} bits) is thus viewed as specifying a number between 0 and 1 to a precision of about two million decimal places – two million decimal digits, because each byte translates into a little more than two decimal digits.

Now, we can also divide the real line [0,1) into I intervals of lengths equal to the probabilities $P(x_1 = a_i)$, as shown in figure 6.2.

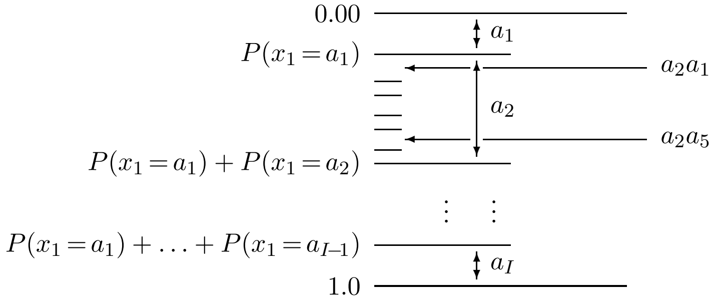

0.00
$P(x_1 = a_1)$
a_1
$a_2 a_1$
a_2
$a_2 a_5$
$P(x_1 = a_1) + P(x_1 = a_2)$
$P(x_1 = a_1) + \ldots + P(x_1 = a_{I-1})$
a_I
1.0

Figure 6.2. A probabilistic model defines real intervals within the real line [0,1).

We may then take each interval a_i and subdivide it into intervals denoted $a_i a_1, a_i a_2, \ldots, a_i a_I$, such that the length of $a_i a_j$ is proportional to $P(x_2 = a_j \mid x_1 = a_i)$. Indeed the length of the interval $a_i a_j$ will be precisely the joint probability

$$P(x_1 = a_i, x_2 = a_j) = P(x_1 = a_i) P(x_2 = a_j \mid x_1 = a_i). \tag{6.1}$$

Iterating this procedure, the interval $[0, 1)$ can be divided into a sequence of intervals corresponding to all possible finite length strings $x_1 x_2 \ldots x_N$, such that the length of an interval is equal to the probability of the string given our model.

```
$u$ := 0.0
$v$ := 1.0
$p$ := $v - u$
for $n$ = 1 to $N$ {
        Compute the cumulative probabilities $Q_n$ and $R_n$ (6.2, 6.3)
        $v$ := $u + pR_n(x_n \,|\, x_1, \ldots, x_{n-1})$
        $u$ := $u + pQ_n(x_n \,|\, x_1, \ldots, x_{n-1})$
        $p$ := $v - u$
}
```

Algorithm 6.3. Arithmetic coding. Iterative procedure to find the interval $[u, v)$ for the string $x_1x_2 \ldots x_N$.

Formulae describing arithmetic coding

The process depicted in figure 6.2 can be written explicitly as follows. The intervals are defined in terms of the lower and upper cumulative probabilities

$$Q_n(a_i \,|\, x_1, \ldots, x_{n-1}) \equiv \sum_{i'=1}^{i-1} P(x_n = a_{i'} \,|\, x_1, \ldots, x_{n-1}), \tag{6.2}$$

$$R_n(a_i \,|\, x_1, \ldots, x_{n-1}) \equiv \sum_{i'=1}^{i} P(x_n = a_{i'} \,|\, x_1, \ldots, x_{n-1}). \tag{6.3}$$

As the nth symbol arrives, we subdivide the $n-1$th interval at the points defined by Q_n and R_n. For example, starting with the first symbol, the intervals 'a_1', 'a_2', and 'a_I' are

$$a_1 \leftrightarrow [Q_1(a_1), R_1(a_1)) = [0, P(x_1 = a_1)), \tag{6.4}$$

$$a_2 \leftrightarrow [Q_1(a_2), R_1(a_2)) = [P(x = a_1), P(x = a_1) + P(x = a_2)), \tag{6.5}$$

and

$$a_I \leftrightarrow [Q_1(a_I), R_1(a_I)) = [P(x_1 = a_1) + \ldots + P(x_1 = a_{I-1}), 1.0). \tag{6.6}$$

Algorithm 6.3 describes the general procedure.

To encode a string $x_1x_2 \ldots x_N$, we locate the interval corresponding to $x_1x_2 \ldots x_N$, and send a binary string whose interval lies within that interval. This encoding can be performed on the fly, as we now illustrate.

Example: compressing the tosses of a bent coin

Imagine that we watch as a bent coin is tossed some number of times (cf. example 2.7 (p.30) and section 3.2 (p.51)). The two outcomes when the coin is tossed are denoted `a` and `b`. A third possibility is that the experiment is halted, an event denoted by the 'end of file' symbol, '□'. Because the coin is bent, we expect that the probabilities of the outcomes `a` and `b` are not equal, though beforehand we don't know which is the more probable outcome.

Encoding

Let the source string be '`bbba`□'. We pass along the string one symbol at a time and use our model to compute the probability distribution of the next symbol given the string thus far. Let these probabilities be:

Context (sequence thus far)	Probability of next symbol		
	$P(\mathtt{a}) = 0.425$	$P(\mathtt{b}) = 0.425$	$P(\square) = 0.15$
`b`	$P(\mathtt{a} \mid \mathtt{b}) = 0.28$	$P(\mathtt{b} \mid \mathtt{b}) = 0.57$	$P(\square \mid \mathtt{b}) = 0.15$
`bb`	$P(\mathtt{a} \mid \mathtt{bb}) = 0.21$	$P(\mathtt{b} \mid \mathtt{bb}) = 0.64$	$P(\square \mid \mathtt{bb}) = 0.15$
`bbb`	$P(\mathtt{a} \mid \mathtt{bbb}) = 0.17$	$P(\mathtt{b} \mid \mathtt{bbb}) = 0.68$	$P(\square \mid \mathtt{bbb}) = 0.15$
`bbba`	$P(\mathtt{a} \mid \mathtt{bbba}) = 0.28$	$P(\mathtt{b} \mid \mathtt{bbba}) = 0.57$	$P(\square \mid \mathtt{bbba}) = 0.15$

Figure 6.4 shows the corresponding intervals. The interval `b` is the middle 0.425 of $[0, 1)$. The interval `bb` is the middle 0.567 of `b`, and so forth.

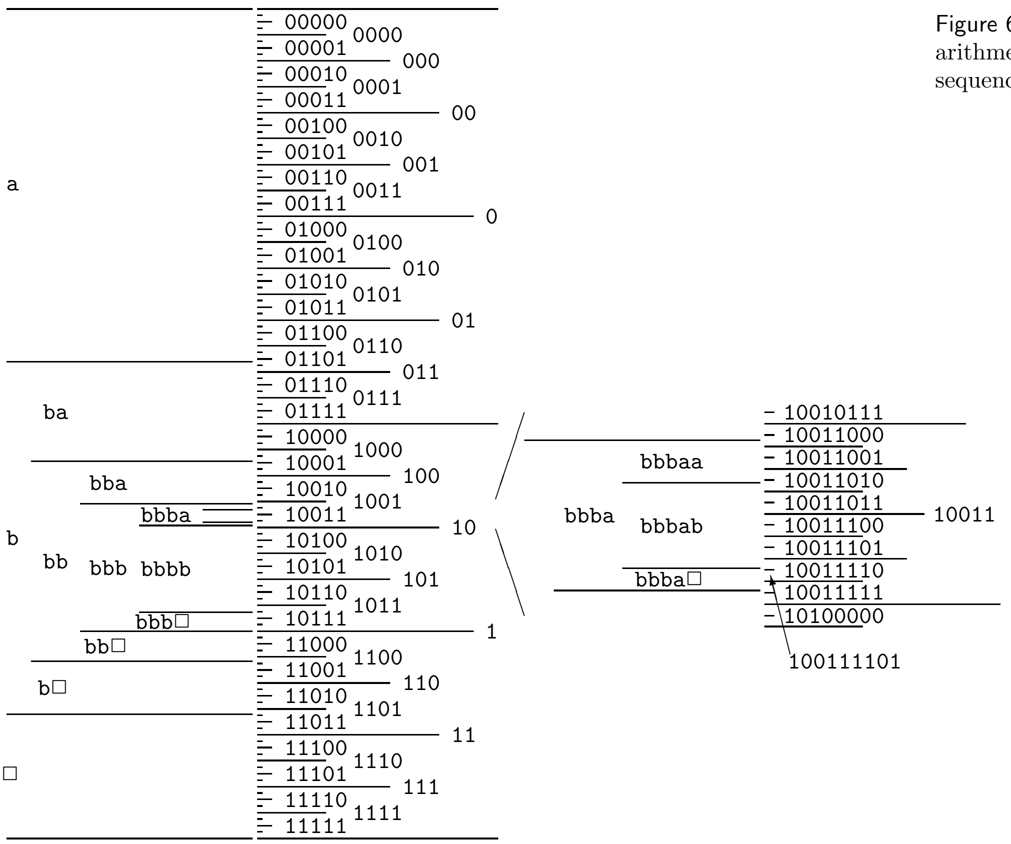


Figure 6.4. Illustration of the arithmetic coding process as the sequence `bbba□` is transmitted.

When the first symbol '`b`' is observed, the encoder knows that the encoded string will start '`01`', '`10`', or '`11`', but does not know which. The encoder writes nothing for the time being, and examines the next symbol, which is '`b`'. The interval '`bb`' lies wholly within interval '`1`', so the encoder can write the first bit: '`1`'. The third symbol '`b`' narrows down the interval a little, but not quite enough for it to lie wholly within interval '`10`'. Only when the next '`a`' is read from the source can we transmit some more bits. Interval '`bbba`' lies wholly within the interval '`1001`', so the encoder adds '`001`' to the '`1`' it has written. Finally when the '□' arrives, we need a procedure for terminating the encoding. Magnifying the interval '`bbba□`' (figure 6.4, right) we note that the marked interval '`100111101`' is wholly contained by `bbba□`, so the encoding can be completed by appending '`11101`'.

Exercise 6.1.[2, p.127] Show that the overhead required to terminate a message is never more than 2 bits, relative to the ideal message length given the probabilistic model $\mathcal{H}$, $h(\mathbf{x} \mid \mathcal{H}) = \log[1/P(\mathbf{x} \mid \mathcal{H})]$.

This is an important result. Arithmetic coding is very nearly optimal. The message length is always within two bits of the Shannon information content of the entire source string, so the expected message length is within two bits of the entropy of the entire message.

Decoding

The decoder receives the string '100111101' and passes along it one symbol at a time. First, the probabilities $P(\mathtt{a}), P(\mathtt{b}), P(\Box)$ are computed using the identical program that the encoder used and the intervals 'a', 'b' and '□' are deduced. Once the first two bits '10' have been examined, it is certain that the original string must have been started with a 'b', since the interval '10' lies wholly within interval 'b'. The decoder can then use the model to compute $P(\mathtt{a} \mid \mathtt{b}), P(\mathtt{b} \mid \mathtt{b}), P(\Box \mid \mathtt{b})$ and deduce the boundaries of the intervals 'ba', 'bb' and 'b□'. Continuing, we decode the second b once we reach '1001', the third b once we reach '100111', and so forth, with the unambiguous identification of 'bbba□' once the whole binary string has been read. With the convention that '□' denotes the end of the message, the decoder knows to stop decoding.

Transmission of multiple files

How might one use arithmetic coding to communicate several distinct files over the binary channel? Once the □ character has been transmitted, we imagine that the decoder is reset into its initial state. There is no transfer of the learnt statistics of the first file to the second file. If, however, we did believe that there is a relationship among the files that we are going to compress, we could define our alphabet differently, introducing a second end-of-file character that marks the end of the file but instructs the encoder and decoder to continue using the same probabilistic model.

The big picture

Notice that to communicate a string of N letters both the encoder and the decoder needed to compute only $N|\mathcal{A}|$ conditional probabilities – the probabilities of each possible letter in each context actually encountered – just as in the guessing game. This cost can be contrasted with the alternative of using a Huffman code with a large block size (in order to reduce the possible one-bit-per-symbol overhead discussed in section 5.6), where *all* block sequences that could occur must be considered and their probabilities evaluated.

Notice how flexible arithmetic coding is: it can be used with any source alphabet and any encoded alphabet. The size of the source alphabet and the encoded alphabet can change with time. Arithmetic coding can be used with any probability distribution, which can change utterly from context to context.

Furthermore, if we would like the symbols of the encoding alphabet (say, 0 and 1) to be used with *unequal* frequency, that can easily be arranged by subdividing the right-hand interval in proportion to the required frequencies.

How the probabilistic model might make its predictions

The technique of arithmetic coding does not force one to produce the predictive probability in any particular way, but the predictive distributions might

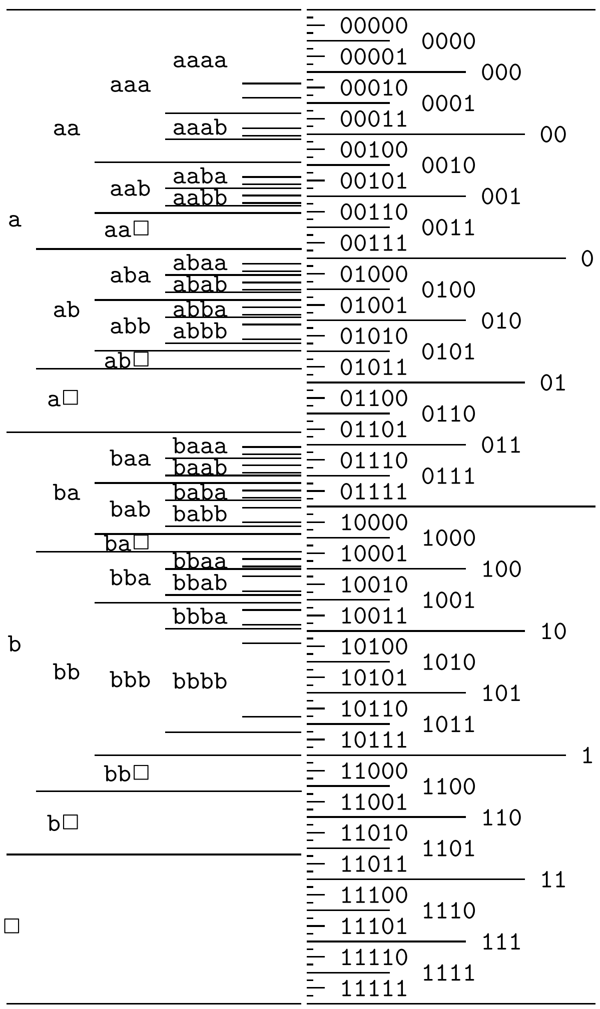


Figure 6.5. Illustration of the intervals defined by a simple Bayesian probabilistic model. The size of an intervals is proportional to the probability of the string. This model anticipates that the source is likely to be biased towards one of a and b, so sequences having lots of as or lots of bs have larger intervals than sequences of the same length that are 50:50 as and bs.

naturally be produced by a Bayesian model.

Figure 6.4 was generated using a simple model that always assigns a probability of 0.15 to □, and assigns the remaining 0.85 to a and b, divided in proportion to probabilities given by Laplace's rule,

$$P_{\rm L}(\mathtt{a}\,|\,x_1,\ldots,x_{n-1}) = \frac{F_{\mathtt{a}}+1}{F_{\mathtt{a}}+F_{\mathtt{b}}+2}, \tag{6.7}$$

where $F_{\mathtt{a}}(x_1,\ldots,x_{n-1})$ is the number of times that a has occurred so far, and $F_{\mathtt{b}}$ is the count of bs. These predictions correspond to a simple Bayesian model that expects and adapts to a non-equal frequency of use of the source symbols a and b within a file.

Figure 6.5 displays the intervals corresponding to a number of strings of length up to five. Note that if the string so far has contained a large number of bs then the probability of b relative to a is increased, and conversely if many as occur then as are made more probable. Larger intervals, remember, require fewer bits to encode.

Details of the Bayesian model

Having emphasized that any model could be used – arithmetic coding is not wedded to any particular set of probabilities – let me explain the simple adaptive probabilistic model used in the preceding example; we first encountered this model in exercise 2.8 (p.30).

Assumptions

The model will be described using parameters $p_\Box$, $p_{\mathtt{a}}$ and $p_{\mathtt{b}}$, defined below, which should not be confused with the predictive probabilities *in a particular context*, for example, $P(\mathtt{a}\,|\,\mathbf{s}=\mathtt{baa})$. A bent coin labelled a and b is tossed some number of times l, which we don't know beforehand. The coin's probability of coming up a when tossed is $p_{\mathtt{a}}$, and $p_{\mathtt{b}} = 1 - p_{\mathtt{a}}$; the parameters $p_{\mathtt{a}}, p_{\mathtt{b}}$ are not known beforehand. The source string $\mathbf{s} = \mathtt{baaba}\Box$ indicates that l was 5 and the sequence of outcomes was baaba.

1. It is assumed that the length of the string l has an exponential probability distribution
$$P(l) = (1 - p_\Box)^l p_\Box. \tag{6.8}$$
This distribution corresponds to assuming a constant probability $p_\Box$ for the termination symbol '□' at each character.

2. It is assumed that the non-terminal characters in the string are selected independently at random from an ensemble with probabilities $\mathcal{P} = \{p_{\mathtt{a}}, p_{\mathtt{b}}\}$; the probability $p_{\mathtt{a}}$ is fixed throughout the string to some unknown value that could be anywhere between 0 and 1. The probability of an a occurring as the next symbol, given $p_{\mathtt{a}}$ (if only we knew it), is $(1 - p_\Box)p_{\mathtt{a}}$. The probability, given $p_{\mathtt{a}}$, that an unterminated string of length F is a given string $\mathbf{s}$ that contains $\{F_{\mathtt{a}}, F_{\mathtt{b}}\}$ counts of the two outcomes is the Bernoulli distribution
$$P(\mathbf{s}\,|\,p_{\mathtt{a}}, F) = p_{\mathtt{a}}^{F_{\mathtt{a}}}(1 - p_{\mathtt{a}})^{F_{\mathtt{b}}}. \tag{6.9}$$

3. We assume a uniform prior distribution for $p_{\mathtt{a}}$,
$$P(p_{\mathtt{a}}) = 1, \quad p_{\mathtt{a}} \in [0, 1], \tag{6.10}$$
and define $p_{\mathtt{b}} \equiv 1 - p_{\mathtt{a}}$. It would be easy to assume other priors on $p_{\mathtt{a}}$, with beta distributions being the most convenient to handle.

This model was studied in section 3.2. The key result we require is the predictive distribution for the next symbol, given the string so far, $\mathbf{s}$. This probability that the next character is a or b (assuming that it is not '□') was derived in equation (3.16) and is precisely Laplace's rule (6.7).

▷ Exercise 6.2.[3] Compare the expected message length when an ASCII file is compressed by the following three methods.

Huffman-with-header. Read the whole file, find the empirical frequency of each symbol, construct a Huffman code for those frequencies, transmit the code by transmitting the lengths of the Huffman codewords, then transmit the file using the Huffman code. (The actual codewords don't need to be transmitted, since we can use a deterministic method for building the tree given the codelengths.)

Arithmetic code using the Laplace model.
$$P_{\rm L}(\mathtt{a}\,|\,x_1, \ldots, x_{n-1}) = \frac{F_{\mathtt{a}} + 1}{\sum_{\mathtt{a}'}(F_{\mathtt{a}'} + 1)}. \tag{6.11}$$

Arithmetic code using a Dirichlet model. This model's predictions are:
$$P_{\rm D}(\mathtt{a}\,|\,x_1, \ldots, x_{n-1}) = \frac{F_{\mathtt{a}} + \alpha}{\sum_{\mathtt{a}'}(F_{\mathtt{a}'} + \alpha)}, \tag{6.12}$$

where α is fixed to a number such as 0.01. A small value of α corresponds to a more responsive version of the Laplace model; the probability over characters is expected to be more nonuniform; $\alpha = 1$ reproduces the Laplace model.

Take care that the header of your Huffman message is self-delimiting. Special cases worth considering are (a) short files with just a few hundred characters; (b) large files in which some characters are never used.

▶ 6.3 Further applications of arithmetic coding

Efficient generation of random samples

Arithmetic coding not only offers a way to compress strings believed to come from a given model; it also offers a way to generate random strings from a model. Imagine sticking a pin into the unit interval at random, that line having been divided into subintervals in proportion to probabilities p_i; the probability that your pin will lie in interval i is p_i.

So to generate a sample from a model, all we need to do is feed ordinary random bits into an arithmetic *decoder* for that model. An infinite random bit sequence corresponds to the selection of a point at random from the line $[0, 1)$, so the decoder will then select a string at random from the assumed distribution. This arithmetic method is guaranteed to use very nearly the smallest number of random bits possible to make the selection – an important point in communities where random numbers are expensive! [This is not a joke. Large amounts of money are spent on generating random bits in software and hardware. Random numbers are valuable.]

A simple example of the use of this technique is in the generation of random bits with a nonuniform distribution $\{p_0, p_1\}$.

Exercise 6.3.[2, p.128] Compare the following two techniques for generating random symbols from a nonuniform distribution $\{p_0, p_1\} = \{0.99, 0.01\}$:

(a) The standard method: use a standard random number generator to generate an integer between 1 and 2^{32}. Rescale the integer to $(0, 1)$. Test whether this uniformly distributed random variable is less than 0.99, and emit a `0` or `1` accordingly.

(b) Arithmetic coding using the correct model, fed with standard random bits.

Roughly how many random bits will each method use to generate a thousand samples from this sparse distribution?

Efficient data-entry devices

When we enter text into a computer, we make gestures of some sort – maybe we tap a keyboard, or scribble with a pointer, or click with a mouse; an *efficient* text entry system is one where the number of gestures required to enter a given text string is *small*.

Compression:
text → bits

Writing:
text ← gestures

Writing can be viewed as an inverse process to data compression. In data compression, the aim is to map a given text string into a *small* number of bits. In text entry, we want a small sequence of gestures to produce our intended text.

By inverting an arithmetic coder, we can obtain an information-efficient text entry device that is driven by continuous pointing gestures (Ward *et al.*, 2000). In this system, called Dasher, the user zooms in on the unit interval to locate the interval corresponding to their intended string, in the same style as figure 6.4. A language model (exactly as used in text compression) controls the sizes of the intervals such that probable strings are quick and easy to identify. After an hour's practice, a novice user can write with one finger driving Dasher at about 25 words per minute – that's about half their normal ten-finger typing speed on a regular keyboard. It's even possible to write at 25 words per minute, *hands-free*, using gaze direction to drive Dasher (Ward and MacKay, 2002). Dasher is available as free software for various platforms.[1]

▶ 6.4 Lempel–Ziv coding

The Lempel–Ziv algorithms, which are widely used for data compression (e.g., the `compress` and `gzip` commands), are different in philosophy to arithmetic coding. There is no separation between modelling and coding, and no opportunity for explicit modelling.

Basic Lempel–Ziv algorithm

The method of compression is to replace a substring with a pointer to an earlier occurrence of the same substring. For example if the string is 1011010100010. . . , we parse it into an ordered *dictionary* of substrings that have not appeared before as follows: λ, 1, 0, 11, 01, 010, 00, 10, ... We include the empty substring λ as the first substring in the dictionary and order the substrings in the dictionary by the order in which they emerged from the source. After every comma, we look along the next part of the input sequence until we have read a substring that has not been marked off before. A moment's reflection will confirm that this substring is longer by one bit than a substring that has occurred earlier in the dictionary. This means that we can encode each substring by giving a *pointer* to the earlier occurrence of that prefix and then sending the extra bit by which the new substring in the dictionary differs from the earlier substring. If, at the nth bit, we have enumerated $s(n)$ substrings, then we can give the value of the pointer in $\lceil \log_2 s(n) \rceil$ bits. The code for the above sequence is then as shown in the fourth line of the following table (with punctuation included for clarity), the upper lines indicating the source string and the value of $s(n)$:

source substrings	λ	1	0	11	01	010	00	10
$s(n)$	0	1	2	3	4	5	6	7
$s(n)_{\text{binary}}$	000	001	010	011	100	101	110	111
(pointer, bit)		$(, 1)$	$(0, 0)$	$(01, 1)$	$(10, 1)$	$(100, 0)$	$(010, 0)$	$(001, 0)$

Notice that the first pointer we send is empty, because, given that there is only one substring in the dictionary – the string λ – no bits are needed to convey the 'choice' of that substring as the prefix. The encoded string is 100011101100001000010. The encoding, in this simple case, is actually a longer string than the source string, because there was no obvious redundancy in the source string.

▷ Exercise 6.4.[2] Prove that *any* uniquely decodeable code from $\{0, 1\}^+$ to $\{0, 1\}^+$ necessarily makes some strings longer if it makes some strings shorter.

[1] http://www.inference.phy.cam.ac.uk/dasher/

One reason why the algorithm described above lengthens a lot of strings is because it is inefficient – it transmits unnecessary bits; to put it another way, its code is not complete. Once a substring in the dictionary has been joined there by both of its children, then we can be sure that it will not be needed (except possibly as part of our protocol for terminating a message); so at that point we could drop it from our dictionary of substrings and shuffle them all along one, thereby reducing the length of subsequent pointer messages. Equivalently, we could write the second prefix into the dictionary at the point previously occupied by the parent. A second unnecessary overhead is the transmission of the new bit in these cases – the second time a prefix is used, we can be sure of the identity of the next bit.

Decoding

The decoder again involves an identical twin at the decoding end who constructs the dictionary of substrings as the data are decoded.

▷ Exercise 6.5.[*2*, p.128] Encode the string 000000000000100000000000 using the basic Lempel–Ziv algorithm described above.

▷ Exercise 6.6.[*2*, p.128] Decode the string

0010101110110010010001101010101000011

that was encoded using the basic Lempel–Ziv algorithm.

Practicalities

In this description I have not discussed the method for terminating a string.

There are many variations on the Lempel–Ziv algorithm, all exploiting the same idea but using different procedures for dictionary management, etc. The resulting programs are fast, but their performance on compression of English text, although useful, does not match the standards set in the arithmetic coding literature.

Theoretical properties

In contrast to the block code, Huffman code, and arithmetic coding methods we discussed in the last three chapters, the Lempel–Ziv algorithm is defined without making any mention of a probabilistic model for the source. Yet, given any ergodic source (i.e., one that is memoryless on sufficiently long timescales), the Lempel–Ziv algorithm can be proven *asymptotically* to compress down to the entropy of the source. This is why it is called a ‘universal’ compression algorithm. For a proof of this property, see Cover and Thomas (1991).

It achieves its compression, however, only by *memorizing* substrings that have happened so that it has a short name for them the next time they occur. The asymptotic timescale on which this universal performance is achieved may, for many sources, be unfeasibly long, because the number of typical substrings that need memorizing may be enormous. The useful performance of the algorithm in practice is a reflection of the fact that many files contain multiple repetitions of particular short sequences of characters, a form of redundancy to which the algorithm is well suited.

Common ground

I have emphasized the difference in philosophy behind arithmetic coding and Lempel–Ziv coding. There is common ground between them, though: in principle, one can design adaptive probabilistic models, and thence arithmetic codes, that are 'universal', that is, models that will asymptotically compress *any source in some class* to within some factor (preferably 1) of its entropy. However, for practical purposes, I think such universal models can only be constructed if the class of sources is severely restricted. A general purpose compressor that can discover the probability distribution of *any* source would be a general purpose artificial intelligence! A general purpose artificial intelligence does not yet exist.

▶ 6.5 Demonstration

An interactive aid for exploring arithmetic coding, `dasher.tcl`, is available.[2]

A demonstration arithmetic-coding software package written by Radford Neal[3] consists of encoding and decoding modules to which the user adds a module defining the probabilistic model. It should be emphasized that there is no single general-purpose arithmetic-coding compressor; a new model has to be written for each type of source. Radford Neal's package includes a simple adaptive model similar to the Bayesian model demonstrated in section 6.2. The results using this Laplace model should be viewed as a basic benchmark since it is the simplest possible probabilistic model – it simply assumes the characters in the file come independently from a fixed ensemble. The counts $\{F_i\}$ of the symbols $\{a_i\}$ are rescaled and rounded as the file is read such that all the counts lie between 1 and 256.

A state-of-the-art compressor for documents containing text and images, `DjVu`, uses arithmetic coding.[4] It uses a carefully designed approximate arithmetic coder for binary alphabets called the Z-coder (Bottou *et al.*, 1998), which is much faster than the arithmetic coding software described above. One of the neat tricks the Z-coder uses is this: the adaptive model adapts only occasionally (to save on computer time), with the decision about when to adapt being pseudo-randomly controlled by whether the arithmetic encoder emitted a bit.

The JBIG image compression standard for binary images uses arithmetic coding with a context-dependent model, which adapts using a rule similar to Laplace's rule. PPM (Teahan, 1995) is a leading method for text compression, and it uses arithmetic coding.

There are many Lempel–Ziv-based programs. `gzip` is based on a version of Lempel–Ziv called '`LZ77`' (Ziv and Lempel, 1977). `compress` is based on '`LZW`' (Welch, 1984). In my experience the best is `gzip`, with `compress` being inferior on most files.

`bzip` is a *block-sorting file compressor*, which makes use of a neat hack called the *Burrows–Wheeler transform* (Burrows and Wheeler, 1994). This method is not based on an explicit probabilistic model, and it only works well for files larger than several thousand characters; but in practice it is a very effective compressor for files in which the context of a character is a good predictor for that character.[5]

[2] `http://www.inference.phy.cam.ac.uk/mackay/itprnn/softwareI.html`

[3] `ftp://ftp.cs.toronto.edu/pub/radford/www/ac.software.html`

[4] `http://www.djvuzone.org/`

[5] There is a lot of information about the Burrows–Wheeler transform on the net. `http://dogma.net/DataCompression/BWT.shtml`

Compression of a text file

Table 6.6 gives the computer time in seconds taken and the compression achieved when these programs are applied to the LaTeX file containing the text of this chapter, of size 20,942 bytes.

Method	Compression time / sec	Compressed size (%age of 20,942)	Uncompression time / sec
Laplace model	0.28	12 974 (61%)	0.32
`gzip`	0.10	8 177 (39%)	**0.01**
`compress`	0.05	10 816 (51%)	0.05
`bzip`		7 495 (36%)	
`bzip2`		7 640 (36%)	
`ppmz`		**6 800 (32%)**	

Table 6.6. Comparison of compression algorithms applied to a text file.

Compression of a sparse file

Interestingly, `gzip` does not always do so well. Table 6.7 gives the compression achieved when these programs are applied to a text file containing 10^6 characters, each of which is either `0` and `1` with probabilities 0.99 and 0.01. The Laplace model is quite well matched to this source, and the benchmark arithmetic coder gives good performance, followed closely by `compress`; `gzip` is worst. An ideal model for this source would compress the file into about $10^6 H_2(0.01)/8 \simeq 10\,100$ bytes. The Laplace-model compressor falls short of this performance because it is implemented using only eight-bit precision. The `ppmz` compressor compresses the best of all, but takes much more computer time.

Method	Compression time / sec	Compressed size / bytes	Uncompression time / sec
Laplace model	0.45	14 143 (1.4%)	0.57
`gzip`	0.22	20 646 (2.1%)	0.04
`gzip --best+`	1.63	15 553 (1.6%)	0.05
`compress`	0.13	14 785 (1.5%)	0.03
`bzip`	0.30	10 903 (1.09%)	0.17
`bzip2`	0.19	11 260 (1.12%)	0.05
`ppmz`	533	**10 447 (1.04%)**	535

Table 6.7. Comparison of compression algorithms applied to a random file of 10^6 characters, 99% `0`s and 1% `1`s.

▶ 6.6 Summary

In the last three chapters we have studied three classes of data compression codes.

Fixed-length block codes (Chapter 4). These are mappings from a fixed number of source symbols to a fixed-length binary message. Only a tiny fraction of the source strings are given an encoding. These codes were fun for identifying the entropy as the measure of compressibility but they are of little practical use.

Symbol codes (Chapter 5). Symbol codes employ a variable-length code for each symbol in the source alphabet, the codelengths being integer lengths determined by the probabilities of the symbols. Huffman's algorithm constructs an optimal symbol code for a given set of symbol probabilities.

Every source string has a uniquely decodeable encoding, and if the source symbols come from the assumed distribution then the symbol code will compress to an expected length per character L lying in the interval $[H, H+1)$. Statistical fluctuations in the source may make the actual length longer or shorter than this mean length.

If the source is not well matched to the assumed distribution then the mean length is increased by the relative entropy D_{KL} between the source distribution and the code's implicit distribution. For sources with small entropy, the symbol has to emit at least one bit per source symbol; compression below one bit per source symbol can be achieved only by the cumbersome procedure of putting the source data into blocks.

Stream codes. The distinctive property of stream codes, compared with symbol codes, is that they are not constrained to emit at least one bit for every symbol read from the source stream. So large numbers of source symbols may be coded into a smaller number of bits. This property could be obtained using a symbol code only if the source stream were somehow chopped into blocks.

- Arithmetic codes combine a probabilistic model with an encoding algorithm that identifies each string with a sub-interval of $[0, 1)$ of size equal to the probability of that string under the model. This code is almost optimal in the sense that the compressed length of a string $\mathbf{x}$ closely matches the Shannon information content of $\mathbf{x}$ given the probabilistic model. Arithmetic codes fit with the philosophy that good compression requires *data modelling*, in the form of an adaptive Bayesian model.
- Lempel–Ziv codes are adaptive in the sense that they memorize strings that have already occurred. They are built on the philosophy that we don't know anything at all about what the probability distribution of the source will be, and we want a compression algorithm that will perform reasonably well whatever that distribution is.

Both arithmetic codes and Lempel–Ziv codes will fail to decode correctly if any of the bits of the compressed file are altered. So if compressed files are to be stored or transmitted over noisy media, error-correcting codes will be essential. Reliable communication over unreliable channels is the topic of Part II.

▶ 6.7 Exercises on stream codes

Exercise 6.7.$^{[2]}$ Describe an arithmetic coding algorithm to encode random bit strings of length N and weight K (i.e., K ones and $N-K$ zeroes) where N and K are given.

For the case $N=5$, $K=2$, show in detail the intervals corresponding to all source substrings of lengths 1–5.

▷ Exercise 6.8.$^{[2,\ \mathrm{p.128}]}$ How many bits are needed to specify a selection of K objects from N objects? (N and K are assumed to be known and the selection of K objects is unordered.) How might such a selection be made at random without being wasteful of random bits?

▷ Exercise 6.9.[2] A binary source X emits independent identically distributed symbols with probability distribution $\{f_0, f_1\}$, where $f_1 = 0.01$. Find an optimal uniquely-decodeable symbol code for a string $\mathbf{x} = x_1x_2x_3$ of **three** successive samples from this source.

Estimate (to one decimal place) the factor by which the expected length of this optimal code is greater than the entropy of the three-bit string $\mathbf{x}$.

$[H_2(0.01) \simeq 0.08$, where $H_2(x) = x\log_2(1/x) + (1-x)\log_2(1/(1-x)).]$

An arithmetic code is used to compress a string of 1000 samples from the source X. Estimate the mean and standard deviation of the length of the compressed file.

▷ Exercise 6.10.[2] Describe an arithmetic coding algorithm to generate random bit strings of length N with density f (i.e., each bit has probability f of being a one) where N is given.

Exercise 6.11.[2] Use a modified Lempel–Ziv algorithm in which, as discussed on p.120, the dictionary of prefixes is pruned by writing new prefixes into the space occupied by prefixes that will not be needed again. Such prefixes can be identified when both their children have been added to the dictionary of prefixes. (You may neglect the issue of termination of encoding.) Use this algorithm to encode the string `0100001000100010101000001`. Highlight the bits that follow a prefix on the second occasion that that prefix is used. (As discussed earlier, these bits could be omitted.)

Exercise 6.12.[2, p.128] Show that this modified Lempel–Ziv code is still not 'complete', that is, there are binary strings that are not encodings of any string.

▷ Exercise 6.13.[3, p.128] Give examples of simple sources that have low entropy but would not be compressed well by the Lempel–Ziv algorithm.

▶ 6.8 Further exercises on data compression

The following exercises may be skipped by the reader who is eager to learn about noisy channels.

Exercise 6.14.[3, p.130] Consider a Gaussian distribution in N dimensions,

$$P(\mathbf{x}) = \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{\sum_n x_n^2}{2\sigma^2}\right). \tag{6.13}$$

Define the radius of a point $\mathbf{x}$ to be $r = \left(\sum_n x_n^2\right)^{1/2}$. Estimate the mean and variance of the square of the radius, $r^2 = \left(\sum_n x_n^2\right)$.

You may find helpful the integral

$$\int \mathrm{d}x \, \frac{1}{(2\pi\sigma^2)^{1/2}} \, x^4 \exp\left(-\frac{x^2}{2\sigma^2}\right) = 3\sigma^4, \tag{6.14}$$

though you should be able to estimate the required quantities without it.

Assuming that N is large, show that nearly all the probability of a Gaussian is contained in a thin shell of radius $\sqrt{N}\sigma$. Find the thickness of the shell.

Evaluate the probability density (6.13) at a point in that thin shell and at the origin $\mathbf{x} = 0$ and compare. Use the case $N = 1000$ as an example.

Notice that nearly all the probability mass is located in a different part of the space from the region of highest probability density.

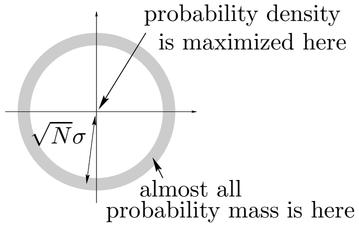


Figure 6.8. Schematic representation of the typical set of an N-dimensional Gaussian distribution.

Exercise 6.15.$^{[2]}$ Explain what is meant by an *optimal binary symbol code*.

Find an optimal binary symbol code for the ensemble:

$$\mathcal{A} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}, \mathtt{h}, \mathtt{i}, \mathtt{j}\},$$

$$\mathcal{P} = \left\{ \frac{1}{100}, \frac{2}{100}, \frac{4}{100}, \frac{5}{100}, \frac{6}{100}, \frac{8}{100}, \frac{9}{100}, \frac{10}{100}, \frac{25}{100}, \frac{30}{100} \right\},$$

and compute the expected length of the code.

Exercise 6.16.$^{[2]}$ A string $\mathbf{y} = x_1 x_2$ consists of *two* independent samples from an ensemble

$$X : \mathcal{A}_X = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}; \mathcal{P}_X = \left\{ \frac{1}{10}, \frac{3}{10}, \frac{6}{10} \right\}.$$

What is the entropy of $\mathbf{y}$? Construct an optimal binary symbol code for the string $\mathbf{y}$, and find its expected length.

Exercise 6.17.$^{[2]}$ Strings of N independent samples from an ensemble with $\mathcal{P} = \{0.1, 0.9\}$ are compressed using an arithmetic code that is matched to that ensemble. Estimate the mean and standard deviation of the compressed strings' lengths for the case $N = 1000$. $[H_2(0.1) \simeq 0.47]$

Exercise 6.18.$^{[3]}$ Source coding with variable-length symbols.

> In the chapters on source coding, we assumed that we were encoding into a binary alphabet $\{\mathtt{0}, \mathtt{1}\}$ in which both symbols should be used with equal frequency. In this question we explore how the encoding alphabet should be used if the symbols take different times to transmit.

A poverty-stricken student communicates for free with a friend using a telephone by selecting an integer $n \in \{1, 2, 3 \ldots\}$, making the friend's phone ring n times, then hanging up in the middle of the nth ring. This process is repeated so that a string of symbols $n_1 n_2 n_3 \ldots$ is received. What is the optimal way to communicate? If large integers n are selected then the message takes longer to communicate. If only small integers n are used then the information content per symbol is small. We aim to maximize the rate of information transfer, per unit time.

Assume that the time taken to transmit a number of rings n and to redial is l_n seconds. Consider a probability distribution over n, $\{p_n\}$. Defining the average duration *per symbol* to be

$$L(\mathbf{p}) = \sum_n p_n l_n \qquad (6.15)$$

and the entropy *per symbol* to be

$$H(\mathbf{p}) = \sum_n p_n \log_2 \frac{1}{p_n}, \tag{6.16}$$

show that for the average information rate *per second* to be maximized, the symbols must be used with probabilities of the form

$$p_n = \frac{1}{Z} 2^{-\beta l_n} \tag{6.17}$$

where $Z = \sum_n 2^{-\beta l_n}$ and β satisfies the implicit equation

$$\beta = \frac{H(\mathbf{p})}{L(\mathbf{p})}, \tag{6.18}$$

that is, β is the rate of communication. Show that these two equations (6.17, 6.18) imply that β must be set such that

$$\log Z = 0. \tag{6.19}$$

Assuming that the channel has the property

$$l_n = n \text{ seconds}, \tag{6.20}$$

find the optimal distribution $\mathbf{p}$ and show that the maximal information rate is 1 bit per second.

How does this compare with the information rate per second achieved if $\mathbf{p}$ is set to $(1/2, 1/2, 0, 0, 0, 0, \ldots)$ — that is, only the symbols $n = 1$ and $n = 2$ are selected, and they have equal probability?

Discuss the relationship between the results (6.17, 6.19) derived above, and the Kraft inequality from source coding theory.

How might a random binary source be efficiently encoded into a sequence of symbols $n_1 n_2 n_3 \ldots$ for transmission over the channel defined in equation (6.20)?

▷ Exercise 6.19.[1] How many bits does it take to shuffle a pack of cards?

▷ Exercise 6.20.[2] In the card game Bridge, the four players receive 13 cards each from the deck of 52 and start each game by looking at their own hand and bidding. The legal bids are, in ascending order 1♣, 1♢, 1♡, 1♠, $1NT$, 2♣, 2♢, … 7♡, 7♠, $7NT$, and successive bids must follow this order; a bid of, say, 2♡ may only be followed by higher bids such as 2♠ or 3♣ or $7NT$. (Let us neglect the ‘double’ bid.)

The players have several aims when bidding. One of the aims is for two partners to communicate to each other as much as possible about what cards are in their hands.

Let us concentrate on this task.

(a) After the cards have been dealt, how many bits are needed for North to convey to South what her hand is?

(b) Assuming that E and W do not bid at all, what is the maximum total information that N and S can convey to each other while bidding? Assume that N starts the bidding, and that once either N or S stops bidding, the bidding stops.

▷ Exercise 6.21.[2] My old ‘arabic’ microwave oven had 11 buttons for entering cooking times, and my new ‘roman’ microwave has just five. The buttons of the roman microwave are labelled ‘10 minutes’, ‘1 minute’, ‘10 seconds’, ‘1 second’, and ‘Start’; I’ll abbreviate these five strings to the symbols M, C, X, I, □. To enter one minute and twenty-three seconds (1:23), the arabic sequence is

$$\texttt{123}\square, \qquad (6.21)$$

and the roman sequence is

$$\texttt{CXXIII}\square. \qquad (6.22)$$

Each of these keypads defines a code mapping the 3599 cooking times from 0:01 to 59:59 into a string of symbols.

(a) Which times can be produced with two or three symbols? (For example, 0:20 can be produced by three symbols in either code: XX□ and 20□.)

(b) Are the two codes complete? Give a detailed answer.

(c) For each code, name a cooking time that it can produce in four symbols that the other code cannot.

(d) Discuss the implicit probability distributions over times to which each of these codes is best matched.

(e) Concoct a plausible probability distribution over times that a real user might use, and evaluate roughly the expected number of symbols, and maximum number of symbols, that each code requires. Discuss the ways in which each code is inefficient or efficient.

(f) Invent a more efficient cooking-time-encoding system for a microwave oven.

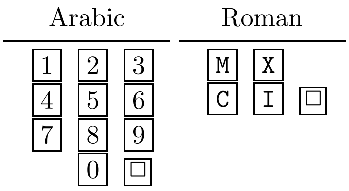


Figure 6.9. Alternative keypads for microwave ovens.

Exercise 6.22.[2, p.132] Is the standard binary representation for positive integers (e.g. $c_{\rm b}(5) = \texttt{101}$) a uniquely decodeable code?

Design a binary code for the positive integers, i.e., a mapping from $n \in \{1, 2, 3, \ldots\}$ to $c(n) \in \{\texttt{0}, \texttt{1}\}^+$, that is uniquely decodeable. Try to design codes that are prefix codes and that satisfy the Kraft equality $\sum_n 2^{-l_n} = 1$.

> Motivations: any data file terminated by a special end of file character can be mapped onto an integer, so a prefix code for integers can be used as a self-delimiting encoding of files too. Large files correspond to large integers. Also, one of the building blocks of a ‘universal’ coding scheme – that is, a coding scheme that will work OK for a large variety of sources – is the ability to encode integers. Finally, in microwave ovens, cooking times are positive integers!

Discuss criteria by which one might compare alternative codes for integers (or, equivalently, alternative self-delimiting codes for files).

▶ 6.9 Solutions

Solution to exercise 6.1 (p.115). The worst-case situation is when the interval to be represented lies just inside a binary interval. In this case, we may choose either of two binary intervals as shown in figure 6.10. These binary intervals

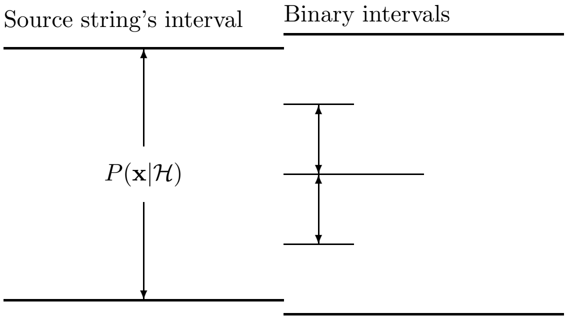


Figure 6.10. Termination of arithmetic coding in the worst case, where there is a two bit overhead. Either of the two binary intervals marked on the right-hand side may be chosen. These binary intervals are no smaller than $P(\mathbf{x}|\mathcal{H})/4$.

are no smaller than $P(\mathbf{x}|\mathcal{H})/4$, so the binary encoding has a length no greater than $\log_2 1/P(\mathbf{x}|\mathcal{H}) + \log_2 4$, which is two bits more than the ideal message length.

Solution to exercise 6.3 (p.118). The standard method uses 32 random bits per generated symbol and so requires 32 000 bits to generate one thousand samples.

Arithmetic coding uses on average about $H_2(0.01) = 0.081$ bits per generated symbol, and so requires about 83 bits to generate one thousand samples (assuming an overhead of roughly two bits associated with termination).

Fluctuations in the number of 1s would produce variations around this mean with standard deviation 21.

Solution to exercise 6.5 (p.120). The encoding is 010100110010110001100, which comes from the parsing

$$\texttt{0, 00, 000, 0000, 001, 00000, 000000} \tag{6.23}$$

which is encoded thus:

$$\texttt{(, 0), (1, 0), (10, 0), (11, 0), (010, 1), (100, 0), (110, 0).} \tag{6.24}$$

Solution to exercise 6.6 (p.120). The decoding is

0100001000100010101000001.

Solution to exercise 6.8 (p.123). This problem is equivalent to exercise 6.7 (p.123).

The selection of K objects from N objects requires $\lceil \log_2 \binom{N}{K} \rceil$ bits $\simeq NH_2(K/N)$ bits. This selection could be made using arithmetic coding. The selection corresponds to a binary string of length N in which the 1 bits represent which objects are selected. Initially the probability of a 1 is K/N and the probability of a 0 is $(N-K)/N$. Thereafter, given that the emitted string thus far, of length n, contains k 1s, the probability of a 1 is $(K-k)/(N-n)$ and the probability of a 0 is $1 - (K-k)/(N-n)$.

Solution to exercise 6.12 (p.124). This modified Lempel–Ziv code is still not 'complete', because, for example, after five prefixes have been collected, the pointer could be any of the strings 000, 001, 010, 011, 100, but it cannot be 101, 110 or 111. Thus there are some binary strings that cannot be produced as encodings.

Solution to exercise 6.13 (p.124). Sources with low entropy that are not well compressed by Lempel–Ziv include:

(a) Sources with some symbols that have long range correlations and intervening random junk. An ideal model should capture what's correlated and compress it. Lempel–Ziv can compress the correlated features only by memorizing all cases of the intervening junk. As a simple example, consider a telephone book in which every line contains an (old number, new number) pair:

```
285-3820:572-5892□
258-8302:593-2010□
```

The number of characters per line is 18, drawn from the 13-character alphabet $\{\texttt{0}, \texttt{1}, \ldots, \texttt{9}, -, :, \Box\}$. The characters '-', ':' and '□' occur in a predictable sequence, so the true information content per line, assuming all the phone numbers are seven digits long, and assuming that they are random sequences, is about 14 bans. (A ban is the information content of a random integer between 0 and 9.) A finite state language model could easily capture the regularities in these data. A Lempel–Ziv algorithm will take a long time before it compresses such a file down to 14 bans per line, however, because in order for it to 'learn' that the string :*ddd* is always followed by -, for any three digits *ddd*, it will have to *see* all those strings. So near-optimal compression will only be achieved after thousands of lines of the file have been read.

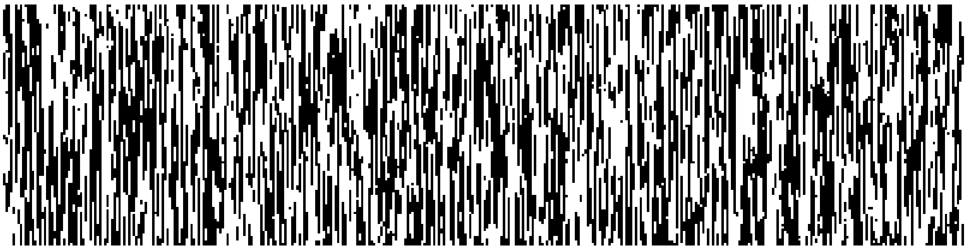


Figure 6.11. A source with low entropy that is not well compressed by Lempel–Ziv. The bit sequence is read from left to right. Each line differs from the line above in $f = 5\%$ of its bits. The image width is 400 pixels.

(b) Sources with long range correlations, for example two-dimensional images that are represented by a sequence of pixels, row by row, so that vertically adjacent pixels are a distance w apart in the source stream, where w is the image width. Consider, for example, a fax transmission in which each line is very similar to the previous line (figure 6.11). The true entropy is only $H_2(f)$ per pixel, where f is the probability that a pixel differs from its parent. Lempel–Ziv algorithms will only compress down to the entropy once *all* strings of length $2^w = 2^{400}$ have occurred and their successors have been memorized. There are only about 2^{300} particles in the universe, so we can confidently say that Lempel–Ziv codes will *never* capture the redundancy of such an image.

Another highly redundant texture is shown in figure 6.12. The image was made by dropping horizontal and vertical pins randomly on the plane. It contains both long-range vertical correlations and long-range horizontal correlations. There is no practical way that Lempel–Ziv, fed with a pixel-by-pixel scan of this image, could capture both these correlations.

Biological computational systems can readily identify the redundancy in these images and in images that are much more complex; thus we might anticipate that the best data compression algorithms will result from the development of artificial intelligence methods.

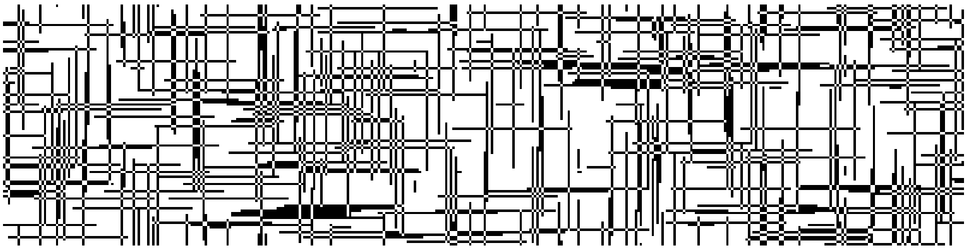


Figure 6.12. A texture consisting of horizontal and vertical pins dropped at random on the plane.

(c) Sources with intricate redundancy, such as files generated by computers. For example, a LaTeX file followed by its encoding into a PostScript file. The information content of this pair of files is roughly equal to the information content of the LaTeX file alone.

(d) A picture of the Mandelbrot set. The picture has an information content equal to the number of bits required to specify the range of the complex plane studied, the pixel sizes, and the colouring rule used.

(e) A picture of a ground state of a frustrated antiferromagnetic Ising model (figure 6.13), which we will discuss in Chapter 31. Like figure 6.12, this binary image has interesting correlations in two directions.

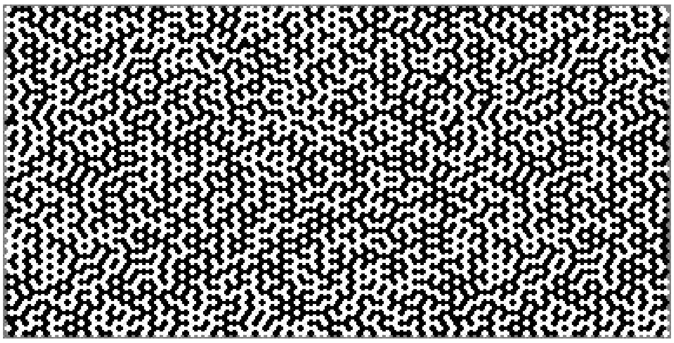


Figure 6.13. Frustrated triangular Ising model in one of its ground states.

(f) Cellular automata – figure 6.14 shows the state history of 100 steps of a cellular automaton with 400 cells. The update rule, in which each cell's new state depends on the state of five preceding cells, was selected at random. The information content is equal to the information in the boundary (400 bits), and the propagation rule, which here can be described in 32 bits. An optimal compressor will thus give a compressed file length which is essentially constant, independent of the vertical height of the image. Lempel–Ziv would only give this zero-cost compression once the cellular automaton has entered a periodic limit cycle, which could easily take about 2^{100} iterations.

In contrast, the JBIG compression method, which models the probability of a pixel given its local context and uses arithmetic coding, would do a good job on these images.

Solution to exercise 6.14 (p.124). For a one-dimensional Gaussian, the variance of x, $\mathcal{E}[x^2]$, is σ^2. So the mean value of r^2 in N dimensions, since the components of $\mathbf{x}$ are independent random variables, is

$$\mathcal{E}[r^2] = N\sigma^2. \tag{6.25}$$

Figure 6.14. The 100-step time-history of a cellular automaton with 400 cells.

The variance of r^2, similarly, is N times the variance of x^2, where x is a one-dimensional Gaussian variable.

$$\mathrm{var}(x^2) = \int \mathrm{d}x \, \frac{1}{(2\pi\sigma^2)^{1/2}} x^4 \exp\left(-\frac{x^2}{2\sigma^2}\right) - \sigma^4. \tag{6.26}$$

The integral is found to be $3\sigma^4$ (equation (6.14)), so $\mathrm{var}(x^2) = 2\sigma^4$. Thus the variance of r^2 is $2N\sigma^4$.

For large N, the central-limit theorem indicates that r^2 has a Gaussian distribution with mean $N\sigma^2$ and standard deviation $\sqrt{2N}\sigma^2$, so the probability density of r must similarly be concentrated about $r \simeq \sqrt{N}\sigma$.

The thickness of this shell is given by turning the standard deviation of r^2 into a standard deviation on r: for small $\delta r/r$, $\delta \log r = \delta r/r = ({1/2})\delta \log r^2 = ({1/2})\delta(r^2)/r^2$, so setting $\delta(r^2) = \sqrt{2N}\sigma^2$, r has standard deviation $\delta r = ({1/2}) r \delta(r^2)/r^2 = \sigma/\sqrt{2}$.

The probability density of the Gaussian at a point $\mathbf{x}_{\mathrm{shell}}$ where $r = \sqrt{N}\sigma$ is

$$P(\mathbf{x}_{\mathrm{shell}}) = \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{N\sigma^2}{2\sigma^2}\right) = \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{N}{2}\right). \tag{6.27}$$

Whereas the probability density at the origin is

$$P(\mathbf{x}\!=\!0) = \frac{1}{(2\pi\sigma^2)^{N/2}}. \tag{6.28}$$

Thus $P(\mathbf{x}_{\mathrm{shell}})/P(\mathbf{x}\!=\!0) = \exp(-N/2)$. The probability density at the typical radius is $e^{-N/2}$ times smaller than the density at the origin. If $N = 1000$, then the probability density at the origin is e^{500} times greater.

7

Codes for Integers

This chapter is an aside, which may safely be skipped.

Solution to exercise 6.22 (p.127)

To discuss the coding of integers we need some definitions.

The standard binary representation of a positive integer n will be denoted by $c_{\rm b}(n)$, e.g., $c_{\rm b}(5) = \texttt{101}$, $c_{\rm b}(45) = \texttt{101101}$.

The standard binary length of a positive integer n, $l_{\rm b}(n)$, is the length of the string $c_{\rm b}(n)$. For example, $l_{\rm b}(5) = 3$, $l_{\rm b}(45) = 6$.

The standard binary representation $c_{\rm b}(n)$ is *not* a uniquely decodeable code for integers since there is no way of knowing when an integer has ended. For example, $c_{\rm b}(5)c_{\rm b}(5)$ is identical to $c_{\rm b}(45)$. It would be uniquely decodeable if we knew the standard binary length of each integer before it was received.

Noticing that all positive integers have a standard binary representation that starts with a `1`, we might define another representation:

The headless binary representation of a positive integer n will be denoted by $c_{\rm B}(n)$, e.g., $c_{\rm B}(5) = \texttt{01}$, $c_{\rm B}(45) = \texttt{01101}$ and $c_{\rm B}(1) = \lambda$ (where λ denotes the null string).

This representation would be uniquely decodeable if we knew the length $l_{\rm b}(n)$ of the integer.

So, how can we make a uniquely decodeable code for integers? Two strategies can be distinguished.

1. **Self-delimiting codes**. We first communicate somehow the length of the integer, $l_{\rm b}(n)$, which is also a positive integer; then communicate the original integer n itself using $c_{\rm B}(n)$.

2. **Codes with 'end of file' characters**. We code the integer into blocks of length b bits, and reserve one of the 2^b symbols to have the special meaning 'end of file'. The coding of integers into blocks is arranged so that this reserved symbol is not needed for any other purpose.

The simplest uniquely decodeable code for integers is the unary code, which can be viewed as a code with an end of file character.

Unary code. An integer n is encoded by sending a string of $n-1$ `0`s followed by a `1`.

n	$c_{\rm U}(n)$
1	`1`
2	`01`
3	`001`
4	`0001`
5	`00001`
$\vdots$	
45	`000000000000000000000000000000000000000000001`

The unary code has length $l_{\rm U}(n) = n$.

The unary code is the optimal code for integers if the probability distribution over n is $p_{\rm U}(n) = 2^{-n}$.

Self-delimiting codes

We can use the unary code to encode the *length* of the binary encoding of n and make a self-delimiting code:

Code C_α. We send the unary code for $l_{\rm b}(n)$, followed by the headless binary representation of n.

$$c_\alpha(n) = c_{\rm U}[l_{\rm b}(n)]c_{\rm B}(n). \tag{7.1}$$

Table 7.1 shows the codes for some integers. The overlining indicates the division of each string into the parts $c_{\rm U}[l_{\rm b}(n)]$ and $c_{\rm B}(n)$. We might equivalently view $c_\alpha(n)$ as consisting of a string of $(l_{\rm b}(n) - 1)$ zeroes followed by the standard binary representation of n, $c_{\rm b}(n)$.

n	$c_{\rm b}(n)$	$l_{\rm b}(n)$	$c_\alpha(n)$
1	`1`	1	$\overline{\texttt{1}}$
2	`10`	2	$\overline{\texttt{01}}\texttt{0}$
3	`11`	2	$\overline{\texttt{01}}\texttt{1}$
4	`100`	3	$\overline{\texttt{001}}\texttt{00}$
5	`101`	3	$\overline{\texttt{001}}\texttt{01}$
6	`110`	3	$\overline{\texttt{001}}\texttt{10}$
$\vdots$			
45	`101101`	6	$\overline{\texttt{000001}}\texttt{01101}$

Table 7.1. C_α.

The codeword $c_\alpha(n)$ has length $l_\alpha(n) = 2l_{\rm b}(n) - 1$.

The implicit probability distribution over n for the code C_α is separable into the product of a probability distribution over the length l,

$$P(l) = 2^{-l}, \tag{7.2}$$

and a uniform distribution over integers having that length,

$$P(n \,|\, l) = \begin{cases} 2^{-l+1} & l_{\rm b}(n) = l \\ 0 & \text{otherwise.} \end{cases} \tag{7.3}$$

Now, for the above code, the header that communicates the length always occupies the same number of bits as the standard binary representation of the integer (give or take one). If we are expecting to encounter large integers (large files) then this representation seems suboptimal, since it leads to all files occupying a size that is double their original uncoded size. Instead of using the unary code to encode the length $l_{\rm b}(n)$, we could use C_α.

n	$c_\beta(n)$	$c_\gamma(n)$
1	$\overline{\texttt{1}}$	$\overline{\texttt{1}}$
2	$\overline{\texttt{010}}\texttt{0}$	$\overline{\texttt{0100}}\texttt{0}$
3	$\overline{\texttt{010}}\texttt{1}$	$\overline{\texttt{0100}}\texttt{1}$
4	$\overline{\texttt{011}}\texttt{00}$	$\overline{\texttt{0101}}\texttt{00}$
5	$\overline{\texttt{011}}\texttt{01}$	$\overline{\texttt{0101}}\texttt{01}$
6	$\overline{\texttt{011}}\texttt{10}$	$\overline{\texttt{0101}}\texttt{10}$
$\vdots$		
45	$\overline{\texttt{00110}}\texttt{01101}$	$\overline{\texttt{01110}}\texttt{01101}$

Table 7.2. C_β and C_γ.

Code C_β. We send the length $l_{\rm b}(n)$ using C_α, followed by the headless binary representation of n.

$$c_\beta(n) = c_\alpha[l_{\rm b}(n)]c_{\rm B}(n). \tag{7.4}$$

Iterating this procedure, we can define a sequence of codes.

Code C_γ.

$$c_\gamma(n) = c_\beta[l_{\rm b}(n)]c_{\rm B}(n). \tag{7.5}$$

Code C_δ.

$$c_\delta(n) = c_\gamma[l_{\rm b}(n)]c_{\rm B}(n). \tag{7.6}$$

Codes with end-of-file symbols

We can also make byte-based representations. (Let's use the term byte flexibly here, to denote any fixed-length string of bits, not just a string of length 8 bits.) If we encode the number in some base, for example decimal, then we can represent each digit in a byte. In order to represent a digit from 0 to 9 in a byte we need four bits. Because $2^4 = 16$, this leaves 6 extra four-bit symbols, {1010, 1011, 1100, 1101, 1110, 1111}, that correspond to no decimal digit. We can use these as end-of-file symbols to indicate the end of our positive integer.

n	$c_3(n)$	$c_7(n)$
1	01 11	001 111
2	10 11	010 111
3	01 00 11	011 111
⋮		
45	01 10 00 00 11	110 011 111

Table 7.3. Two codes with end-of-file symbols, C_3 and C_7. Spaces have been included to show the byte boundaries.

Clearly it is redundant to have more than one end-of-file symbol, so a more efficient code would encode the integer into base 15, and use just the sixteenth symbol, 1111, as the punctuation character. Generalizing this idea, we can make similar byte-based codes for integers in bases 3 and 7, and in any base of the form $2^n - 1$.

These codes are almost complete. (Recall that a code is 'complete' if it satisfies the Kraft inequality with equality.) The codes' remaining inefficiency is that they provide the ability to encode the integer zero and the empty string, neither of which was required.

▷ Exercise 7.1.[2, p.136] Consider the implicit probability distribution over integers corresponding to the code with an end-of-file character.

(a) If the code has eight-bit blocks (i.e., the integer is coded in base 255), what is the mean length in bits of the integer, under the implicit distribution?

(b) If one wishes to encode binary files of expected size about one hundred kilobytes using a code with an end-of-file character, what is the optimal block size?

Encoding a tiny file

To illustrate the codes we have discussed, we now use each code to encode a small file consisting of just 14 characters,

`Claude Shannon`.

- If we map the ASCII characters onto seven-bit symbols (e.g., in decimal, C = 67, l = 108, etc.), this 14 character file corresponds to the integer

$$n = 167\,987\,786\,364\,950\,891\,085\,602\,469\,870 \;\; \text{(decimal)}.$$

- The unary code for n consists of this many (less one) zeroes, followed by a one. If all the oceans were turned into ink, and if we wrote a hundred bits with every cubic millimeter, there might be enough ink to write $c_{\rm U}(n)$.

- The standard binary representation of n is this length-98 sequence of bits:

$$\begin{aligned} c_{\rm b}(n) = \;& \texttt{1000011110110011000011110101110010011001010100000} \\ & \texttt{1010011110100011000011101110110111011011111101110}. \end{aligned}$$

▷ Exercise 7.2.[2] Write down or describe the following self-delimiting representations of the above number n: $c_\alpha(n)$, $c_\beta(n)$, $c_\gamma(n)$, $c_\delta(n)$, $c_3(n)$, $c_7(n)$, and $c_{15}(n)$. Which of these encodings is the shortest? [Answer: c_{15}.]

Comparing the codes

One could answer the question 'which of two codes is superior?' by a sentence of the form 'For $n > k$, code 1 is superior, for $n < k$, code 2 is superior' but I contend that such an answer misses the point: any complete code corresponds to a prior for which it is optimal; you should not say that any other code is superior to it. Other codes are optimal for other priors. These implicit priors should be thought about so as to achieve the best code for one's application.

Notice that one cannot, for free, switch from one code to another, choosing whichever is shorter. If one were to do this, then it would be necessary to lengthen the message in some way that indicates which of the two codes is being used. If this is done by a single leading bit, it will be found that the resulting code is suboptimal because it fails the Kraft equality, as was discussed in exercise 5.33 (p.104).

Another way to compare codes for integers is to consider a sequence of probability distributions, such as monotonic probability distributions over $n \geq 1$, and rank the codes as to how well they encode *any* of these distributions. A code is called a 'universal' code if for any distribution in a given class, it encodes into an average length that is within some factor of the ideal average length.

Let me say this again. We are meeting an alternative world view – rather than figuring out a good prior over integers, as advocated above, many theorists have studied the problem of creating codes that are reasonably good codes for *any* priors in a broad class. Here the class of priors conventionally considered is the set of priors that (a) assign a monotonically decreasing probability over integers and (b) have finite entropy.

Several of the codes we have discussed above are universal. Another code which elegantly transcends the sequence of self-delimiting codes is Elias's 'universal code for integers' (Elias, 1975), which effectively chooses from all the codes $C_\alpha, C_\beta, \ldots$. It works by sending a sequence of messages each of which encodes the length of the next message, and indicates by a single bit whether or not that message is the final integer (in its standard binary representation). Because a length is a positive integer and all positive integers begin with '1', all the leading 1s can be omitted.

```
Write '0'
Loop {
    If $\lfloor \log n \rfloor = 0$ halt
    Prepend $c_{\rm b}(n)$ to the written string
    $n := \lfloor \log n \rfloor$
}
```

Algorithm 7.4. Elias's encoder for an integer n.

The encoder of C_ω is shown in algorithm 7.4. The encoding is generated from right to left. Table 7.5 shows the resulting codewords.

▷ Exercise 7.3.[2] Show that the Elias code is not actually the best code for a prior distribution that expects very large integers. (Do this by constructing another code and specifying how large n must be for your code to give a shorter length than Elias's.)

n	$c_\omega(n)$	n	$c_\omega(n)$	n	$c_\omega(n)$	n	$c_\omega(n)$
1	0	9	1110010	31	10100111110	256	1110001000000000
2	100	10	1110100	32	101011000000	365	1110001011011010
3	110	11	1110110	45	101011011010	511	1110001111111110
4	101000	12	1111000	63	101011111110	512	11100110000000000
5	101010	13	1111010	64	1011010000000	719	11100110110011110
6	101100	14	1111100	127	1011011111110	1023	11100111111111110
7	101110	15	1111110	128	10111100000000	1024	111010100000000000
8	1110000	16	10100100000	255	10111111111110	1025	111010100000000010

Table 7.5. Elias's 'universal' code for integers. Examples from 1 to 1025.

Solutions

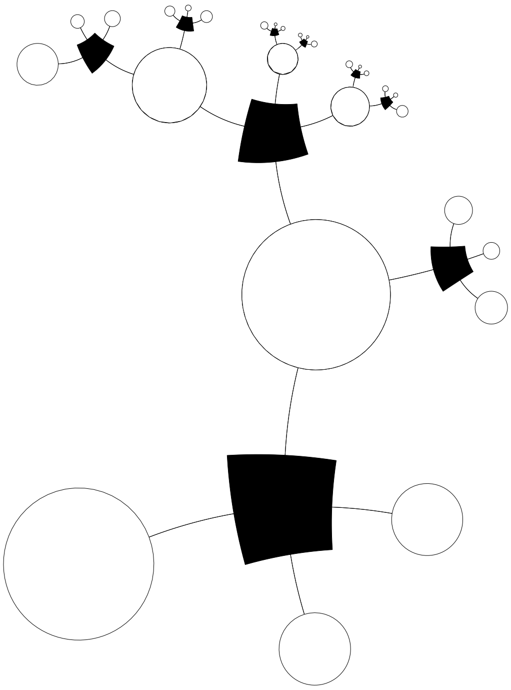
Solution to exercise 7.1 (p.134). The use of the end-of-file symbol in a code that represents the integer in some base q corresponds to a belief that there is a probability of $(1/(q+1))$ that the current character is the last character of the number. Thus the prior to which this code is matched puts an exponential prior distribution over the length of the integer.

(a) The expected number of characters is $q+1 = 256$, so the expected length of the integer is $256 \times 8 \simeq 2000$ bits.

(b) We wish to find q such that $q \log q \simeq 800\,000$ bits. A value of q between 2^{15} and 2^{16} satisfies this constraint, so 16-bit blocks are roughly the optimal size, assuming there is one end-of-file character.

Part II

Noisy-Channel Coding



8

Dependent Random Variables

In the last three chapters on data compression we concentrated on random vectors $\mathbf{x}$ coming from an extremely simple probability distribution, namely the separable distribution in which each component x_n is independent of the others.

In this chapter, we consider *joint ensembles* in which the random variables are dependent. This material has two motivations. First, data from the real world have interesting correlations, so to do data compression well, we need to know how to work with models that include dependences. Second, a noisy channel with input x and output y defines a joint ensemble in which x and y are dependent – if they were independent, it would be impossible to communicate over the channel – so communication over noisy channels (the topic of chapters 9–11) is described in terms of the entropy of joint ensembles.

▶ 8.1 More about entropy

This section gives definitions and exercises to do with entropy, carrying on from section 2.4.

The joint entropy of X, Y is:

$$H(X,Y) = \sum_{xy \in \mathcal{A}_X \mathcal{A}_Y} P(x,y) \log \frac{1}{P(x,y)}. \tag{8.1}$$

Entropy is additive for independent random variables:

$$H(X,Y) = H(X) + H(Y) \text{ iff } P(x,y) = P(x)P(y). \tag{8.2}$$

The conditional entropy of X **given** $y = b_k$ is the entropy of the probability distribution $P(x \,|\, y = b_k)$.

$$H(X \,|\, y = b_k) \equiv \sum_{x \in \mathcal{A}_X} P(x \,|\, y = b_k) \log \frac{1}{P(x \,|\, y = b_k)}. \tag{8.3}$$

The conditional entropy of X **given** Y is the average, over y, of the conditional entropy of X given y.

$$\begin{aligned} H(X \,|\, Y) &\equiv \sum_{y \in \mathcal{A}_Y} P(y) \left[\sum_{x \in \mathcal{A}_X} P(x \,|\, y) \log \frac{1}{P(x \,|\, y)} \right] \\ &= \sum_{xy \in \mathcal{A}_X \mathcal{A}_Y} P(x,y) \log \frac{1}{P(x \,|\, y)}. \end{aligned} \tag{8.4}$$

This measures the average uncertainty that remains about x when y is known.

The marginal entropy of X is another name for the entropy of X, $H(X)$, used to contrast it with the conditional entropies listed above.

Chain rule for information content. From the product rule for probabilities, equation (2.6), we obtain:

$$\log \frac{1}{P(x,y)} = \log \frac{1}{P(x)} + \log \frac{1}{P(y \,|\, x)} \tag{8.5}$$

so

$$h(x,y) = h(x) + h(y \,|\, x). \tag{8.6}$$

In words, this says that the information content of x and y is the information content of x plus the information content of y given x.

Chain rule for entropy. The joint entropy, conditional entropy and marginal entropy are related by:

$$H(X,Y) = H(X) + H(Y \,|\, X) = H(Y) + H(X \,|\, Y). \tag{8.7}$$

In words, this says that the uncertainty of X and Y is the uncertainty of X plus the uncertainty of Y given X.

The mutual information between X **and** Y is

$$I(X;Y) \equiv H(X) - H(X \,|\, Y), \tag{8.8}$$

and satisfies $I(X;Y) = I(Y;X)$, and $I(X;Y) \geq 0$. It measures the average reduction in uncertainty about x that results from learning the value of y; **or vice versa**, the average amount of information that x conveys about y.

The conditional mutual information between X **and** Y **given** $z\!=\!c_k$ is the mutual information between the random variables X and Y in the joint ensemble $P(x,y \,|\, z\!=\!c_k)$,

$$I(X;Y \,|\, z\!=\!c_k) = H(X \,|\, z\!=\!c_k) - H(X \,|\, Y, z\!=\!c_k). \tag{8.9}$$

The conditional mutual information between X **and** Y **given** Z is the average over z of the above conditional mutual information.

$$I(X;Y \,|\, Z) = H(X \,|\, Z) - H(X \,|\, Y, Z). \tag{8.10}$$

No other 'three-term entropies' will be defined. For example, expressions such as $I(X;Y;Z)$ and $I(X \,|\, Y;Z)$ are illegal. But you may put conjunctions of arbitrary numbers of variables in each of the three spots in the expression $I(X;Y \,|\, Z)$ – for example, $I(A,B;C,D \,|\, E,F)$ is fine: it measures how much information on average c and d convey about a and b, assuming e and f are known.

Figure 8.1 shows how the total entropy $H(X,Y)$ of a joint ensemble can be broken down. **This figure is important.** $\ast$

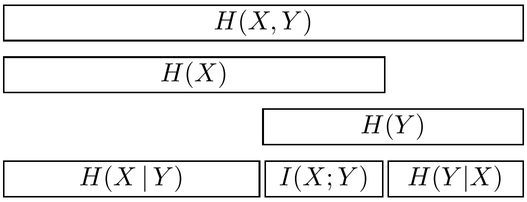


Figure 8.1. The relationship between joint information, marginal entropy, conditional entropy and mutual entropy.

8.2 Exercises

▷ Exercise 8.1.[1] Consider three independent random variables u, v, w with entropies H_u, H_v, H_w. Let $X \equiv (U, V)$ and $Y \equiv (V, W)$. What is $H(X, Y)$? What is $H(X \mid Y)$? What is $I(X;Y)$?

▷ Exercise 8.2.[3, p.142] Referring to the definitions of conditional entropy (8.3–8.4), confirm (with an example) that it is possible for $H(X \mid y = b_k)$ to exceed $H(X)$, but that the average, $H(X \mid Y)$, is less than $H(X)$. So data are helpful – they do not increase uncertainty, on average.

▷ Exercise 8.3.[2, p.143] Prove the chain rule for entropy, equation (8.7). $[H(X, Y) = H(X) + H(Y \mid X)]$.

Exercise 8.4.[2, p.143] Prove that the mutual information $I(X;Y) \equiv H(X) - H(X \mid Y)$ satisfies $I(X;Y) = I(Y;X)$ and $I(X;Y) \geq 0$.

[Hint: see exercise 2.26 (p.37) and note that

$$I(X;Y) = D_{\mathrm{KL}}(P(x,y)||P(x)P(y)).] \tag{8.11}$$

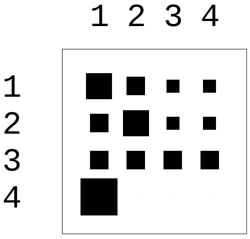
Exercise 8.5.[4] The 'entropy distance' between two random variables can be defined to be the difference between their joint entropy and their mutual information:

$$D_H(X,Y) \equiv H(X,Y) - I(X;Y). \tag{8.12}$$

Prove that the entropy distance satisfies the axioms for a distance – $D_H(X,Y) \geq 0$, $D_H(X,X) = 0$, $D_H(X,Y) = D_H(Y,X)$, and $D_H(X,Z) \leq D_H(X,Y) + D_H(Y,Z)$. [Incidentally, we are unlikely to see $D_H(X,Y)$ again but it is a good function on which to practise inequality-proving.]

Exercise 8.6.[2, p.147] A joint ensemble XY has the following joint distribution.

$P(x,y)$		x			
		1	2	3	4
	1	1/8	1/16	1/32	1/32
y	2	1/16	1/8	1/32	1/32
	3	1/16	1/16	1/16	1/16
	4	1/4	0	0	0



What is the joint entropy $H(X,Y)$? What are the marginal entropies $H(X)$ and $H(Y)$? For each value of y, what is the conditional entropy $H(X \mid y)$? What is the conditional entropy $H(X \mid Y)$? What is the conditional entropy of Y given X? What is the mutual information between X and Y?

Exercise 8.7.[*2*, p.143] Consider the ensemble XYZ in which $\mathcal{A}_X = \mathcal{A}_Y = \mathcal{A}_Z = \{0,1\}$, x and y are independent with $\mathcal{P}_X = \{p, 1-p\}$ and $\mathcal{P}_Y = \{q, 1-q\}$ and

$$z = (x+y) \bmod 2. \tag{8.13}$$

(a) If $q = 1/2$, what is $\mathcal{P}_Z$? What is $I(Z;X)$?

(b) For general p and q, what is $\mathcal{P}_Z$? What is $I(Z;X)$? Notice that this ensemble is related to the binary symmetric channel, with $x =$ input, $y =$ noise, and $z =$ output.

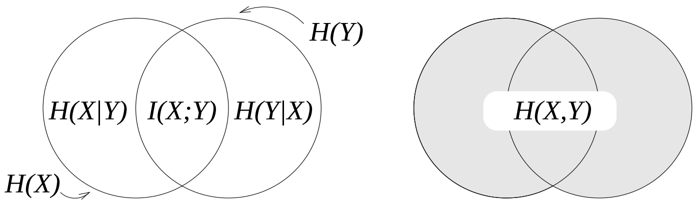


Figure 8.2. A misleading representation of entropies (contrast with figure 8.1).

Three term entropies

Exercise 8.8.[*3*, p.143] Many texts draw figure 8.1 in the form of a Venn diagram (figure 8.2). Discuss why this diagram is a misleading representation of entropies. Hint: consider the three-variable ensemble XYZ in which $x \in \{0,1\}$ and $y \in \{0,1\}$ are independent binary variables and $z \in \{0,1\}$ is defined to be $z = x + y \bmod 2$.

▶ 8.3 Further exercises

The data-processing theorem

The data processing theorem states that data processing can only destroy information.

Exercise 8.9.[*3*, p.144] Prove this theorem by considering an ensemble WDR in which w is the state of the world, d is data gathered, and r is the processed data, so that these three variables form a *Markov chain*

$$w \rightarrow d \rightarrow r, \tag{8.14}$$

that is, the probability $P(w,d,r)$ can be written as

$$P(w,d,r) = P(w)P(d \,|\, w)P(r \,|\, d). \tag{8.15}$$

Show that the average information that R conveys about W, $I(W;R)$, is less than or equal to the average information that D conveys about W, $I(W;D)$.

This theorem is as much a caution about our definition of 'information' as it is a caution about data processing!

Inference and information measures

Exercise 8.10.[2] The three cards.

(a) One card is white on both faces; one is black on both faces; and one is white on one side and black on the other. The three cards are shuffled and their orientations randomized. One card is drawn and placed on the table. The upper face is black. What is the colour of its lower face? (Solve the inference problem.)

(b) Does seeing the top face convey *information* about the colour of the bottom face? Discuss the *information contents* and *entropies* in this situation. Let the value of the upper face's colour be u and the value of the lower face's colour be l. Imagine that we draw a random card and learn both u and l. What is the entropy of u, $H(U)$? What is the entropy of l, $H(L)$? What is the mutual information between U and L, $I(U;L)$?

Entropies of Markov processes

▷ Exercise 8.11.[3] In the guessing game, we imagined predicting the next letter in a document starting from the beginning and working towards the end. Consider the task of predicting the *reversed* text, that is, predicting the letter that precedes those already known. Most people find this a harder task. Assuming that we model the language using an N-gram model (which says the probability of the next character depends only on the $N-1$ preceding characters), is there any difference between the average information contents of the reversed language and the forward language?

▶ 8.4 Solutions

Solution to exercise 8.2 (p.140). See exercise 8.6 (p.140) for an example where $H(X \mid y)$ exceeds $H(X)$ (set $y=3$).

We can prove the inequality $H(X \mid Y) \leq H(X)$ by turning the expression into a relative entropy (using Bayes' theorem) and invoking Gibbs' inequality (exercise 2.26 (p.37)):

$$\begin{aligned}
H(X \mid Y) &\equiv \sum_{y \in \mathcal{A}_Y} P(y) \left[\sum_{x \in \mathcal{A}_X} P(x \mid y) \log \frac{1}{P(x \mid y)} \right] \\
&= \sum_{xy \in \mathcal{A}_X \mathcal{A}_Y} P(x,y) \log \frac{1}{P(x \mid y)} && (8.16) \\
&= \sum_{xy} P(x) P(y \mid x) \log \frac{P(y)}{P(y \mid x) P(x)} && (8.17) \\
&= \sum_{x} P(x) \log \frac{1}{P(x)} + \sum_{x} P(x) \sum_{y} P(y \mid x) \log \frac{P(y)}{P(y \mid x)}. && (8.18)
\end{aligned}$$

The last expression is a sum of relative entropies between the distributions $P(y \mid x)$ and $P(y)$. So

$$H(X \mid Y) \leq H(X) + 0, \tag{8.19}$$

with equality only if $P(y \mid x) = P(y)$ for all x and y (that is, only if X and Y are independent).

Solution to exercise 8.3 (p.140). The chain rule for entropy follows from the decomposition of a joint probability:

$$\begin{aligned} H(X,Y) &= \sum_{xy} P(x,y) \log \frac{1}{P(x,y)} && (8.20) \\ &= \sum_{xy} P(x)P(y \,|\, x) \left[\log \frac{1}{P(x)} + \log \frac{1}{P(y \,|\, x)} \right] && (8.21) \\ &= \sum_{x} P(x) \log \frac{1}{P(x)} + \sum_{x} P(x) \sum_{y} P(y \,|\, x) \log \frac{1}{P(y \,|\, x)} && (8.22) \\ &= H(X) + H(Y \,|\, X). && (8.23) \end{aligned}$$

Solution to exercise 8.4 (p.140). Symmetry of mutual information:

$$\begin{aligned} I(X;Y) &= H(X) - H(X \,|\, Y) && (8.24) \\ &= \sum_{x} P(x) \log \frac{1}{P(x)} - \sum_{xy} P(x,y) \log \frac{1}{P(x \,|\, y)} && (8.25) \\ &= \sum_{xy} P(x,y) \log \frac{P(x \,|\, y)}{P(x)} && (8.26) \\ &= \sum_{xy} P(x,y) \log \frac{P(x,y)}{P(x)P(y)}. && (8.27) \end{aligned}$$

This expression is symmetric in x and y so

$$I(X;Y) = H(X) - H(X \,|\, Y) = H(Y) - H(Y \,|\, X). \qquad (8.28)$$

We can prove that mutual information is positive in two ways. One is to continue from

$$I(X;Y) = \sum_{x,y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)} \qquad (8.29)$$

which is a relative entropy and use Gibbs' inequality (proved on p.44), which asserts that this relative entropy is ≥ 0, with equality only if $P(x,y) = P(x)P(y)$, that is, if X and Y are independent.

The other is to use Jensen's inequality on

$$-\sum_{x,y} P(x,y) \log \frac{P(x)P(y)}{P(x,y)} \geq -\log \sum_{x,y} \frac{P(x,y)}{P(x,y)} P(x)P(y) = \log 1 = 0. \qquad (8.30)$$

Solution to exercise 8.7 (p.141). $z = x + y \bmod 2$.

(a) If $q = 1/2$, $\mathcal{P}_Z = \{1/2, 1/2\}$ and $I(Z;X) = H(Z) - H(Z \,|\, X) = 1 - 1 = 0$.

(b) For general q and p, $\mathcal{P}_Z = \{pq + (1-p)(1-q),\ p(1-q) + q(1-p)\}$. The mutual information is $I(Z;X) = H(Z) - H(Z \,|\, X) = H_2(pq + (1-p)(1-q)) - H_2(q)$.

Three term entropies

Solution to exercise 8.8 (p.141). The depiction of entropies in terms of Venn diagrams is misleading for at least two reasons.

First, one is used to thinking of Venn diagrams as depicting sets; but what are the 'sets' $H(X)$ and $H(Y)$ depicted in figure 8.2, and what are the objects that are members of those sets? I think this diagram encourages the novice student to make inappropriate analogies. For example, some students imagine

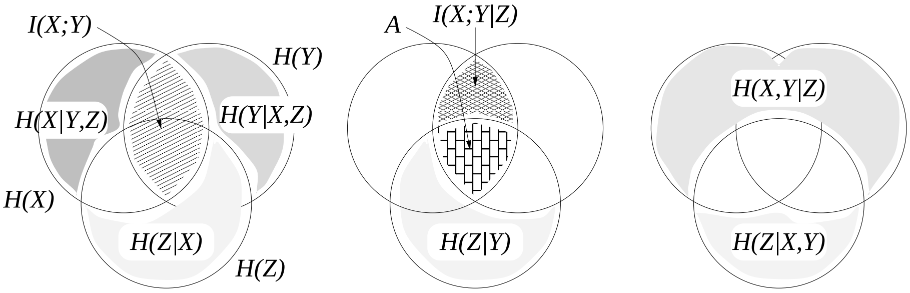


Figure 8.3. A misleading representation of entropies, continued.

that the random outcome (x, y) might correspond to a point in the diagram, and thus confuse entropies with probabilities.

Secondly, the depiction in terms of Venn diagrams encourages one to believe that all the areas correspond to positive quantities. In the special case of two random variables it is indeed true that $H(X \mid Y)$, $I(X;Y)$ and $H(Y \mid X)$ are positive quantities. But as soon as we progress to three-variable ensembles, we obtain a diagram with positive-looking areas that may actually correspond to negative quantities. Figure 8.3 correctly shows relationships such as

$$H(X) + H(Z \mid X) + H(Y \mid X, Z) = H(X, Y, Z). \tag{8.31}$$

But it gives the misleading impression that the conditional mutual information $I(X;Y \mid Z)$ is *less than* the mutual information $I(X;Y)$. In fact the area labelled A can correspond to a *negative* quantity. Consider the joint ensemble (X, Y, Z) in which $x \in \{0, 1\}$ and $y \in \{0, 1\}$ are independent binary variables and $z \in \{0, 1\}$ is defined to be $z = x + y \bmod 2$. Then clearly $H(X) = H(Y) = 1$ bit. Also $H(Z) = 1$ bit. And $H(Y \mid X) = H(Y) = 1$ since the two variables are independent. So the mutual information between X and Y is zero. $I(X;Y) = 0$. However, if z is observed, X and Y become dependent — knowing x, given z, tells you what y is: $y = z - x \bmod 2$. So $I(X;Y \mid Z) = 1$ bit. Thus the area labelled A must correspond to -1 bits for the figure to give the correct answers.

The above example is not at all a capricious or exceptional illustration. The binary symmetric channel with input X, noise Y, and output Z is a situation in which $I(X;Y) = 0$ (input and noise are independent) but $I(X;Y \mid Z) > 0$ (once you see the output, the unknown input and the unknown noise are intimately related!).

The Venn diagram representation is therefore valid only if one is aware that positive areas may represent negative quantities. With this proviso kept in mind, the interpretation of entropies in terms of sets can be helpful (Yeung, 1991).

Solution to exercise 8.9 (p.141). For any joint ensemble XYZ, the following chain rule for mutual information holds.

$$I(X;Y,Z) = I(X;Y) + I(X;Z \mid Y). \tag{8.32}$$

Now, in the case $w \rightarrow d \rightarrow r$, w and r are independent given d, so $I(W;R \mid D) = 0$. Using the chain rule twice, we have:

$$I(W;D,R) = I(W;D) \tag{8.33}$$

and

$$I(W;D,R) = I(W;R) + I(W;D \mid R), \tag{8.34}$$

so

$$I(W;R) - I(W;D) \leq 0. \tag{8.35}$$

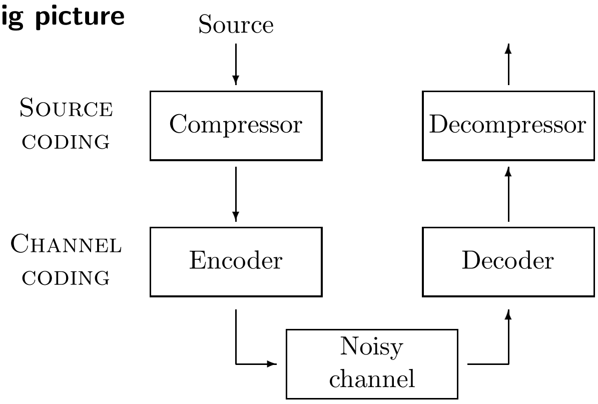
About Chapter 9

Before reading Chapter 9, you should have read Chapter 1 and worked on exercise 2.26 (p.37), and exercises 8.2–8.7 (pp.140–141).

9

Communication over a Noisy Channel

▶ 9.1 The big picture



In Chapters 4–6, we discussed source coding with block codes, symbol codes and stream codes. We implicitly assumed that the channel from the compressor to the decompressor was noise-free. Real channels are noisy. We will now spend two chapters on the subject of noisy-channel coding – the fundamental possibilities and limitations of error-free communication through a noisy channel. The aim of channel coding is to make the noisy channel behave like a noiseless channel. We will assume that the data to be transmitted has been through a good compressor, so the bit stream has no obvious redundancy. The channel code, which makes the transmission, will put back redundancy of a special sort, designed to make the noisy received signal decodeable.

Suppose we transmit 1000 bits per second with $p_0 = p_1 = 1/2$ over a noisy channel that flips bits with probability $f = 0.1$. What is the rate of transmission of information? We might guess that the rate is 900 bits per second by subtracting the expected number of errors per second. But this is not correct, because the recipient does not know where the errors occurred. Consider the case where the noise is so great that the received symbols are independent of the transmitted symbols. This corresponds to a noise level of $f = 0.5$, since half of the received symbols are correct due to chance alone. But when $f = 0.5$, no information is transmitted at all.

Given what we have learnt about entropy, it seems reasonable that a measure of the information transmitted is given by the mutual information between the source and the received signal, that is, the entropy of the source minus the conditional entropy of the source given the received signal.

We will now review the definition of conditional entropy and mutual information. Then we will examine whether it is possible to use such a noisy channel to communicate *reliably*. We will show that for any channel Q there is a non-zero rate, the capacity $C(Q)$, up to which information can be sent with arbitrarily small probability of error.

▶ 9.2 Review of probability and information

As an example, we take the joint distribution XY from exercise 8.6 (p.140). The marginal distributions $P(x)$ and $P(y)$ are shown in the margins.

$P(x,y)$		x				$P(y)$
		1	2	3	4	
	1	1/8	1/16	1/32	1/32	1/4
y	2	1/16	1/8	1/32	1/32	1/4
	3	1/16	1/16	1/16	1/16	1/4
	4	1/4	0	0	0	1/4
$P(x)$		1/2	1/4	1/8	1/8	

The joint entropy is $H(X,Y) = 27/8$ bits. The marginal entropies are $H(X) = 7/4$ bits and $H(Y) = 2$ bits.

We can compute the conditional distribution of x for each value of y, and the entropy of each of those conditional distributions:

$P(x \mid y)$		x				$H(X \mid y)$/bits
		1	2	3	4	
	1	1/2	1/4	1/8	1/8	7/4
y	2	1/4	1/2	1/8	1/8	7/4
	3	1/4	1/4	1/4	1/4	2
	4	1	0	0	0	0
						$H(X \mid Y) = 11/8$

Note that whereas $H(X \mid y=4) = 0$ is less than $H(X)$, $H(X \mid y=3)$ is greater than $H(X)$. So in some cases, learning y can *increase* our uncertainty about x. Note also that although $P(x \mid y=2)$ is a different distribution from $P(x)$, the conditional entropy $H(X \mid y=2)$ is equal to $H(X)$. So learning that y is 2 changes our knowledge about x but does not reduce the uncertainty of x, as measured by the entropy. On average though, learning y does convey information about x, since $H(X \mid Y) < H(X)$.

One may also evaluate $H(Y|X) = 13/8$ bits. The mutual information is $I(X;Y) = H(X) - H(X \mid Y) = 3/8$ bits.

▶ 9.3 Noisy channels

A discrete memoryless channel Q is characterized by an input alphabet $\mathcal{A}_X$, an output alphabet $\mathcal{A}_Y$, and a set of conditional probability distributions $P(y \mid x)$, one for each $x \in \mathcal{A}_X$.

These *transition probabilities* may be written in a matrix

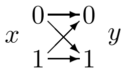
$$Q_{j|i} = P(y=b_j \mid x=a_i). \tag{9.1}$$

I usually orient this matrix with the output variable j indexing the rows and the input variable i indexing the columns, so that each column of $\mathbf{Q}$ is a probability vector. With this convention, we can obtain the probability of the output, $\mathbf{p}_Y$, from a probability distribution over the input, $\mathbf{p}_X$, by right-multiplication:

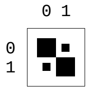
$$\mathbf{p}_Y = \mathbf{Q}\mathbf{p}_X. \tag{9.2}$$

Some useful model channels are:

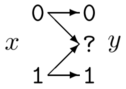
Binary symmetric channel. $\mathcal{A}_X = \{\texttt{0}, \texttt{1}\}$. $\mathcal{A}_Y = \{\texttt{0}, \texttt{1}\}$.



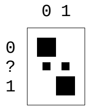
$$\begin{array}{rclrcl} P(y=\texttt{0}\,|\,x=\texttt{0}) &=& 1-f; & P(y=\texttt{0}\,|\,x=\texttt{1}) &=& f; \\ P(y=\texttt{1}\,|\,x=\texttt{0}) &=& f; & P(y=\texttt{1}\,|\,x=\texttt{1}) &=& 1-f. \end{array}$$



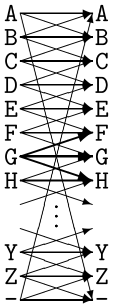
Binary erasure channel. $\mathcal{A}_X = \{\texttt{0}, \texttt{1}\}$. $\mathcal{A}_Y = \{\texttt{0}, \texttt{?}, \texttt{1}\}$.



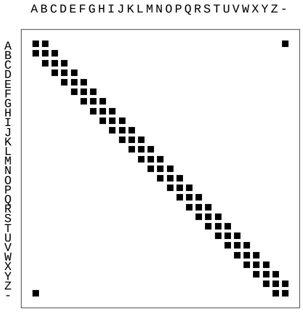
$$\begin{array}{rclrcl} P(y=\texttt{0}\,|\,x=\texttt{0}) &=& 1-f; & P(y=\texttt{0}\,|\,x=\texttt{1}) &=& 0; \\ P(y=\texttt{?}\,|\,x=\texttt{0}) &=& f; & P(y=\texttt{?}\,|\,x=\texttt{1}) &=& f; \\ P(y=\texttt{1}\,|\,x=\texttt{0}) &=& 0; & P(y=\texttt{1}\,|\,x=\texttt{1}) &=& 1-f. \end{array}$$



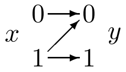
Noisy typewriter. $\mathcal{A}_X = \mathcal{A}_Y =$ the 27 letters {A, B, . . . , Z, -}. The letters are arranged in a circle, and when the typist attempts to type B, what comes out is either A, B or C, with probability 1/3 each; when the input is C, the output is B, C or D; and so forth, with the final letter '-' adjacent to the first letter A.



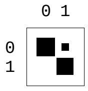
$$\begin{array}{rcl} & \vdots & \\ P(y=\texttt{F}\,|\,x=\texttt{G}) &=& 1/3; \\ P(y=\texttt{G}\,|\,x=\texttt{G}) &=& 1/3; \\ P(y=\texttt{H}\,|\,x=\texttt{G}) &=& 1/3; \\ & \vdots & \end{array}$$



Z channel. $\mathcal{A}_X = \{\texttt{0}, \texttt{1}\}$. $\mathcal{A}_Y = \{\texttt{0}, \texttt{1}\}$.



$$\begin{array}{rclrcl} P(y=\texttt{0}\,|\,x=\texttt{0}) &=& 1; & P(y=\texttt{0}\,|\,x=\texttt{1}) &=& f; \\ P(y=\texttt{1}\,|\,x=\texttt{0}) &=& 0; & P(y=\texttt{1}\,|\,x=\texttt{1}) &=& 1-f. \end{array}$$



▶ 9.4 Inferring the input given the output

If we assume that the input x to a channel comes from an ensemble X, then we obtain a joint ensemble XY in which the random variables x and y have the joint distribution:

$$P(x, y) = P(y\,|\,x)P(x). \tag{9.3}$$

Now if we receive a particular symbol y, what was the input symbol x? We typically won't know for certain. We can write down the posterior distribution of the input using Bayes' theorem:

$$P(x\,|\,y) = \frac{P(y\,|\,x)P(x)}{P(y)} = \frac{P(y\,|\,x)P(x)}{\sum_{x'} P(y\,|\,x')P(x')}. \tag{9.4}$$

Example 9.1. Consider a binary symmetric channel with probability of error $f = 0.15$. Let the input ensemble be $\mathcal{P}_X : \{p_0 = 0.9, p_1 = 0.1\}$. Assume we observe $y = 1$.

$$\begin{aligned} P(x=1\,|\,y=1) &= \frac{P(y=1\,|\,x=1)P(x=1)}{\sum_{x'} P(y\,|\,x')P(x')} \\ &= \frac{0.85 \times 0.1}{0.85 \times 0.1 + 0.15 \times 0.9} \\ &= \frac{0.085}{0.22} \;=\; 0.39. \end{aligned} \tag{9.5}$$

Thus '$x = 1$' is still less probable than '$x = 0$', although it is not as improbable as it was before.

Exercise 9.2.$^{[\mathit{1},\ \text{p.157}]}$ Now assume we observe $y = 0$. Compute the probability of $x = 1$ given $y = 0$.

Example 9.3. Consider a Z channel with probability of error $f = 0.15$. Let the input ensemble be $\mathcal{P}_X : \{p_0 = 0.9, p_1 = 0.1\}$. Assume we observe $y = 1$.

$$\begin{aligned} P(x=1 \,|\, y=1) &= \frac{0.85 \times 0.1}{0.85 \times 0.1 + 0 \times 0.9} \\ &= \frac{0.085}{0.085} \;=\; 1.0. \end{aligned} \tag{9.6}$$

So given the output $y = 1$ we become certain of the input.

Exercise 9.4.$^{[\mathit{1},\ \text{p.157}]}$ Alternatively, assume we observe $y = 0$. Compute $P(x = 1 \,|\, y = 0)$.

▶ 9.5 Information conveyed by a channel

We now consider how much information can be communicated through a channel. In operational terms, we are interested in finding ways of using the channel such that all the bits that are communicated are recovered with negligible probability of error. In mathematical terms, assuming a particular input ensemble X, we can measure how much information the output conveys about the input by the mutual information:

$$I(X;Y) \equiv H(X) - H(X \,|\, Y) = H(Y) - H(Y|X). \tag{9.7}$$

Our aim is to establish the connection between these two ideas. Let us evaluate $I(X;Y)$ for some of the channels above.

Hint for computing mutual information

We will tend to think of $I(X;Y)$ as $H(X) - H(X \,|\, Y)$, i.e., how much the uncertainty of the input X is reduced when we look at the output Y. But for computational purposes it is often handy to evaluate $H(Y) - H(Y|X)$ instead.

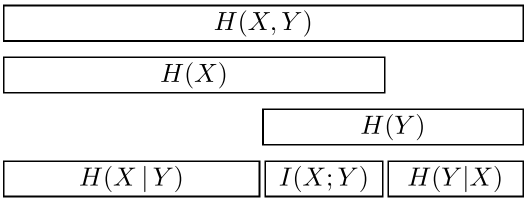


Figure 9.1. The relationship between joint information, marginal entropy, conditional entropy and mutual entropy. This figure is important, so I'm showing it twice.

Example 9.5. Consider the binary symmetric channel again, with $f = 0.15$ and $\mathcal{P}_X : \{p_0 = 0.9, p_1 = 0.1\}$. We already evaluated the marginal probabilities $P(y)$ implicitly above: $P(y = 0) = 0.78$; $P(y = 1) = 0.22$. The mutual information is:

$$I(X;Y) \;=\; H(Y) - H(Y|X).$$

What is $H(Y|X)$? It is defined to be the weighted sum over x of $H(Y \mid x)$; but $H(Y \mid x)$ is the same for each value of x: $H(Y \mid x = \mathtt{0})$ is $H_2(0.15)$, and $H(Y \mid x = \mathtt{1})$ is $H_2(0.15)$. So

$$\begin{aligned} I(X;Y) &= H(Y) - H(Y|X) \\ &= H_2(0.22) - H_2(0.15) \\ &= 0.76 - 0.61 \;=\; 0.15 \text{ bits.} \end{aligned} \tag{9.8}$$

This may be contrasted with the entropy of the source $H(X) = H_2(0.1) = 0.47$ bits.

Note: here we have used the binary entropy function $H_2(p) \equiv H(p, 1-p) = p \log \frac{1}{p} + (1-p) \log \frac{1}{(1-p)}$.

Throughout this book, log means $\log_2$.

Example 9.6. And now the Z channel, with $\mathcal{P}_X$ as above. $P(y=1) = 0.085$.

$$\begin{aligned} I(X;Y) &= H(Y) - H(Y|X) \\ &= H_2(0.085) - [0.9H_2(0) + 0.1H_2(0.15)] \\ &= 0.42 - (0.1 \times 0.61) = 0.36 \text{ bits.} \end{aligned} \tag{9.9}$$

The entropy of the source, as above, is $H(X) = 0.47$ bits. Notice that the mutual information $I(X;Y)$ for the Z channel is bigger than the mutual information for the binary symmetric channel with the same f. The Z channel is a more reliable channel.

Exercise 9.7.[*1*, p.157] Compute the mutual information between X and Y for the binary symmetric channel with $f = 0.15$ when the input distribution is $\mathcal{P}_X = \{p_0 = 0.5, p_1 = 0.5\}$.

Exercise 9.8.[*2*, p.157] Compute the mutual information between X and Y for the Z channel with $f = 0.15$ when the input distribution is $\mathcal{P}_X : \{p_0 = 0.5, p_1 = 0.5\}$.

Maximizing the mutual information

We have observed in the above examples that the mutual information between the input and the output depends on the chosen input ensemble.

Let us assume that we wish to maximize the mutual information conveyed by the channel by choosing the best possible input ensemble. We define the *capacity* of the channel to be its maximum mutual information.

The capacity of a channel Q is:

$$C(Q) = \max_{\mathcal{P}_X} I(X;Y). \tag{9.10}$$

The distribution $\mathcal{P}_X$ that achieves the maximum is called the *optimal input distribution*, denoted by $\mathcal{P}_X^*$. [There may be multiple optimal input distributions achieving the same value of $I(X;Y)$.]

In Chapter 10 we will show that the capacity does indeed measure the maximum amount of error-free information that can be transmitted over the channel per unit time.

Example 9.9. Consider the binary symmetric channel with $f = 0.15$. Above, we considered $\mathcal{P}_X = \{p_0 = 0.9, p_1 = 0.1\}$, and found $I(X;Y) = 0.15$ bits.

How much better can we do? By symmetry, the optimal input distribution is $\{0.5, 0.5\}$ and the capacity is

$$C(Q_{\text{BSC}}) \;=\; H_2(0.5) - H_2(0.15) \;=\; 1.0 - 0.61 \;=\; 0.39\,\text{bits}. \tag{9.11}$$

We'll justify the symmetry argument later. If there's any doubt about the symmetry argument, we can always resort to explicit maximization of the mutual information $I(X;Y)$,

$$I(X;Y) = H_2((1{-}f)p_1 + (1{-}p_1)f) - H_2(f) \quad \text{(figure 9.2)}. \tag{9.12}$$

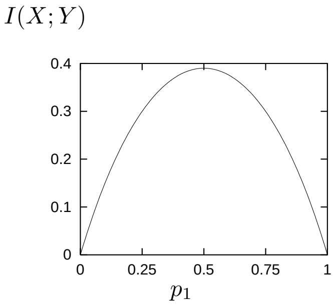


Figure 9.2. The mutual information $I(X;Y)$ for a binary symmetric channel with $f = 0.15$ as a function of the input distribution.

Example 9.10. The noisy typewriter. The optimal input distribution is a uniform distribution over x, and gives $C = \log_2 9$ bits.

Example 9.11. Consider the Z channel with $f = 0.15$. Identifying the optimal input distribution is not so straightforward. We evaluate $I(X;Y)$ explicitly for $\mathcal{P}_X = \{p_0, p_1\}$. First, we need to compute $P(y)$. The probability of $y = 1$ is easiest to write down:

$$P(y{=}1) \;=\; p_1(1-f). \tag{9.13}$$

Then the mutual information is:

$$\begin{aligned} I(X;Y) &= H(Y) - H(Y|X) \\ &= H_2(p_1(1-f)) - (p_0 H_2(0) + p_1 H_2(f)) \\ &= H_2(p_1(1-f)) - p_1 H_2(f). \end{aligned} \tag{9.14}$$

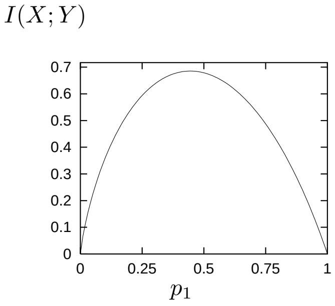


Figure 9.3. The mutual information $I(X;Y)$ for a Z channel with $f = 0.15$ as a function of the input distribution.

This is a non-trivial function of p_1, shown in figure 9.3. It is maximized for $f = 0.15$ by $p_1^* = 0.445$. We find $C(Q_Z) = 0.685$. Notice the optimal input distribution is not $\{0.5, 0.5\}$. We can communicate slightly more information by using input symbol 0 more frequently than 1.

Exercise 9.12.[1, p.158] What is the capacity of the binary symmetric channel for general f?

Exercise 9.13.[2, p.158] Show that the capacity of the binary erasure channel with $f = 0.15$ is $C_{\text{BEC}} = 0.85$. What is its capacity for general f? Comment.

▶ 9.6 The noisy-channel coding theorem

It seems plausible that the 'capacity' we have defined may be a measure of information conveyed by a channel; what is not obvious, and what we will prove in the next chapter, is that the capacity indeed measures the rate at which blocks of data can be communicated over the channel *with arbitrarily small probability of error.*

We make the following definitions.

An (N, K) **block code** for a channel Q is a list of $S = 2^K$ codewords

$$\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(2^K)}\}, \quad \mathbf{x}^{(s)} \in \mathcal{A}_X^N,$$

each of length N. Using this code we can encode a signal $s \in \{1, 2, 3, \ldots, 2^K\}$ as $\mathbf{x}^{(s)}$. [The number of codewords S is an integer, but the number of bits specified by choosing a codeword, $K \equiv \log_2 S$, is not necessarily an integer.]

The *rate* of the code is $R = K/N$ bits per channel use.

[We will use this definition of the rate for any channel, not only channels with binary inputs; note however that it is sometimes conventional to define the rate of a code for a channel with q input symbols to be $K/(N \log q)$.]

A decoder for an (N, K) block code is a mapping from the set of length-N strings of channel outputs, $\mathcal{A}_Y^N$, to a codeword label $\hat{s} \in \{0, 1, 2, \ldots, 2^K\}$.

The extra symbol $\hat{s} = 0$ can be used to indicate a 'failure'.

The probability of block error of a code and decoder, for a given channel, and for a given probability distribution over the encoded signal $P(s_{\rm in})$, is:

$$p_{\rm B} = \sum_{s_{\rm in}} P(s_{\rm in}) P(s_{\rm out} \neq s_{\rm in} \,|\, s_{\rm in}). \tag{9.15}$$

The maximal probability of block error is

$$p_{\rm BM} = \max_{s_{\rm in}} P(s_{\rm out} \neq s_{\rm in} \,|\, s_{\rm in}). \tag{9.16}$$

The optimal decoder for a channel code is the one that minimizes the probability of block error. It decodes an output $\mathbf{y}$ as the input s that has maximum posterior probability $P(s \,|\, \mathbf{y})$.

$$P(s \,|\, \mathbf{y}) = \frac{P(\mathbf{y} \,|\, s) P(s)}{\sum_{s'} P(\mathbf{y} \,|\, s') P(s')} \tag{9.17}$$

$$\hat{s}_{\rm optimal} = \operatorname{argmax} P(s \,|\, \mathbf{y}). \tag{9.18}$$

A uniform prior distribution on s is usually assumed, in which case the optimal decoder is also the *maximum likelihood decoder*, i.e., the decoder that maps an output $\mathbf{y}$ to the input s that has maximum *likelihood* $P(\mathbf{y} \,|\, s)$.

The probability of bit error $p_{\rm b}$ is defined assuming that the codeword number s is represented by a binary vector $\mathbf{s}$ of length K bits; it is the average probability that a bit of $\mathbf{s}_{\rm out}$ is not equal to the corresponding bit of $\mathbf{s}_{\rm in}$ (averaging over all K bits).

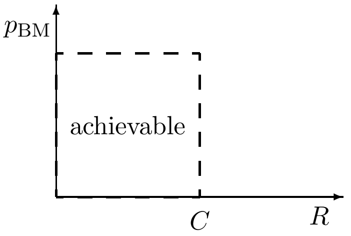


Figure 9.4. Portion of the $R, p_{\rm BM}$ plane asserted to be achievable by the first part of Shannon's noisy channel coding theorem.

Shannon's noisy-channel coding theorem (part one). Associated with each discrete memoryless channel, there is a non-negative number C (called the channel capacity) with the following property. For any $\epsilon > 0$ and $R < C$, for large enough N, there exists a block code of length N and rate $\geq R$ and a decoding algorithm, such that the maximal probability of block error is $< \epsilon$.

Confirmation of the theorem for the noisy typewriter channel

In the case of the noisy typewriter, we can easily confirm the theorem, because we can create a completely error-free communication strategy using a block code of length $N = 1$: we use only the letters `B`, `E`, `H`, ..., `Z`, i.e., every third letter. These letters form a *non-confusable subset* of the input alphabet (see figure 9.5). Any output can be uniquely decoded. The number of inputs in the non-confusable subset is 9, so the error-free information rate of this system is $\log_2 9$ bits, which is equal to the capacity C, which we evaluated in example 9.10 (p.151).

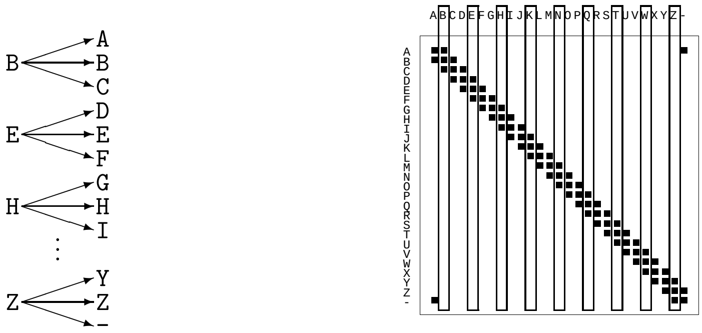


Figure 9.5. A non-confusable subset of inputs for the noisy typewriter.

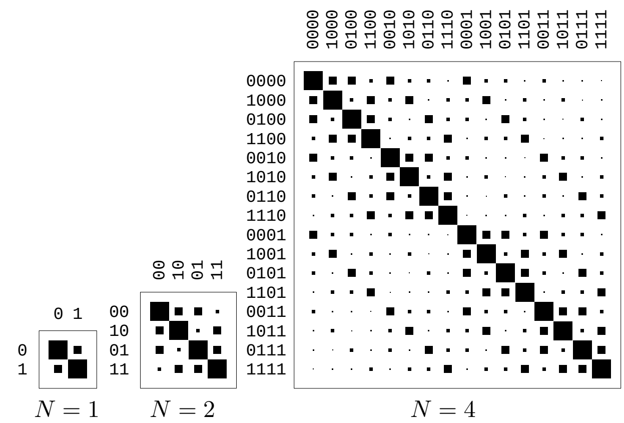


Figure 9.6. Extended channels obtained from a binary symmetric channel with transition probability 0.15.

How does this translate into the terms of the theorem? The following table explains.

The theorem	How it applies to the noisy typewriter
Associated with each discrete memoryless channel, there is a non-negative number C.	The capacity C is $\log_2 9$.
For any $\epsilon > 0$ and $R < C$, for large enough N,	No matter what ϵ and R are, we set the blocklength N to 1.
there exists a block code of length N and rate $\geq R$	The block code is $\{\mathtt{B}, \mathtt{E}, \ldots, \mathtt{Z}\}$. The value of K is given by $2^K = 9$, so $K = \log_2 9$, and this code has rate $\log_2 9$, which is greater than the requested value of R.
and a decoding algorithm,	The decoding algorithm maps the received letter to the nearest letter in the code;
such that the maximal probability of block error is $< \epsilon$.	the maximal probability of block error is zero, which is less than the given ϵ.

▶ 9.7 Intuitive preview of proof

Extended channels

To prove the theorem for any given channel, we consider the *extended channel* corresponding to N uses of the channel. The extended channel has $|\mathcal{A}_X|^N$ possible inputs $\mathbf{x}$ and $|\mathcal{A}_Y|^N$ possible outputs. Extended channels obtained from a binary symmetric channel and from a Z channel are shown in figures 9.6 and 9.7, with $N = 2$ and $N = 4$.

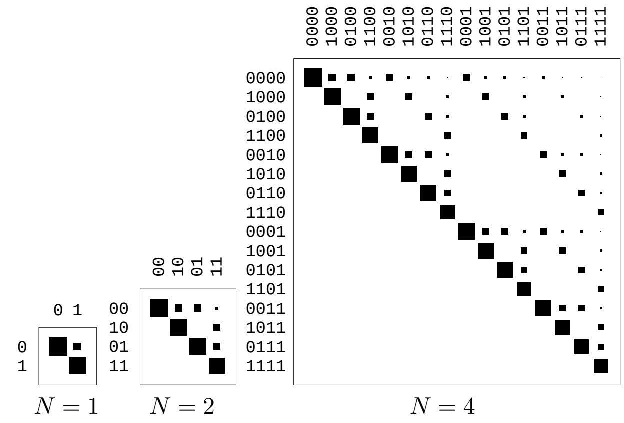


Figure 9.7. Extended channels obtained from a Z channel with transition probability 0.15. Each column corresponds to an input, and each row is a different output.

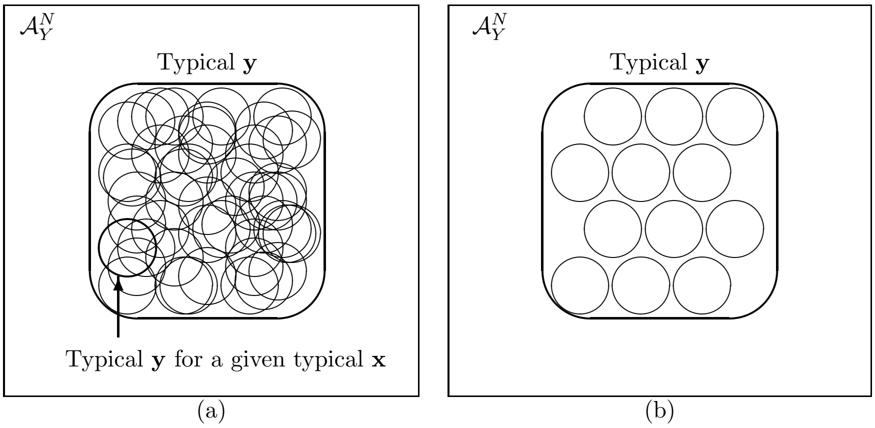


Figure 9.8. (a) Some typical outputs in $\mathcal{A}_Y^N$ corresponding to typical inputs $\mathbf{x}$. (b) A subset of the typical sets shown in (a) that do not overlap each other. This picture can be compared with the solution to the noisy typewriter in figure 9.5.

Exercise 9.14.[2, p.159] Find the transition probability matrices $\mathbf{Q}$ for the extended channel, with $N = 2$, derived from the binary erasure channel having erasure probability 0.15.

By selecting two columns of this transition probability matrix, we can define a rate-$1/2$ code for this channel with blocklength $N = 2$. What is the best choice of two columns? What is the decoding algorithm?

To prove the noisy-channel coding theorem, we make use of large blocklengths N. The intuitive idea is that, if N is large, *an extended channel looks a lot like the noisy typewriter.* Any particular input $\mathbf{x}$ is very likely to produce an output in a small subspace of the output alphabet – the typical output set, given that input. So we can find a non-confusable subset of the inputs that produce essentially disjoint output sequences. For a given N, let us consider a way of generating such a non-confusable subset of the inputs, and count up how many distinct inputs it contains.

Imagine making an input sequence $\mathbf{x}$ for the extended channel by drawing it from an ensemble X^N, where X is an arbitrary ensemble over the input alphabet. Recall the source coding theorem of Chapter 4, and consider the number of probable output sequences $\mathbf{y}$. The total number of typical output sequences $\mathbf{y}$ is $2^{NH(Y)}$, all having similar probability. For any particular typical input sequence $\mathbf{x}$, there are about $2^{NH(Y|X)}$ probable sequences. Some of these subsets of $\mathcal{A}_Y^N$ are depicted by circles in figure 9.8a.

We now imagine restricting ourselves to a subset of the typical inputs $\mathbf{x}$ such that the corresponding typical output sets do not overlap, as shown in figure 9.8b. We can then bound the number of non-confusable inputs by dividing the size of the typical $\mathbf{y}$ set, $2^{NH(Y)}$, by the size of each typical-$\mathbf{y}$-

given-typical-**x** set, $2^{NH(Y|X)}$. So the number of non-confusable inputs, if they are selected from the set of typical inputs $\mathbf{x} \sim X^N$, is $\leq 2^{NH(Y)-NH(Y|X)} = 2^{NI(X;Y)}$.

The maximum value of this bound is achieved if X is the ensemble that maximizes $I(X;Y)$, in which case the number of non-confusable inputs is $\leq 2^{NC}$. Thus asymptotically up to C bits per cycle, and no more, can be communicated with vanishing error probability. □

This sketch has not rigorously proved that reliable communication really is possible – that's our task for the next chapter.

▶ 9.8 Further exercises

Exercise 9.15.[3, p.159] Refer back to the computation of the capacity of the Z channel with $f = 0.15$.

(a) Why is p_1^* less than 0.5? One could argue that it is good to favour the `0` input, since it is transmitted without error – and also argue that it is good to favour the `1` input, since it often gives rise to the highly prized `1` output, which allows certain identification of the input! Try to make a convincing argument.

(b) In the case of general f, show that the optimal input distribution is

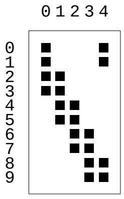
$$p_1^* = \frac{1/(1-f)}{1 + 2^{(H_2(f)/(1-f))}}. \tag{9.19}$$

(c) What happens to p_1^* if the noise level f is very close to 1?

Exercise 9.16.[2, p.159] Sketch graphs of the capacity of the Z channel, the binary symmetric channel and the binary erasure channel as a function of f.

▷ Exercise 9.17.[2] What is the capacity of the five-input, ten-output channel whose transition probability matrix is

$$\begin{bmatrix} 0.25 & 0 & 0 & 0 & 0.25 \\ 0.25 & 0 & 0 & 0 & 0.25 \\ 0.25 & 0.25 & 0 & 0 & 0 \\ 0.25 & 0.25 & 0 & 0 & 0 \\ 0 & 0.25 & 0.25 & 0 & 0 \\ 0 & 0.25 & 0.25 & 0 & 0 \\ 0 & 0 & 0.25 & 0.25 & 0 \\ 0 & 0 & 0.25 & 0.25 & 0 \\ 0 & 0 & 0 & 0.25 & 0.25 \\ 0 & 0 & 0 & 0.25 & 0.25 \end{bmatrix} \quad ? \tag{9.20}$$



Exercise 9.18.[2, p.159] Consider a Gaussian channel with binary input $x \in \{-1, +1\}$ and *real* output alphabet $\mathcal{A}_Y$, with transition probability density

$$Q(y \,|\, x, \alpha, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \, e^{-\frac{(y - x\alpha)^2}{2\sigma^2}}, \tag{9.21}$$

where α is the signal amplitude.

(a) Compute the posterior probability of x given y, assuming that the two inputs are equiprobable. Put your answer in the form

$$P(x\!=\!1 \,|\, y, \alpha, \sigma) = \frac{1}{1 + e^{-a(y)}}. \tag{9.22}$$

Sketch the value of $P(x=1 \mid y, \alpha, \sigma)$ as a function of y.

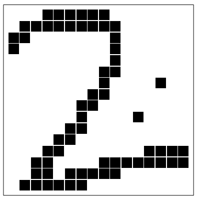
(b) Assume that a single bit is to be transmitted. What is the optimal decoder, and what is its probability of error? Express your answer in terms of the signal-to-noise ratio α^2/σ^2 and the error function (the cumulative probability function of the Gaussian distribution),

$$\Phi(z) \equiv \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}} \, \mathrm{d}z. \tag{9.23}$$

[Note that this definition of the error function $\Phi(z)$ may not correspond to other people's.]

Pattern recognition as a noisy channel

We may think of many pattern recognition problems in terms of communication channels. Consider the case of recognizing handwritten digits (such as postcodes on envelopes). The author of the digit wishes to communicate a message from the set $\mathcal{A}_X = \{0, 1, 2, 3, \ldots, 9\}$; this selected message is the input to the channel. What comes out of the channel is a pattern of ink on paper. If the ink pattern is represented using 256 binary pixels, the channel Q has as its output a random variable $y \in \mathcal{A}_Y = \{0, 1\}^{256}$. An example of an element from this alphabet is shown in the margin.



Exercise 9.19.[2] Estimate how many patterns in $\mathcal{A}_Y$ are recognizable as the character '2'. [The aim of this problem is to try to demonstrate the existence of *as many patterns as possible* that are recognizable as 2s.]

Discuss how one might model the channel $P(y \mid x=2)$. Estimate the entropy of the probability distribution $P(y \mid x=2)$.

One strategy for doing pattern recognition is to create a model for $P(y \mid x)$ for each value of the input $x = \{0, 1, 2, 3, \ldots, 9\}$, then use Bayes' theorem to infer x given y.

$$P(x \mid y) = \frac{P(y \mid x) P(x)}{\sum_{x'} P(y \mid x') P(x')}. \tag{9.24}$$

This strategy is known as *full probabilistic modelling* or *generative modelling*. This is essentially how current speech recognition systems work. In addition to the channel model, $P(y \mid x)$, one uses a prior probability distribution $P(x)$, which in the case of both character recognition and speech recognition is a language model that specifies the probability of the next character/word given the context and the known grammar and statistics of the language.

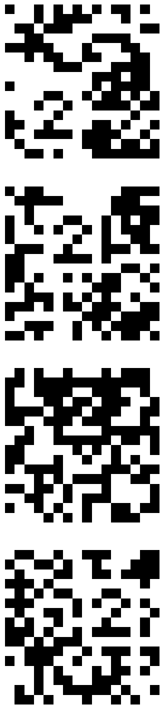


Figure 9.9. Some more 2s.

Random coding

Exercise 9.20.[2, p.160] Given twenty-four people in a room, what is the probability that there are at least two people present who have the same birthday (i.e., day and month of birth)? What is the expected number of pairs of people with the same birthday? Which of these two questions is easiest to solve? Which answer gives most insight? You may find it helpful to solve these problems and those that follow using notation such as A = number of days in year = 365 and S = number of people = 24.

▷ Exercise 9.21.[2] The birthday problem may be related to a coding scheme. Assume we wish to convey a message to an outsider identifying one of the twenty-four people. We could simply communicate a number s from $\mathcal{A}_S = \{1, 2, \ldots, 24\}$, having agreed a mapping of people onto numbers; alternatively, we could convey a number from $\mathcal{A}_X = \{1, 2, \ldots, 365\}$, identifying the day of the year that is the selected person's birthday (with apologies to leapyearians). [The receiver is assumed to know all the people's birthdays.] What, roughly, is the probability of error of this communication scheme, assuming it is used for a single transmission? What is the capacity of the communication channel, and what is the rate of communication attempted by this scheme?

▷ Exercise 9.22.$^{[2]}$ Now imagine that there are K rooms in a building, each containing q people. (You might think of $K = 2$ and $q = 24$ as an example.) The aim is to communicate a selection of one person from each room by transmitting an ordered list of K days (from $\mathcal{A}_X$). Compare the probability of error of the following two schemes.

(a) As before, where each room transmits the birthday of the selected person.

(b) To each K-tuple of people, one drawn from each room, an ordered K-tuple of randomly selected days from $\mathcal{A}_X$ is assigned (this K-tuple has nothing to do with their birthdays). This enormous list of $S = q^K$ strings is known to the receiver. When the building has selected a particular person from each room, the ordered string of days corresponding to that K-tuple of people is transmitted.

What is the probability of error when $q = 364$ and $K = 1$? What is the probability of error when $q = 364$ and K is large, e.g. $K = 6000$?

▶ 9.9 Solutions

Solution to exercise 9.2 (p.149). If we assume we observe $y = 0$,

$$P(x=1 \,|\, y=0) = \frac{P(y=0 \,|\, x=1)P(x=1)}{\sum_{x'} P(y \,|\, x')P(x')} \tag{9.25}$$

$$= \frac{0.15 \times 0.1}{0.15 \times 0.1 + 0.85 \times 0.9} \tag{9.26}$$

$$= \frac{0.015}{0.78} = 0.019. \tag{9.27}$$

Solution to exercise 9.4 (p.149). If we observe $y = 0$,

$$P(x=1 \,|\, y=0) = \frac{0.15 \times 0.1}{0.15 \times 0.1 + 1.0 \times 0.9} \tag{9.28}$$

$$= \frac{0.015}{0.915} = 0.016. \tag{9.29}$$

Solution to exercise 9.7 (p.150). The probability that $y = 1$ is 0.5, so the mutual information is:

$$I(X;Y) = H(Y) - H(Y \,|\, X) \tag{9.30}$$

$$= H_2(0.5) - H_2(0.15) \tag{9.31}$$

$$= 1 - 0.61 = 0.39 \text{ bits.} \tag{9.32}$$

Solution to exercise 9.8 (p.150). We again compute the mutual information using $I(X;Y) = H(Y) - H(Y \,|\, X)$. The probability that $y = 0$ is 0.575, and

$H(Y \mid X) = \sum_x P(x) H(Y \mid x) = P(x\!=\!1)H(Y \mid x\!=\!1) \, + \, P(x\!=\!0)H(Y \mid x\!=\!0)$ so the mutual information is:

$$\begin{aligned} I(X;Y) &= H(Y) - H(Y \mid X) && (9.33)\\ &= H_2(0.575) - [0.5 \times H_2(0.15) + 0.5 \times 0] && (9.34)\\ &= 0.98 - 0.30 \;=\; 0.679 \text{ bits.} && (9.35) \end{aligned}$$

Solution to exercise 9.12 (p.151). By symmetry, the optimal input distribution is $\{0.5, 0.5\}$. Then the capacity is

$$\begin{aligned} C \;=\; I(X;Y) &= H(Y) - H(Y \mid X) && (9.36)\\ &= H_2(0.5) - H_2(f) && (9.37)\\ &= 1 - H_2(f). && (9.38) \end{aligned}$$

Would you like to find the optimal input distribution without invoking symmetry? We can do this by computing the mutual information in the general case where the input ensemble is $\{p_0, p_1\}$:

$$\begin{aligned} I(X;Y) &= H(Y) - H(Y \mid X) && (9.39)\\ &= H_2(p_0 f + p_1(1-f)) - H_2(f). && (9.40) \end{aligned}$$

The only p-dependence is in the first term $H_2(p_0 f + p_1(1-f))$, which is maximized by setting the argument to 0.5. This value is given by setting $p_0 = 1/2$.

Solution to exercise 9.13 (p.151). Answer 1. By symmetry, the optimal input distribution is $\{0.5, 0.5\}$. The capacity is most easily evaluated by writing the mutual information as $I(X;Y) = H(X) - H(X \mid Y)$. The conditional entropy $H(X \mid Y)$ is $\sum_y P(y) H(X \mid y)$; when y is known, x is uncertain only if $y = ?$, which occurs with probability $f/2 + f/2$, so the conditional entropy $H(X \mid Y)$ is $f H_2(0.5)$.

$$\begin{aligned} C \;=\; I(X;Y) &= H(X) - H(X \mid Y) && (9.41)\\ &= H_2(0.5) - f H_2(0.5) && (9.42)\\ &= 1 - f. && (9.43) \end{aligned}$$

The binary erasure channel fails a fraction f of the time. Its capacity is precisely $1 - f$, which is the fraction of the time that the channel is reliable. This result seems very reasonable, but it is far from obvious how to encode information so as to communicate *reliably* over this channel.

Answer 2. Alternatively, without invoking the symmetry assumed above, we can start from the input ensemble $\{p_0, p_1\}$. The probability that $y = ?$ is $p_0 f + p_1 f = f$, and when we receive $y = ?$, the posterior probability of x is the same as the prior probability, so:

$$\begin{aligned} I(X;Y) &= H(X) - H(X \mid Y) && (9.44)\\ &= H_2(p_1) - f H_2(p_1) && (9.45)\\ &= (1-f) H_2(p_1). && (9.46) \end{aligned}$$

This mutual information achieves its maximum value of $(1-f)$ when $p_1 = 1/2$.

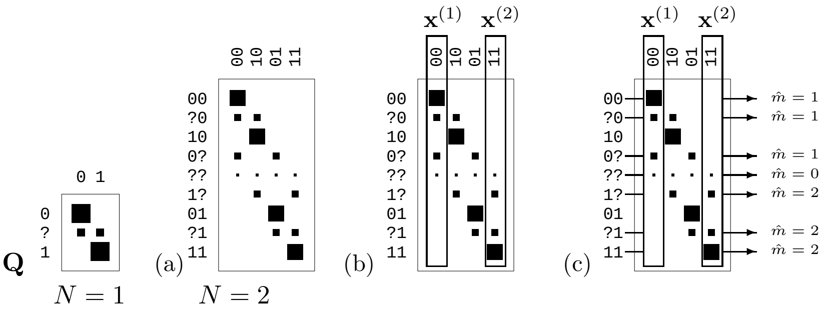


Figure 9.10. (a) The extended channel ($N = 2$) obtained from a binary erasure channel with erasure probability 0.15. (b) A block code consisting of the two codewords 00 and 11. (c) The optimal decoder for this code.

Solution to exercise 9.14 (p.153). The extended channel is shown in figure 9.10. The best code for this channel with $N = 2$ is obtained by choosing two columns that have minimal overlap, for example, columns 00 and 11. The decoding algorithm returns '00' if the extended channel output is among the top four and '11' if it's among the bottom four, and gives up if the output is '??'.

Solution to exercise 9.15 (p.155). In example 9.11 (p.151) we showed that the mutual information between input and output of the Z channel is

$$\begin{aligned} I(X;Y) &= H(Y) - H(Y \mid X) \\ &= H_2(p_1(1-f)) - p_1 H_2(f). \end{aligned} \tag{9.47}$$

We differentiate this expression with respect to p_1, taking care not to confuse $\log_2$ with $\log_e$:

$$\frac{\mathrm{d}}{\mathrm{d}p_1} I(X;Y) = (1-f)\log_2 \frac{1-p_1(1-f)}{p_1(1-f)} - H_2(f). \tag{9.48}$$

Setting this derivative to zero and rearranging using skills developed in exercise 2.17 (p.36), we obtain:

$$p_1^*(1-f) = \frac{1}{1+2^{H_2(f)/(1-f)}}, \tag{9.49}$$

so the optimal input distribution is

$$p_1^* = \frac{1/(1-f)}{1+2^{(H_2(f)/(1-f))}}. \tag{9.50}$$

As the noise level f tends to 1, this expression tends to $1/e$ (as you can prove using L'Hôpital's rule).

For all values of f, p_1^* is smaller than 1/2. A rough intuition for why input 1 is used less than input 0 is that when input 1 is used, the noisy channel injects entropy into the received string; whereas when input 0 is used, the noise has zero entropy.

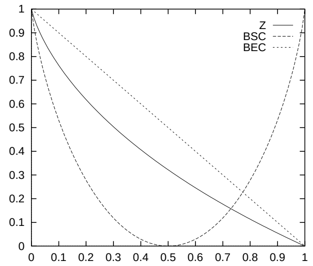


Figure 9.11. Capacities of the Z channel, binary symmetric channel, and binary erasure channel.

Solution to exercise 9.16 (p.155). The capacities of the three channels are shown in figure 9.11. For any $f < 0.5$, the BEC is the channel with highest capacity and the BSC the lowest.

Solution to exercise 9.18 (p.155). The logarithm of the posterior probability ratio, given y, is

$$a(y) = \ln \frac{P(x=1 \mid y, \alpha, \sigma)}{P(x=-1 \mid y, \alpha, \sigma)} = \ln \frac{Q(y \mid x=1, \alpha, \sigma)}{Q(y \mid x=-1, \alpha, \sigma)} = 2\frac{\alpha y}{\sigma^2}. \tag{9.51}$$

Using our skills picked up from exercise 2.17 (p.36), we rewrite this in the form

$$P(x\!=\!1 \,|\, y, \alpha, \sigma) = \frac{1}{1 + e^{-a(y)}}. \tag{9.52}$$

The optimal decoder selects the most probable hypothesis; this can be done simply by looking at the sign of $a(y)$. If $a(y) > 0$ then decode as $\hat{x} = 1$.

The probability of error is

$$p_{\mathrm{b}} = \int_{-\infty}^{0} \mathrm{d}y \, Q(y \,|\, x\!=\!1, \alpha, \sigma) = \int_{-\infty}^{-x\alpha} \mathrm{d}y \, \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{y^2}{2\sigma^2}} = \Phi\left(-\frac{x\alpha}{\sigma}\right). \tag{9.53}$$

Random coding

Solution to exercise 9.20 (p.156). The probability that $S = 24$ people whose birthdays are drawn at random from $A = 365$ days all have *distinct* birthdays is

$$\frac{A(A-1)(A-2)\ldots(A-S+1)}{A^S}. \tag{9.54}$$

The probability that two (or more) people share a birthday is one minus this quantity, which, for $S = 24$ and $A = 365$, is about 0.5. This exact way of answering the question is not very informative since it is not clear for what value of S the probability changes from being close to 0 to being close to 1.

The number of pairs is $S(S-1)/2$, and the probability that a particular pair shares a birthday is $1/A$, so the *expected number* of collisions is

$$\frac{S(S-1)}{2}\frac{1}{A}. \tag{9.55}$$

This answer is more instructive. The expected number of collisions is tiny if $S \ll \sqrt{A}$ and big if $S \gg \sqrt{A}$.

We can also approximate the probability that all birthdays are distinct, for small S, thus:

$$\begin{aligned}\frac{A(A-1)(A-2)\ldots(A-S+1)}{A^S} &= (1)(1 - {}^{1}\!/\!{}_{A})(1 - {}^{2}\!/\!{}_{A})\ldots(1 - {}^{(S-1)}\!/\!{}_{A}) \\ &\simeq \exp(0)\exp(-1/A)\exp(-2/A)\ldots\exp(-(S\!-\!1)/A) \qquad (9.56) \\ &\simeq \exp\left(-\frac{1}{A}\sum_{i=1}^{S-1} i\right) = \exp\left(-\frac{S(S-1)/2}{A}\right). \qquad (9.57)\end{aligned}$$

About Chapter 10

Before reading Chapter 10, you should have read Chapters 4 and 9. Exercise 9.14 (p. 153) is especially recommended.

Cast of characters

Q	the noisy channel
C	the capacity of the channel
X^N	an ensemble used to create a random code
$\mathcal{C}$	a random code
N	the length of the codewords
$\mathbf{x}^{(s)}$	a codeword, the sth in the code
s	the number of a chosen codeword (mnemonic: the *source* selects s)
$S = 2^K$	the total number of codewords in the code
$K = \log_2 S$	the number of bits conveyed by the choice of one codeword from S, assuming it is chosen with uniform probability
$\mathbf{s}$	a binary representation of the number s
$R = K/N$	the rate of the code, in bits per channel use (sometimes called R' instead)
$\hat{s}$	the decoder's guess of s

10

The Noisy-Channel Coding Theorem

▶ 10.1 The theorem

The theorem has three parts, two positive and one negative. The main positive result is the first.

1. For every discrete memoryless channel, the channel capacity
$$C = \max_{\mathcal{P}_X} I(X;Y) \tag{10.1}$$
has the following property. For any $\epsilon > 0$ and $R < C$, for large enough N, there exists a code of length N and rate $\geq R$ and a decoding algorithm, such that the maximal probability of block error is $< \epsilon$.

2. If a probability of bit error $p_{\rm b}$ is acceptable, rates up to $R(p_{\rm b})$ are achievable, where
$$R(p_{\rm b}) = \frac{C}{1 - H_2(p_{\rm b})}. \tag{10.2}$$

3. For any $p_{\rm b}$, rates greater than $R(p_{\rm b})$ are not achievable.

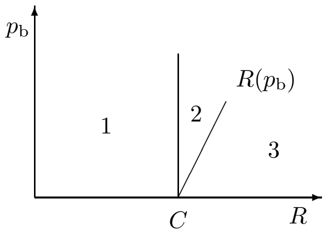


Figure 10.1. Portion of the $R, p_{\rm b}$ plane to be proved achievable $(1, 2)$ and not achievable (3).

▶ 10.2 Jointly-typical sequences

We formalize the intuitive preview of the last chapter.

We will define codewords $\mathbf{x}^{(s)}$ as coming from an ensemble X^N, and consider the random selection of one codeword and a corresponding channel output $\mathbf{y}$, thus defining a joint ensemble $(XY)^N$. We will use a *typical-set decoder*, which decodes a received signal $\mathbf{y}$ as s if $\mathbf{x}^{(s)}$ and $\mathbf{y}$ are *jointly typical*, a term to be defined shortly.

The proof will then centre on determining the probabilities (a) that the true input codeword is *not* jointly typical with the output sequence; and (b) that a *false* input codeword is jointly typical with the output. We will show that, for large N, both probabilities go to zero as long as there are fewer than 2^{NC} codewords, and the ensemble X is the optimal input distribution.

Joint typicality. A pair of sequences $\mathbf{x}, \mathbf{y}$ of length N are defined to be jointly typical (to tolerance β) with respect to the distribution $P(x,y)$ if

$$\mathbf{x} \text{ is typical of } P(\mathbf{x}), \quad \text{i.e.,} \quad \left| \frac{1}{N} \log \frac{1}{P(\mathbf{x})} - H(X) \right| < \beta,$$

$$\mathbf{y} \text{ is typical of } P(\mathbf{y}), \quad \text{i.e.,} \quad \left| \frac{1}{N} \log \frac{1}{P(\mathbf{y})} - H(Y) \right| < \beta,$$

$$\text{and } \mathbf{x}, \mathbf{y} \text{ is typical of } P(\mathbf{x}, \mathbf{y}), \quad \text{i.e.,} \quad \left| \frac{1}{N} \log \frac{1}{P(\mathbf{x}, \mathbf{y})} - H(X,Y) \right| < \beta.$$

The jointly-typical set $J_{N\beta}$ is the set of all jointly-typical sequence pairs of length N.

Example. Here is a jointly-typical pair of length $N = 100$ for the ensemble $P(x, y)$ in which $P(x)$ has $(p_0, p_1) = (0.9, 0.1)$ and $P(y \mid x)$ corresponds to a binary symmetric channel with noise level 0.2.

```
x  1111111111000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
y  0011111111000000000000000000000000000000000000000000000000000000000000000000000000111111111111111111
```

Notice that $\mathbf{x}$ has 10 1s, and so is typical of the probability $P(\mathbf{x})$ (at any tolerance β); and $\mathbf{y}$ has 26 1s, so it is typical of $P(\mathbf{y})$ (because $P(y = 1) = 0.26$); and $\mathbf{x}$ and $\mathbf{y}$ differ in 20 bits, which is the typical number of flips for this channel.

Joint typicality theorem. Let $\mathbf{x}, \mathbf{y}$ be drawn from the ensemble $(XY)^N$ defined by

$$P(\mathbf{x}, \mathbf{y}) = \prod_{n=1}^{N} P(x_n, y_n).$$

Then

1. the probability that $\mathbf{x}, \mathbf{y}$ are jointly typical (to tolerance β) tends to 1 as $N \to \infty$;
2. the number of jointly-typical sequences $|J_{N\beta}|$ is close to $2^{NH(X,Y)}$. To be precise,
$$|J_{N\beta}| \leq 2^{N(H(X,Y)+\beta)}; \tag{10.3}$$
3. if $\mathbf{x}' \sim X^N$ and $\mathbf{y}' \sim Y^N$, i.e., $\mathbf{x}'$ and $\mathbf{y}'$ are *independent* samples with the same marginal distribution as $P(\mathbf{x}, \mathbf{y})$, then the probability that $(\mathbf{x}', \mathbf{y}')$ lands in the jointly-typical set is about $2^{-NI(X;Y)}$. To be precise,
$$P((\mathbf{x}', \mathbf{y}') \in J_{N\beta}) \leq 2^{-N(I(X;Y)-3\beta)}. \tag{10.4}$$

Proof. The proof of parts 1 and 2 by the law of large numbers follows that of the source coding theorem in Chapter 4. For part 2, let the pair x, y play the role of x in the source coding theorem, replacing $P(x)$ there by the probability distribution $P(x, y)$.

For the third part,

$$\begin{aligned} P((\mathbf{x}', \mathbf{y}') \in J_{N\beta}) &= \sum_{(\mathbf{x},\mathbf{y}) \in J_{N\beta}} P(\mathbf{x})P(\mathbf{y}) && (10.5) \\ &\leq |J_{N\beta}| \, 2^{-N(H(X)-\beta)} \, 2^{-N(H(Y)-\beta)} && (10.6) \\ &\leq 2^{N(H(X,Y)+\beta)-N(H(X)+H(Y)-2\beta)} && (10.7) \\ &= 2^{-N(I(X;Y)-3\beta)}. \qquad \square && (10.8) \end{aligned}$$

A cartoon of the jointly-typical set is shown in figure 10.2. Two independent typical vectors are jointly typical with probability

$$P((\mathbf{x}', \mathbf{y}') \in J_{N\beta}) \simeq 2^{-N(I(X;Y))} \tag{10.9}$$

because the *total* number of independent typical pairs is the area of the dashed rectangle, $2^{NH(X)} 2^{NH(Y)}$, and the number of jointly-typical pairs is roughly $2^{NH(X,Y)}$, so the probability of hitting a jointly-typical pair is roughly

$$2^{NH(X,Y)}/2^{NH(X)+NH(Y)} = 2^{-NI(X;Y)}. \tag{10.10}$$

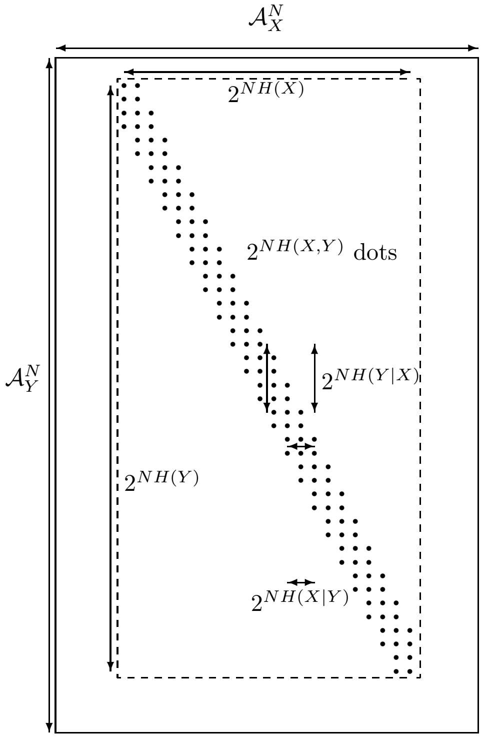


Figure 10.2. The jointly-typical set. The horizontal direction represents $\mathcal{A}_X^N$, the set of all input strings of length N. The vertical direction represents $\mathcal{A}_Y^N$, the set of all output strings of length N. The outer box contains all conceivable input–output pairs. Each dot represents a jointly-typical pair of sequences $(\mathbf{x}, \mathbf{y})$. The total number of jointly-typical sequences is about $2^{NH(X,Y)}$.

10.3 Proof of the noisy-channel coding theorem

Analogy

Imagine that we wish to prove that there is a baby in a class of one hundred babies who weighs less than 10 kg. Individual babies are difficult to catch and weigh. Shannon's method of solving the task is to scoop up all the babies and weigh them all at once on a big weighing machine. If we find that their *average* weight is smaller than 10 kg, there must exist *at least one* baby who weighs less than 10 kg – indeed there must be many! Shannon's method isn't guaranteed to reveal the existence of an underweight child, since it relies on there being a tiny number of elephants in the class. But if we use his method and get a total weight smaller than 1000 kg then our task is solved.

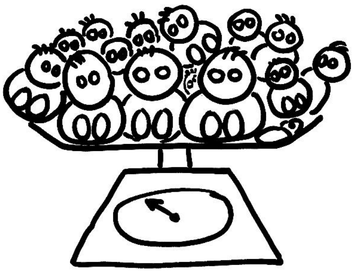


Figure 10.3. Shannon's method for proving one baby weighs less than 10 kg.

From skinny children to fantastic codes

We wish to show that there exists a code and a decoder having small probability of error. Evaluating the probability of error of any particular coding and decoding system is not easy. Shannon's innovation was this: instead of constructing a good coding and decoding system and evaluating its error probability, Shannon calculated the average probability of block error of *all* codes, and proved that this average is small. There must then exist individual codes that have small probability of block error.

Random coding and typical-set decoding

Consider the following encoding–decoding system, whose rate is R'.

1. We fix $P(x)$ and generate the $S = 2^{NR'}$ codewords of a $(N, NR') =$

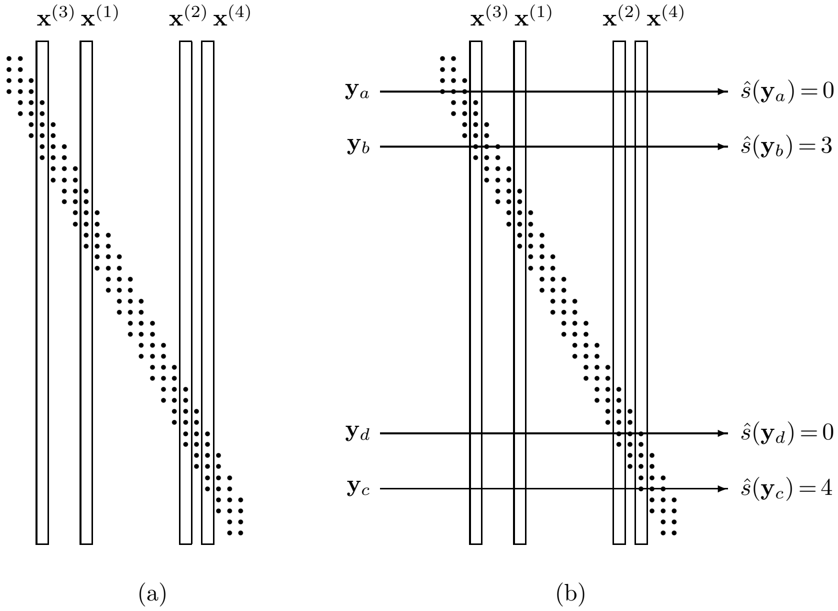

$\mathbf{x}^{(3)}$ $\mathbf{x}^{(1)}$ $\mathbf{x}^{(2)}$ $\mathbf{x}^{(4)}$
$\mathbf{y}_a$
$\mathbf{y}_b$
$\mathbf{y}_d$
$\mathbf{y}_c$
$\hat{s}(\mathbf{y}_a) = 0$
$\hat{s}(\mathbf{y}_b) = 3$
$\hat{s}(\mathbf{y}_d) = 0$
$\hat{s}(\mathbf{y}_c) = 4$
(a)
(b)

Figure 10.4. (a) A random code. (b) Example decodings by the typical set decoder. A sequence that is not jointly typical with any of the codewords, such as $\mathbf{y}_a$, is decoded as $\hat{s} = 0$. A sequence that is jointly typical with codeword $\mathbf{x}^{(3)}$ alone, $\mathbf{y}_b$, is decoded as $\hat{s} = 3$. Similarly, $\mathbf{y}_c$ is decoded as $\hat{s} = 4$. A sequence that is jointly typical with more than one codeword, such as $\mathbf{y}_d$, is decoded as $\hat{s} = 0$.

(N, K) code $\mathcal{C}$ at random according to

$$P(\mathbf{x}) = \prod_{n=1}^{N} P(x_n). \tag{10.11}$$

A random code is shown schematically in figure 10.4a.

2. The code is known to both sender and receiver.

3. A message s is chosen from $\{1, 2, \ldots, 2^{NR'}\}$, and $\mathbf{x}^{(s)}$ is transmitted. The received signal is $\mathbf{y}$, with

$$P(\mathbf{y} \,|\, \mathbf{x}^{(s)}) = \prod_{n=1}^{N} P(y_n \,|\, x_n^{(s)}). \tag{10.12}$$

4. The signal is decoded by *typical-set decoding*.

   Typical-set decoding. Decode $\mathbf{y}$ as $\hat{s}$ if $(\mathbf{x}^{(\hat{s})}, \mathbf{y})$ are jointly typical *and* there is no other s' such that $(\mathbf{x}^{(s')}, \mathbf{y})$ are jointly typical; otherwise declare a failure ($\hat{s} = 0$).

   This is not the optimal decoding algorithm, but it will be good enough, and easier to analyze. The typical-set decoder is illustrated in figure 10.4b.

5. A decoding error occurs if $\hat{s} \neq s$.

There are three probabilities of error that we can distinguish. First, there is the probability of block error for a particular code $\mathcal{C}$, that is,

$$p_{\rm B}(\mathcal{C}) \equiv P(\hat{s} \neq s \,|\, \mathcal{C}). \tag{10.13}$$

This is a difficult quantity to evaluate for any given code.

Second, there is the average over all codes of this block error probability,

$$\langle p_{\rm B} \rangle \equiv \sum_{\mathcal{C}} P(\hat{s} \neq s \,|\, \mathcal{C}) P(\mathcal{C}). \tag{10.14}$$

Fortunately, this quantity is much easier to evaluate than the first quantity $P(\hat{s} \neq s \,|\, \mathcal{C})$.

$\langle p_{\rm B} \rangle$ is just the probability that there is a decoding error at step 5 of the five-step process on the previous page.

Third, the maximal block error probability of a code $\mathcal{C}$,

$$p_{\rm BM}(\mathcal{C}) \equiv \max_{s} P(\hat{s} \neq s \,|\, s, \mathcal{C}), \tag{10.15}$$

is the quantity we are most interested in: we wish to show that there exists a code $\mathcal{C}$ with the required rate whose maximal block error probability is small.

We will get to this result by first finding the average block error probability, $\langle p_{\rm B} \rangle$. Once we have shown that this can be made smaller than a desired small number, we immediately deduce that there must exist *at least one* code $\mathcal{C}$ whose block error probability is also less than this small number. Finally, we show that this code, whose block error probability is satisfactorily small but whose maximal block error probability is unknown (and could conceivably be enormous), can be modified to make a code of slightly smaller rate whose maximal block error probability is also guaranteed to be small. We modify the code by throwing away the worst 50% of its codewords.

We therefore now embark on finding the average probability of block error.

Probability of error of typical-set decoder

There are two sources of error when we use typical-set decoding. Either (a) the output $\mathbf{y}$ is not jointly typical with the transmitted codeword $\mathbf{x}^{(s)}$, or (b) there is some other codeword in $\mathcal{C}$ that is jointly typical with $\mathbf{y}$.

By the symmetry of the code construction, the average probability of error averaged over all codes does not depend on the selected value of s; we can assume without loss of generality that $s = 1$.

(a) The probability that the input $\mathbf{x}^{(1)}$ and the output $\mathbf{y}$ are not jointly typical vanishes, by the joint typicality theorem's first part (p.163). We give a name, δ, to the upper bound on this probability, satisfying $\delta \to 0$ as $N \to \infty$; for any desired δ, we can find a blocklength $N(\delta)$ such that the $P((\mathbf{x}^{(1)}, \mathbf{y}) \notin J_{N\beta}) \leq \delta$.

(b) The probability that $\mathbf{x}^{(s')}$ and $\mathbf{y}$ are jointly typical, for a *given* $s' \neq 1$ is $\leq 2^{-N(I(X;Y)-3\beta)}$, by part 3. And there are $(2^{NR'} - 1)$ rival values of s' to worry about.

Thus the average probability of error $\langle p_{\rm B} \rangle$ satisfies:

$$\begin{aligned} \langle p_{\rm B} \rangle \;&\leq\; \delta + \sum_{s'=2}^{2^{NR'}} 2^{-N(I(X;Y)-3\beta)} & (10.16)\\ &\leq\; \delta + 2^{-N(I(X;Y)-R'-3\beta)}. & (10.17)\end{aligned}$$

The inequality (10.16) that bounds a total probability of error $P_{\rm TOT}$ by the sum of the probabilities $P_{s'}$ of all sorts of events s' each of which is sufficient to cause error,

$$P_{\rm TOT} \leq P_1 + P_2 + \cdots,$$

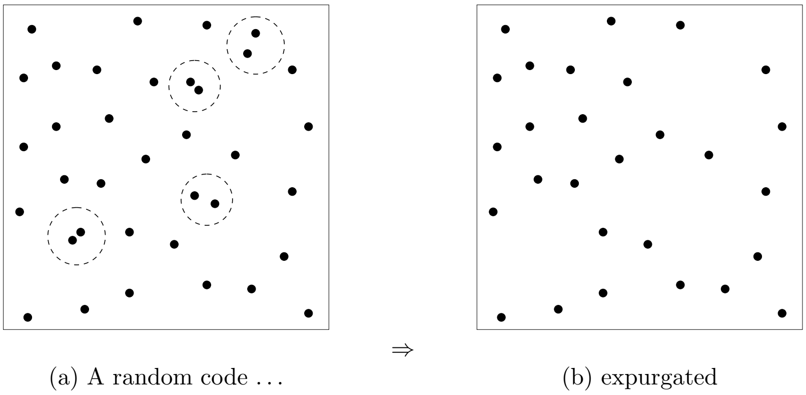
is called a union bound. It is only an equality if the different events that cause error never occur at the same time as each other.

The average probability of error (10.17) can be made $< 2\delta$ by increasing N if

$$R' < I(X;Y) - 3\beta. \tag{10.18}$$

We are almost there. We make three modifications:

1. We choose $P(x)$ in the proof to be the optimal input distribution of the channel. Then the condition $R' < I(X;Y) - 3\beta$ becomes $R' < C - 3\beta$.



(a) A random code . . .

(b) expurgated

Figure 10.5. How expurgation works. (a) In a typical random code, a small fraction of the codewords are involved in collisions – pairs of codewords are sufficiently close to each other that the probability of error when either codeword is transmitted is not tiny. We obtain a new code from a random code by deleting all these confusable codewords. (b) The resulting code has slightly fewer codewords, so has a slightly lower rate, and its maximal probability of error is greatly reduced.

2. Since the average probability of error over all codes is $< 2\delta$, there must exist *a* code with mean probability of block error $p_{\rm B}(\mathcal{C}) < 2\delta$.

3. To show that not only the average but also the maximal probability of error, $p_{\rm BM}$, can be made small, we modify this code by throwing away the worst half of the codewords – the ones most likely to produce errors. Those that remain must all have *conditional* probability of error less than 4δ. We use these remaining codewords to define a new code. This new code has $2^{NR'-1}$ codewords, i.e., we have reduced the rate from R' to $R' - 1/N$ (a negligible reduction, if N is large), and achieved $p_{\rm BM} < 4\delta$. This trick is called *expurgation* (figure 10.5). The resulting code may not be the best code of its rate and length, but it is still good enough to prove the noisy-channel coding theorem, which is what we are trying to do here.

In conclusion, we can 'construct' a code of rate $R' - 1/N$, where $R' < C - 3\beta$, with maximal probability of error $< 4\delta$. We obtain the theorem as stated by setting $R' = (R + C)/2$, $\delta = \epsilon/4$, $\beta < (C - R')/3$, and N sufficiently large for the remaining conditions to hold. The theorem's first part is thus proved. □

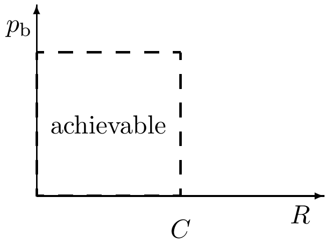


Figure 10.6. Portion of the $R, p_{\rm b}$ plane proved achievable in the first part of the theorem. [We've proved that the maximal probability of block error $p_{\rm BM}$ can be made arbitrarily small, so the same goes for the bit error probability $p_{\rm b}$, which must be smaller than $p_{\rm BM}$.]

▶ 10.4 Communication (with errors) above capacity

We have proved, for any discrete memoryless channel, the achievability of a portion of the $R, p_{\rm b}$ plane shown in figure 10.6. We have shown that we can turn any noisy channel into an essentially noiseless binary channel with rate up to C bits per cycle. We now extend the right-hand boundary of the region of achievability at non-zero error probabilities. [This is called *rate-distortion theory*.]

We do this with a new trick. Since we know we can make the noisy channel into a perfect channel with a smaller rate, it is sufficient to consider communication with errors over a *noiseless* channel. How fast can we communicate over a noiseless channel, if we are allowed to make errors?

Consider a noiseless binary channel, and assume that we force communication at a rate greater than its capacity of 1 bit. For example, if we require the sender to attempt to communicate at $R = 2$ bits per cycle then he must effectively throw away half of the information. What is the best way to do this if the aim is to achieve the smallest possible probability of bit error? One simple strategy is to communicate a fraction $1/R$ of the source bits, and ignore the rest. The receiver guesses the missing fraction $1 - 1/R$ at random, and the average probability of bit error is

$$p_{\rm b} = \frac{1}{2}(1 - 1/R). \tag{10.19}$$

The curve corresponding to this strategy is shown by the dashed line in figure 10.7.

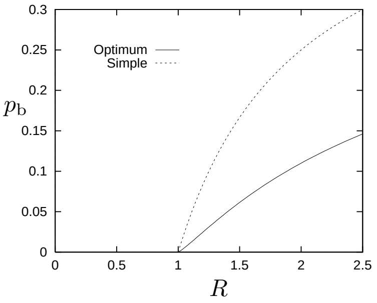


Figure 10.7. A simple bound on achievable points $(R, p_{\rm b})$, and Shannon's bound.

We can do better than this (in terms of minimizing $p_{\rm b}$) by spreading out the risk of corruption evenly among all the bits. In fact, we can achieve $p_{\rm b} = H_2^{-1}(1 - 1/R)$, which is shown by the solid curve in figure 10.7. So, how can this optimum be achieved?

We reuse a tool that we just developed, namely the (N, K) code for a noisy channel, and we turn it on its head, using the *decoder* to define a lossy compressor. Specifically, we take an excellent (N, K) code for the binary symmetric channel. Assume that such a code has a rate $R' = K/N$, and that it is capable of correcting errors introduced by a binary symmetric channel whose transition probability is q. Asymptotically, rate-R' codes exist that have $R' \simeq 1 - H_2(q)$. Recall that, if we attach one of these capacity-achieving codes of length N to a binary symmetric channel then (a) the probability distribution over the outputs is close to uniform, since the entropy of the output is equal to the entropy of the source (NR') plus the entropy of the noise $(NH_2(q))$, and (b) the optimal decoder of the code, in this situation, typically maps a received vector of length N to a transmitted vector differing in qN bits from the received vector.

We take the signal that we wish to send, and chop it into blocks of length N (yes, N, not K). We pass each block through the *decoder*, and obtain a shorter signal of length K bits, which we communicate over the noiseless channel. To decode the transmission, we pass the K bit message to the *encoder* of the original code. The reconstituted message will now differ from the original message in some of its bits – typically qN of them. So the probability of bit error will be $p_{\rm b} = q$. The rate of this lossy compressor is $R = N/K = 1/R' = 1/(1 - H_2(p_{\rm b}))$.

Now, attaching this lossy compressor to our capacity-C error-free communicator, we have proved the achievability of communication up to the curve $(p_{\rm b}, R)$ defined by:

$$R = \frac{C}{1 - H_2(p_{\rm b})}. \qquad \Box \tag{10.20}$$

For further reading about rate-distortion theory, see Gallager (1968), p. 451, or McEliece (2002), p. 75.

▶ 10.5 The non-achievable region (part 3 of the theorem)

The source, encoder, noisy channel and decoder define a Markov chain:

$s \rightarrow \mathbf{x} \rightarrow \mathbf{y} \rightarrow \hat{s}$

$$P(s, \mathbf{x}, \mathbf{y}, \hat{s}) = P(s)P(\mathbf{x} \,|\, s)P(\mathbf{y} \,|\, \mathbf{x})P(\hat{s} \,|\, \mathbf{y}). \tag{10.21}$$

The data processing inequality (exercise 8.9, p.141) must apply to this chain: $I(s; \hat{s}) \leq I(\mathbf{x}; \mathbf{y})$. Furthermore, by the definition of channel capacity, $I(\mathbf{x}; \mathbf{y}) \leq NC$, so $I(s; \hat{s}) \leq NC$.

Assume that a system achieves a rate R and a bit error probability $p_{\rm b}$; then the mutual information $I(s; \hat{s})$ is $\geq NR(1 - H_2(p_{\rm b}))$. But $I(s; \hat{s}) > NC$ is not achievable, so $R > \frac{C}{1 - H_2(p_{\rm b})}$ is not achievable. □

Exercise 10.1.[3] Fill in the details in the preceding argument. If the bit errors between $\hat{s}$ and s are independent then we have $I(s; \hat{s}) = NR(1 - H_2(p_{\rm b}))$.

What if we have complex correlations among those bit errors? Why does the inequality $I(s;\hat{s}) \geq NR(1 - H_2(p_\text{b}))$ hold?

▶ 10.6 Computing capacity

We have proved that the capacity of a channel is the maximum rate at which reliable communication can be achieved. How can we compute the capacity of a given discrete memoryless channel? We need to find its optimal input distribution. In general we can find the optimal input distribution by a computer search, making use of the derivative of the mutual information with respect to the input probabilities.

Sections 10.6–10.8 contain advanced material. The first-time reader is encouraged to skip to section 10.9 (p.172).

▷ Exercise 10.2.[2] Find the derivative of $I(X;Y)$ with respect to the input probability p_i, $\partial I(X;Y)/\partial p_i$, for a channel with conditional probabilities $Q_{j|i}$.

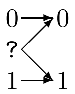
Exercise 10.3.[2] Show that $I(X;Y)$ is a concave $\frown$ function of the input probability vector $\mathbf{p}$.

Since $I(X;Y)$ is concave $\frown$ in the input distribution $\mathbf{p}$, any probability distribution $\mathbf{p}$ at which $I(X;Y)$ is stationary must be a global maximum of $I(X;Y)$. So it is tempting to put the derivative of $I(X;Y)$ into a routine that finds a local maximum of $I(X;Y)$, that is, an input distribution $P(x)$ such that

$$\frac{\partial I(X;Y)}{\partial p_i} = \lambda \quad \text{for all } i, \tag{10.22}$$

where λ is a Lagrange multiplier associated with the constraint $\sum_i p_i = 1$. However, this approach may fail to find the right answer, because $I(X;Y)$ might be maximized by a distribution that has $p_i = 0$ for some inputs. A simple example is given by the ternary confusion channel.

Ternary confusion channel. $\mathcal{A}_X = \{0, ?, 1\}$. $\mathcal{A}_Y = \{0, 1\}$.



$$\begin{array}{lcllcllcl} P(y\!=\!0\,|\,x\!=\!0) & = & 1\,; & P(y\!=\!0\,|\,x\!=\!?) & = & 1/2\,; & P(y\!=\!0\,|\,x\!=\!1) & = & 0\,; \\ P(y\!=\!1\,|\,x\!=\!0) & = & 0\,; & P(y\!=\!1\,|\,x\!=\!?) & = & 1/2\,; & P(y\!=\!1\,|\,x\!=\!1) & = & 1. \end{array}$$

Whenever the input ? is used, the output is random; the other inputs are reliable inputs. The maximum information rate of 1 bit is achieved by making no use of the input ?.

▷ Exercise 10.4.[2, p.173] Sketch the mutual information for this channel as a function of the input distribution $\mathbf{p}$. Pick a convenient two-dimensional representation of $\mathbf{p}$.

The optimization routine must therefore take account of the possibility that, as we go up hill on $I(X;Y)$, we may run into the inequality constraints $p_i \geq 0$.

▷ Exercise 10.5.[2, p.174] Describe the condition, similar to equation (10.22), that is satisfied at a point where $I(X;Y)$ is maximized, and describe a computer program for finding the capacity of a channel.

Results that may help in finding the optimal input distribution

1. All outputs must be used.
2. $I(X;Y)$ is a convex$\smile$ function of the channel parameters.
3. There may be several optimal input distributions, but they all look the same at the output.

Reminder: The term 'convex$\smile$' means 'convex', and the term 'concave$\frown$' means 'concave'; the little smile and frown symbols are included simply to remind you what convex and concave mean.

▷ Exercise 10.6.$^{[2]}$ Prove that no output y is unused by an optimal input distribution, unless it is unreachable, that is, has $Q(y\,|\,x) = 0$ for all x.

Exercise 10.7.$^{[2]}$ Prove that $I(X;Y)$ is a convex$\smile$ function of $Q(y\,|\,x)$.

Exercise 10.8.$^{[2]}$ Prove that all optimal input distributions of a channel have the same output probability distribution $P(y) = \sum_x P(x)Q(y\,|\,x)$.

These results, along with the fact that $I(X;Y)$ is a concave$\frown$ function of the input probability vector $\mathbf{p}$, prove the validity of the symmetry argument that we have used when finding the capacity of symmetric channels. If a channel is invariant under a group of symmetry operations – for example, interchanging the input symbols and interchanging the output symbols – then, given any optimal input distribution that is not symmetric, i.e., is not invariant under these operations, we can create another input distribution by averaging together this optimal input distribution and all its permuted forms that we can make by applying the symmetry operations to the original optimal input distribution. The permuted distributions must have the same $I(X;Y)$ as the original, by symmetry, so the new input distribution created by averaging must have $I(X;Y)$ bigger than or equal to that of the original distribution, because of the concavity of I.

Symmetric channels

In order to use symmetry arguments, it will help to have a definition of a symmetric channel. I like Gallager's (1968) definition.

A discrete memoryless channel is a symmetric channel if the set of outputs can be partitioned into subsets in such a way that for each subset the matrix of transition probabilities has the property that each row (if more than 1) is a permutation of each other row and each column is a permutation of each other column.

Example 10.9. This channel

$$\begin{array}{rclrcl} P(y\!=\!0\,|\,x\!=\!0) &=& 0.7\,; & P(y\!=\!0\,|\,x\!=\!1) &=& 0.1\,; \\ P(y\!=\!\texttt{?}\,|\,x\!=\!0) &=& 0.2\,; & P(y\!=\!\texttt{?}\,|\,x\!=\!1) &=& 0.2\,; \\ P(y\!=\!1\,|\,x\!=\!0) &=& 0.1\,; & P(y\!=\!1\,|\,x\!=\!1) &=& 0.7. \end{array} \tag{10.23}$$

is a symmetric channel because its outputs can be partitioned into $(0,1)$ and $\texttt{?}$, so that the matrix can be rewritten:

$$\begin{array}{rclrcl} \hline P(y\!=\!0\,|\,x\!=\!0) &=& 0.7\,; & P(y\!=\!0\,|\,x\!=\!1) &=& 0.1\,; \\ P(y\!=\!1\,|\,x\!=\!0) &=& 0.1\,; & P(y\!=\!1\,|\,x\!=\!1) &=& 0.7\,; \\ \hline P(y\!=\!\texttt{?}\,|\,x\!=\!0) &=& 0.2\,; & P(y\!=\!\texttt{?}\,|\,x\!=\!1) &=& 0.2. \\ \hline \end{array} \tag{10.24}$$

Symmetry is a useful property because, as we will see in a later chapter, communication at capacity can be achieved over symmetric channels by *linear* codes.

Exercise 10.10.[2] Prove that for a symmetric channel with any number of inputs, the uniform distribution over the inputs is an optimal input distribution.

▷ Exercise 10.11.[2, p.174] Are there channels that are not symmetric whose optimal input distributions are uniform? Find one, or prove there are none.

▶ 10.7 Other coding theorems

The noisy-channel coding theorem that we proved in this chapter is quite general, applying to any discrete memoryless channel; but it is not very specific. The theorem only says that reliable communication with error probability ϵ and rate R can be achieved by using codes with *sufficiently large* blocklength N. The theorem does not say how large N needs to be to achieve given values of R and ϵ.

Presumably, the smaller ϵ is and the closer R is to C, the larger N has to be.

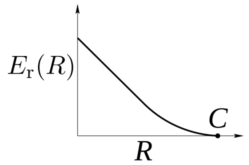


Figure 10.8. A typical random-coding exponent.

Noisy-channel coding theorem – version with explicit N-dependence

> For a discrete memoryless channel, a blocklength N and a rate R, there exist block codes of length N whose average probability of error satisfies:
> $$p_B \leq \exp\left[-NE_r(R)\right] \tag{10.25}$$
> where $E_r(R)$ is the *random-coding exponent* of the channel, a convex ⌣, decreasing, positive function of R for $0 \leq R < C$. The random-coding exponent is also known as the reliability function.
>
> [By an expurgation argument it can also be shown that there exist block codes for which the *maximal* probability of error p_{BM} is also exponentially small in N.]

The definition of $E_r(R)$ is given in Gallager (1968), p. 139. $E_r(R)$ approaches zero as $R \to C$; the typical behaviour of this function is illustrated in figure 10.8. The computation of the random-coding exponent for interesting channels is a challenging task on which much effort has been expended. Even for simple channels like the binary symmetric channel, there is no simple expression for $E_r(R)$.

Lower bounds on the error probability as a function of blocklength

The theorem stated above asserts that there are codes with p_B smaller than $\exp\left[-NE_r(R)\right]$. But how small can the error probability be? Could it be much smaller?

> For any code with blocklength N on a discrete memoryless channel, the probability of error assuming all source messages are used with equal probability satisfies
> $$p_B \gtrsim \exp[-NE_{sp}(R)], \tag{10.26}$$

where the function $E_{\mathrm{sp}}(R)$, the *sphere-packing exponent* of the channel, is a convex $\smile$, decreasing, positive function of R for $0 \leq R < C$.

For a precise statement of this result and further references, see Gallager (1968), p. 157.

▶ 10.8 Noisy-channel coding theorems and coding practice

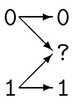
Imagine a customer who wants to buy an error-correcting code and decoder for a noisy channel. The results described above allow us to offer the following service: if he tells us the properties of his channel, the desired rate R and the desired error probability p_{B}, we can, after working out the relevant functions C, $E_{\mathrm{r}}(R)$, and $E_{\mathrm{sp}}(R)$, advise him that there exists a solution to his problem using a particular blocklength N; indeed that almost any randomly chosen code with that blocklength should do the job. Unfortunately we have not found out how to implement these encoders and decoders in practice; the cost of implementing the encoder and decoder for a random code with large N would be exponentially large in N.

Furthermore, for practical purposes, the customer is unlikely to know exactly what channel he is dealing with. So Berlekamp (1980) suggests that the sensible way to approach error-correction is to design encoding-decoding systems and plot their performance on a *variety* of idealized channels as a function of the channel's noise level. These charts (one of which is illustrated on page 568) can then be shown to the customer, who can choose among the systems on offer without having to specify what he really thinks his channel is like. With this attitude to the practical problem, the importance of the functions $E_{\mathrm{r}}(R)$ and $E_{\mathrm{sp}}(R)$ is diminished.

▶ 10.9 Further exercises

Exercise 10.12.$^{[2]}$ A binary erasure channel with input x and output y has transition probability matrix:

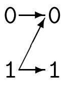
$$\mathbf{Q} = \begin{bmatrix} 1-q & 0 \\ q & q \\ 0 & 1-q \end{bmatrix}$$



Find the *mutual information* $I(X;Y)$ between the input and output for general input distribution $\{p_0, p_1\}$, and show that the *capacity* of this channel is $C = 1 - q$ bits.

A Z channel has transition probability matrix:

$$\mathbf{Q} = \begin{bmatrix} 1 & q \\ 0 & 1-q \end{bmatrix}$$



Show that, using a $(2, 1)$ code, **two** uses of a Z channel can be made to emulate **one** use of an erasure channel, and state the erasure probability of that erasure channel. Hence show that the capacity of the Z channel, C_{Z}, satisfies $C_{\mathrm{Z}} \geq \frac{1}{2}(1 - q)$ bits.

Explain why the result $C_{\mathrm{Z}} \geq \frac{1}{2}(1 - q)$ is an inequality rather than an equality.

Exercise 10.13.[3, p.174] A transatlantic cable contains $N = 20$ indistinguishable electrical wires. You have the job of figuring out which wire is which, that is, to create a consistent labelling of the wires at each end. Your only tools are the ability to connect wires to each other in groups of two or more, and to test for connectedness with a continuity tester. What is the smallest number of transatlantic trips you need to make, and how do you do it?

How would you solve the problem for larger N such as $N = 1000$?

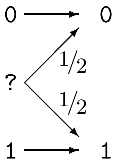
As an illustration, if N were 3 then the task can be solved in two steps by labelling one wire at one end a, connecting the other two together, crossing the Atlantic, measuring which two wires are connected, labelling them b and c and the unconnected one a, then connecting b to a and returning across the Atlantic, whereupon on disconnecting b from c, the identities of b and c can be deduced.

This problem can be solved by persistent search, but the reason it is posed in this chapter is that it can also be solved by a greedy approach based on maximizing the acquired *information*. Let the unknown permutation of wires be x. Having chosen a set of connections of wires $\mathcal{C}$ at one end, you can then make measurements at the other end, and these measurements y convey *information* about x. How much? And for what set of connections is the information that y conveys about x maximized?

▶ 10.10 Solutions

Solution to exercise 10.4 (p.169). If the input distribution is $\mathbf{p} = (p_0, p_?, p_1)$, the mutual information is

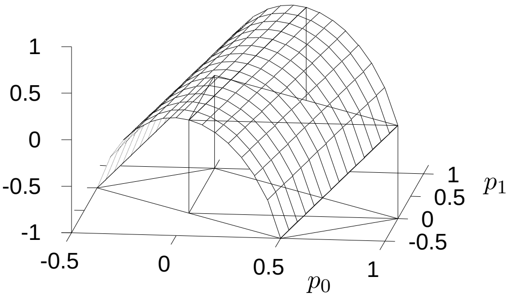
$$I(X;Y) = H(Y) - H(Y|X) = H_2(p_0 + p_?/2) - p_?. \tag{10.27}$$

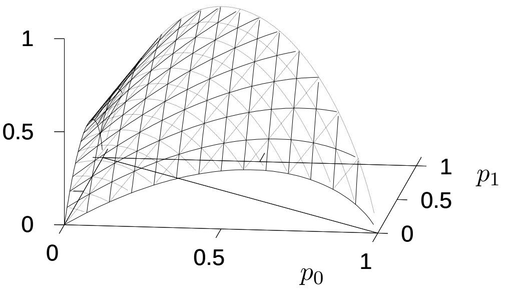


We can build a good sketch of this function in two ways: by careful inspection of the function, or by looking at special cases.

For the plots, the two-dimensional representation of $\mathbf{p}$ I will use has p_0 and p_1 as the independent variables, so that $\mathbf{p} = (p_0, p_?, p_1) = (p_0, (1-p_0-p_1), p_1)$.

By inspection. If we use the quantities $p_* \equiv p_0 + p_?/2$ and $p_?$ as our two degrees of freedom, the mutual information becomes very simple: $I(X;Y) = H_2(p_*) - p_?$. Converting back to $p_0 = p_* - p_?/2$ and $p_1 = 1 - p_* - p_?/2$, we obtain the sketch shown at the left below. This function is like a tunnel rising up the direction of increasing p_0 and p_1. To obtain the required plot of $I(X;Y)$ we have to strip away the parts of this tunnel that live outside the feasible simplex of probabilities; we do this by redrawing the surface, showing only the parts where $p_0 > 0$ and $p_1 > 0$. A full plot of the function is shown at the right.





Special cases. In the special case $p_? = 0$, the channel is a noiseless binary channel, and $I(X;Y) = H_2(p_0)$.

In the special case $p_0 = p_1$, the term $H_2(p_0 + p_?/2)$ is equal to 1, so $I(X;Y) = 1 - p_?$.

In the special case $p_0 = 0$, the channel is a Z channel with error probability 0.5. We know how to sketch that, from the previous chapter (figure 9.3).

These special cases allow us to construct the skeleton shown in figure 10.9.

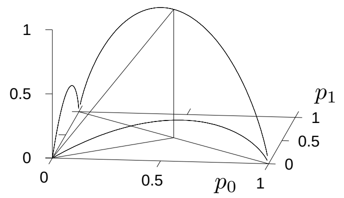


Figure 10.9. Skeleton of the mutual information for the ternary confusion channel.

Solution to exercise 10.5 (p.169). Necessary and sufficient conditions for $\mathbf{p}$ to maximize $I(X;Y)$ are

$$\left.\begin{array}{rcl} \frac{\partial I(X;Y)}{\partial p_i} & = \lambda & \text{and} \quad p_i > 0 \\ \frac{\partial I(X;Y)}{\partial p_i} & \leq \lambda & \text{and} \quad p_i = 0 \end{array}\right\} \text{ for all } i, \tag{10.28}$$

where λ is a constant related to the capacity by $C = \lambda + \log_2 e$.

This result can be used in a computer program that evaluates the derivatives, and increments and decrements the probabilities p_i in proportion to the differences between those derivatives.

This result is also useful for lazy human capacity-finders who are good guessers. Having guessed the optimal input distribution, one can simply confirm that equation (10.28) holds.

Solution to exercise 10.11 (p.171). We certainly expect nonsymmetric channels with uniform optimal input distributions to exist, since when inventing a channel we have $I(J-1)$ degrees of freedom whereas the optimal input distribution is just $(I-1)$-dimensional; so in the $I(J-1)$-dimensional space of perturbations around a symmetric channel, we expect there to be a subspace of perturbations of dimension $I(J-1)-(I-1) = I(J-2)+1$ that leave the optimal input distribution unchanged.

Here is an explicit example, a bit like a Z channel.

$$\mathbf{Q} = \left[\begin{array}{cccc} 0.9585 & 0.0415 & 0.35 & 0.0 \\ 0.0415 & 0.9585 & 0.0 & 0.35 \\ 0 & 0 & 0.65 & 0 \\ 0 & 0 & 0 & 0.65 \end{array}\right] \tag{10.29}$$

Solution to exercise 10.13 (p.173). The labelling problem can be solved for any $N > 2$ with just two trips, one each way across the Atlantic.

The key step in the information-theoretic approach to this problem is to write down the information content of one *partition*, the combinatorial object that is the connecting together of subsets of wires. If N wires are grouped together into g_1 subsets of size 1, g_2 subsets of size 2, ..., then the number of such partitions is

$$\Omega = \frac{N!}{\prod_r (r!)^{g_r} g_r!}, \tag{10.30}$$

and the information content of one such partition is the log of this quantity. In a greedy strategy we choose the first partition to maximize this information content.

One game we can play is to maximize this information content with respect to the quantities g_r, treated as real numbers, subject to the constraint $\sum_r g_r r = N$. Introducing a Lagrange multiplier λ for the constraint, the derivative is

$$\frac{\partial}{\partial g_r}\left(\log \Omega + \lambda \sum_r g_r r\right) = -\log r! - \log g_r + \lambda r, \tag{10.31}$$

which, when set to zero, leads to the rather nice expression

$$g_r = \frac{e^{\lambda r}}{r!}; \tag{10.32}$$

the optimal g_r is proportional to a Poisson distribution! We can solve for the Lagrange multiplier by plugging g_r into the constraint $\sum_r g_r r = N$, which gives the implicit equation

$$N = \mu\, e^{\mu}, \tag{10.33}$$

where $\mu \equiv e^{\lambda}$ is a convenient reparameterization of the Lagrange multiplier. Figure 10.10a shows a graph of $\mu(N)$; figure 10.10b shows the deduced non-integer assignments g_r when $\mu = 2.2$, and nearby integers $g_r = \{1, 2, 2, 1, 1\}$ that motivate setting the first partition to (a)(bc)(de)(fgh)(ijk)(lmno)(pqrst).

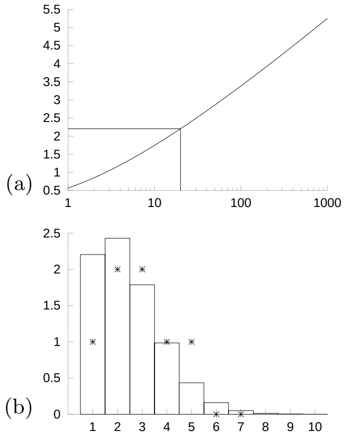


Figure 10.10. Approximate solution of the cable-labelling problem using Lagrange multipliers. (a) The parameter μ as a function of N; the value $\mu(20) = 2.2$ is highlighted. (b) Non-integer values of the function $g_r = {}^{\mu^r}\!/r!$ are shown by lines and integer values of g_r motivated by those non-integer values are shown by crosses.

This partition produces a random partition at the other end, which has an information content of $\log \Omega = 40.4$ bits, which is a lot more than half the total information content we need to acquire to infer the transatlantic permutation, $\log 20! \simeq 61$ bits. [In contrast, if all the wires are joined together in pairs, the information content generated is only about 29 bits.] How to choose the second partition is left to the reader. A Shannonesque approach is appropriate, picking a random partition at the other end, using the same $\{g_r\}$; you need to ensure the two partitions are as unlike each other as possible.

If $N \neq 2$, 5 or 9, then the labelling problem has solutions that are particularly simple to implement, called Knowlton–Graham partitions: partition $\{1, \ldots, N\}$ into disjoint sets in two ways A and B, subject to the condition that at most one element appears both in an A set of cardinality j and in a B set of cardinality k, for each j and k (Graham, 1966; Graham and Knowlton, 1968).

About Chapter 11

Before reading Chapter 11, you should have read Chapters 9 and 10.

You will also need to be familiar with the *Gaussian distribution*.

One-dimensional Gaussian distribution. If a random variable y is Gaussian and has mean μ and variance σ^2, which we write:

$$y \sim \text{Normal}(\mu, \sigma^2), \text{ or } P(y) = \text{Normal}(y; \mu, \sigma^2), \tag{11.1}$$

then the distribution of y is:

$$P(y \,|\, \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-(y-\mu)^2/2\sigma^2\right]. \tag{11.2}$$

[I use the symbol P for both probability densities and probabilities.]

The inverse-variance $\tau \equiv {}^1\!/\!{}_{\sigma^2}$ is sometimes called the *precision* of the Gaussian distribution.

Multi-dimensional Gaussian distribution. If $\mathbf{y} = (y_1, y_2, \ldots, y_N)$ has a multivariate Gaussian distribution, then

$$P(\mathbf{y} \,|\, \mathbf{x}, \mathbf{A}) = \frac{1}{Z(\mathbf{A})} \exp\left(-\frac{1}{2}(\mathbf{y}-\mathbf{x})^{\mathsf{T}}\mathbf{A}(\mathbf{y}-\mathbf{x})\right), \tag{11.3}$$

where $\mathbf{x}$ is the mean of the distribution, $\mathbf{A}$ is the inverse of the variance–covariance matrix, and the normalizing constant is $Z(\mathbf{A}) = (\det(\mathbf{A}/2\pi))^{-1/2}$.

This distribution has the property that the variance Σ_{ii} of y_i, and the covariance Σ_{ij} of y_i and y_j are given by

$$\Sigma_{ij} \equiv \mathcal{E}\left[(y_i - \bar{y}_i)(y_j - \bar{y}_j)\right] = A^{-1}_{ij}, \tag{11.4}$$

where $\mathbf{A}^{-1}$ is the inverse of the matrix $\mathbf{A}$.

The marginal distribution $P(y_i)$ of one component y_i is Gaussian; the joint marginal distribution of any subset of the components is multivariate-Gaussian; and the conditional density of any subset, given the values of another subset, for example, $P(y_i \,|\, y_j)$, is also Gaussian.

11

Error-Correcting Codes & Real Channels

The noisy-channel coding theorem that we have proved shows that there exist reliable error-correcting codes for any noisy channel. In this chapter we address two questions.

First, many practical channels have real, rather than discrete, inputs and outputs. What can Shannon tell us about these continuous channels? And how should digital signals be mapped into analogue waveforms, and *vice versa*?

Second, how are practical error-correcting codes made, and what is achieved in practice, relative to the possibilities proved by Shannon?

▶ 11.1 The Gaussian channel

The most popular model of a real-input, real-output channel is the Gaussian channel.

The Gaussian channel has a real input x and a real output y. The conditional distribution of y given x is a Gaussian distribution:

$$P(y \mid x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-(y-x)^2/2\sigma^2\right]. \tag{11.5}$$

This channel has a continuous input and output but is discrete in time. We will show below that certain continuous-time channels are equivalent to the discrete-time Gaussian channel.

This channel is sometimes called the additive white Gaussian noise (AWGN) channel.

As with discrete channels, we will discuss what rate of error-free information communication can be achieved over this channel.

Motivation in terms of a continuous-time channel

Consider a physical (electrical, say) channel with inputs and outputs that are continuous in time. We put in $x(t)$, and out comes $y(t) = x(t) + n(t)$.

Our transmission has a power cost. The average power of a transmission of length T may be constrained thus:

$$\int_0^T \mathrm{d}t\, [x(t)]^2/T \leq P. \tag{11.6}$$

The received signal is assumed to differ from $x(t)$ by additive noise $n(t)$ (for example Johnson noise), which we will model as white Gaussian noise. The magnitude of this noise is quantified by the *noise spectral density*, N_0.

How could such a channel be used to communicate information? Consider transmitting a set of N real numbers $\{x_n\}_{n=1}^{N}$ in a signal of duration T made up of a weighted combination of orthonormal basis functions $\phi_n(t)$,

$$x(t) = \sum_{n=1}^{N} x_n \phi_n(t), \qquad (11.7)$$

where $\int_0^T \mathrm{d}t\, \phi_n(t)\phi_m(t) = \delta_{nm}$. The receiver can then compute the scalars:

$$\begin{aligned} y_n \;\equiv\; \int_0^T \mathrm{d}t\, \phi_n(t) y(t) &= x_n + \int_0^T \mathrm{d}t\, \phi_n(t) n(t) && (11.8)\\ &\equiv x_n + n_n && (11.9)\end{aligned}$$

for $n = 1 \ldots N$. If there were no noise, then y_n would equal x_n. The white Gaussian noise $n(t)$ adds scalar noise n_n to the estimate y_n. This noise is Gaussian:

$$n_n \sim \text{Normal}(0, N_0/2), \qquad (11.10)$$

where N_0 is the spectral density introduced above. Thus a continuous channel used in this way is equivalent to the Gaussian channel defined at equation (11.5). The power constraint $\int_0^T \mathrm{d}t\, [x(t)]^2 \leq PT$ defines a constraint on the signal amplitudes x_n,

$$\sum_n x_n^2 \leq PT \qquad \Rightarrow \qquad \overline{x_n^2} \leq \frac{PT}{N}. \qquad (11.11)$$

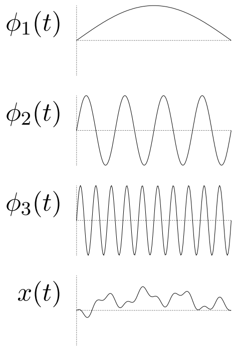


Figure 11.1. Three basis functions, and a weighted combination of them, $x(t) = \sum_{n=1}^{N} x_n \phi_n(t)$, with $x_1 = 0.4$, $x_2 = -0.2$, and $x_3 = 0.1$.

Before returning to the Gaussian channel, we define the *bandwidth* (measured in Hertz) of the continuous channel to be:

$$W = \frac{N^{\max}}{2T}, \qquad (11.12)$$

where $N^{\max}$ is the maximum number of orthonormal functions that can be produced in an interval of length T. This definition can be motivated by imagining creating a band-limited signal of duration T from orthonormal cosine and sine curves of maximum frequency W. The number of orthonormal functions is $N^{\max} = 2WT$. This definition relates to the Nyquist sampling theorem: if the highest frequency present in a signal is W, then the signal can be fully determined from its values at a series of discrete sample points separated by the Nyquist interval $\Delta t = {}^1\!/_{2W}$ seconds.

So the use of a real continuous channel with bandwidth W, noise spectral density N_0, and power P is equivalent to $N/T = 2W$ uses per second of a Gaussian channel with noise level $\sigma^2 = N_0/2$ and subject to the signal power constraint $\overline{x_n^2} \leq {}^P\!/_{2W}$.

Definition of $E_{\rm b}/N_0$

Imagine that the Gaussian channel $y_n = x_n + n_n$ is used with an encoding system to transmit *binary* source bits at a rate of R bits per channel use. How can we compare two encoding systems that have different rates of communication R and that use different powers $\overline{x_n^2}$? Transmitting at a large rate R is good; using small power is good too.

It is conventional to measure the rate-compensated signal-to-noise ratio by the ratio of the power per source bit $E_{\rm b} = \overline{x_n^2}/R$ to the noise spectral density N_0:

$$E_{\rm b}/N_0 = \frac{\overline{x_n^2}}{2\sigma^2 R}. \qquad (11.13)$$

$E_{\rm b}/N_0$ is dimensionless, but it is usually reported in the units of decibels; the value given is $10 \log_{10} E_{\rm b}/N_0$.

$E_{\rm b}/N_0$ is one of the measures used to compare coding schemes for Gaussian channels.

▶ 11.2 Inferring the input to a real channel

'The best detection of pulses'

In 1944 Shannon wrote a memorandum (Shannon, 1993) on the problem of best differentiating between two types of pulses of known shape, represented by vectors $\mathbf{x}_0$ and $\mathbf{x}_1$, given that one of them has been transmitted over a noisy channel. This is a pattern recognition problem. It is assumed that the noise is Gaussian with probability density

$$P(\mathbf{n}) = \left[\det\left(\frac{\mathbf{A}}{2\pi}\right)\right]^{1/2} \exp\left(-\frac{1}{2}\mathbf{n}^{\mathsf{T}}\mathbf{A}\mathbf{n}\right), \tag{11.14}$$

where $\mathbf{A}$ is the inverse of the variance–covariance matrix of the noise, a symmetric and positive-definite matrix. (If $\mathbf{A}$ is a multiple of the identity matrix, $\mathbf{I}/\sigma^2$, then the noise is 'white'. For more general $\mathbf{A}$, the noise is 'coloured'.) The probability of the received vector $\mathbf{y}$ given that the source signal was s (either zero or one) is then

$$P(\mathbf{y}\,|\,s) = \left[\det\left(\frac{\mathbf{A}}{2\pi}\right)\right]^{1/2} \exp\left(-\frac{1}{2}(\mathbf{y}-\mathbf{x}_s)^{\mathsf{T}}\mathbf{A}(\mathbf{y}-\mathbf{x}_s)\right). \tag{11.15}$$

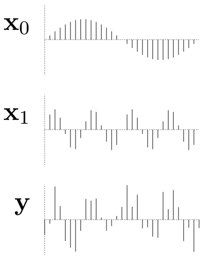


Figure 11.2. Two pulses $\mathbf{x}_0$ and $\mathbf{x}_1$, represented as 31-dimensional vectors, and a noisy version of one of them, $\mathbf{y}$.

The optimal detector is based on the posterior probability ratio:

$$\begin{aligned}
\frac{P(s=1\,|\,\mathbf{y})}{P(s=0\,|\,\mathbf{y})} &= \frac{P(\mathbf{y}\,|\,s=1)}{P(\mathbf{y}\,|\,s=0)}\frac{P(s=1)}{P(s=0)} && (11.16)\\
&= \exp\left(-\frac{1}{2}(\mathbf{y}-\mathbf{x}_1)^{\mathsf{T}}\mathbf{A}(\mathbf{y}-\mathbf{x}_1) + \frac{1}{2}(\mathbf{y}-\mathbf{x}_0)^{\mathsf{T}}\mathbf{A}(\mathbf{y}-\mathbf{x}_0) + \ln\frac{P(s=1)}{P(s=0)}\right)\\
&= \exp\left(\mathbf{y}^{\mathsf{T}}\mathbf{A}(\mathbf{x}_1-\mathbf{x}_0)+\theta\right), && (11.17)
\end{aligned}$$

where θ is a constant independent of the received vector $\mathbf{y}$,

$$\theta = -\frac{1}{2}\mathbf{x}_1^{\mathsf{T}}\mathbf{A}\mathbf{x}_1 + \frac{1}{2}\mathbf{x}_0^{\mathsf{T}}\mathbf{A}\mathbf{x}_0 + \ln\frac{P(s=1)}{P(s=0)}. \tag{11.18}$$

If the detector is forced to make a decision (i.e., guess either $s=1$ or $s=0$) then the decision that minimizes the probability of error is to guess the most probable hypothesis. We can write the optimal decision in terms of a *discriminant function*:

$$a(\mathbf{y}) \equiv \mathbf{y}^{\mathsf{T}}\mathbf{A}(\mathbf{x}_1-\mathbf{x}_0)+\theta \tag{11.19}$$

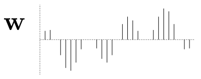


Figure 11.3. The weight vector $\mathbf{w} \propto \mathbf{x}_1 - \mathbf{x}_0$ that is used to discriminate between $\mathbf{x}_0$ and $\mathbf{x}_1$.

with the decisions

$$\begin{array}{lcl}
a(\mathbf{y}) > 0 & \rightarrow & \text{guess } s=1\\
a(\mathbf{y}) < 0 & \rightarrow & \text{guess } s=0\\
a(\mathbf{y}) = 0 & \rightarrow & \text{guess either.}
\end{array} \tag{11.20}$$

Notice that $a(\mathbf{y})$ is a linear function of the received vector,

$$a(\mathbf{y}) = \mathbf{w}^{\mathsf{T}}\mathbf{y}+\theta, \tag{11.21}$$

where $\mathbf{w} \equiv \mathbf{A}(\mathbf{x}_1-\mathbf{x}_0)$.

▶ 11.3 Capacity of Gaussian channel

Until now we have measured the joint, marginal, and conditional entropy of discrete variables only. In order to define the information conveyed by continuous variables, there are two issues we must address – the infinite length of the real line, and the infinite precision of real numbers.

Infinite inputs

How much information can we convey in one use of a Gaussian channel? If we are allowed to put *any* real number x into the Gaussian channel, we could communicate an enormous string of N digits $d_1 d_2 d_3 \ldots d_N$ by setting $x = d_1 d_2 d_3 \ldots d_N 000 \ldots 000$. The amount of error-free information conveyed in just a single transmission could be made arbitrarily large by increasing N, and the communication could be made arbitrarily reliable by increasing the number of zeroes at the end of x. There is usually some power cost associated with large inputs, however, not to mention practical limits in the dynamic range acceptable to a receiver. It is therefore conventional to introduce a *cost function* $v(x)$ for every input x, and constrain codes to have an average cost $\bar{v}$ less than or equal to some maximum value. A generalized channel coding theorem, including a cost function for the inputs, can be proved – see McEliece (1977). The result is a channel capacity $C(\bar{v})$ that is a function of the permitted cost. For the Gaussian channel we will assume a cost

$$v(x) = x^2 \tag{11.22}$$

such that the 'average power' $\overline{x^2}$ of the input is constrained. We motivated this cost function above in the case of real electrical channels in which the physical power consumption is indeed quadratic in x. The constraint $\overline{x^2} = \bar{v}$ makes it impossible to communicate infinite information in one use of the Gaussian channel.

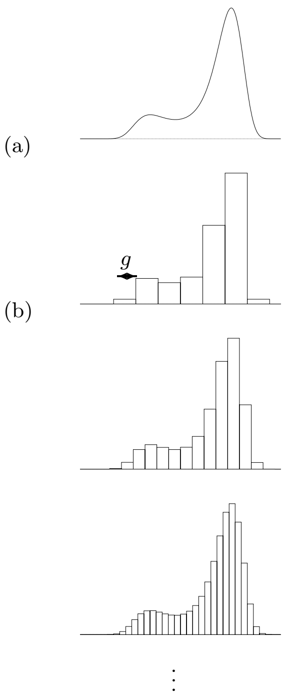


Figure 11.4. (a) A probability density $P(x)$. Question: can we define the 'entropy' of this density? (b) We could evaluate the entropies of a sequence of probability distributions with decreasing grain-size g, but these entropies tend to $\int P(x) \log \frac{1}{P(x)g} \, \mathrm{d}x$, which is not independent of g: the entropy goes up by one bit for every halving of g. $\int P(x) \log \frac{1}{P(x)} \, \mathrm{d}x$ is an illegal integral.

Infinite precision

It is tempting to define joint, marginal, and conditional entropies for real variables simply by replacing summations by integrals, but this is not a well defined operation. As we discretize an interval into smaller and smaller divisions, the entropy of the discrete distribution diverges (as the logarithm of the granularity) (figure 11.4). Also, it is not permissible to take the logarithm of a dimensional quantity such as a probability density $P(x)$ (whose dimensions are $[x]^{-1}$).

There is one information measure, however, that has a well-behaved limit, namely the mutual information – and this is the one that really matters, since it measures how much information one variable conveys about another. In the discrete case,

$$I(X;Y) = \sum_{x,y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)}. \tag{11.23}$$

Now because the argument of the log is a ratio of two probabilities over the same space, it is OK to have $P(x,y)$, $P(x)$ and $P(y)$ be probability densities and replace the sum by an integral:

$$\begin{aligned} I(X;Y) &= \int \mathrm{d}x \, \mathrm{d}y \, P(x,y) \log \frac{P(x,y)}{P(x)P(y)} && (11.24) \\ &= \int \mathrm{d}x \, \mathrm{d}y \, P(x) P(y \,|\, x) \log \frac{P(y \,|\, x)}{P(y)}. && (11.25) \end{aligned}$$

We can now ask these questions for the Gaussian channel: (a) what probability distribution $P(x)$ maximizes the mutual information (subject to the constraint $\overline{x^2} = v$)? and (b) does the maximal mutual information still measure the maximum error-free communication rate of this real channel, as it did for the discrete channel?

Exercise 11.1.[*3*, p.189] Prove that the probability distribution $P(x)$ that maximizes the mutual information (subject to the constraint $\overline{x^2} = v$) is a Gaussian distribution of mean zero and variance v.

▷ Exercise 11.2.[*2*, p.189] Show that the mutual information $I(X;Y)$, in the case of this optimized distribution, is

$$C = \frac{1}{2}\log\left(1+\frac{v}{\sigma^2}\right). \tag{11.26}$$

This is an important result. We see that the capacity of the Gaussian channel is a function of the *signal-to-noise ratio* v/σ^2.

Inferences given a Gaussian input distribution

If $P(x) = \text{Normal}(x;0,v)$ and $P(y\,|\,x) = \text{Normal}(y;x,\sigma^2)$ then the marginal distribution of y is $P(y) = \text{Normal}(y;0,v{+}\sigma^2)$ and the posterior distribution of the input, given that the output is y, is:

$$\begin{aligned}
P(x\,|\,y) &\propto P(y\,|\,x)P(x) && (11.27)\\
&\propto \exp(-(y-x)^2/2\sigma^2)\exp(-x^2/2v) && (11.28)\\
&= \text{Normal}\left(x;\frac{v}{v+\sigma^2}\,y\,,\,\left(\frac{1}{v}+\frac{1}{\sigma^2}\right)^{-1}\right). && (11.29)
\end{aligned}$$

[The step from (11.28) to (11.29) is made by completing the square in the exponent.] This formula deserves careful study. The mean of the posterior distribution, $\frac{v}{v+\sigma^2}\,y$, can be viewed as a weighted combination of the value that best fits the output, $x = y$, and the value that best fits the prior, $x = 0$:

$$\frac{v}{v+\sigma^2}\,y = \frac{1/\sigma^2}{1/v+1/\sigma^2}\,y + \frac{1/v}{1/v+1/\sigma^2}\,0. \tag{11.30}$$

The weights $1/\sigma^2$ and $1/v$ are the *precisions* of the two Gaussians that we multiplied together in equation (11.28): the prior and the likelihood.

The precision of the posterior distribution is the sum of these two precisions. This is a general property: whenever two independent sources contribute information, via Gaussian distributions, about an unknown variable, the precisions add. [This is the dual to the better-known relationship 'when independent variables are added, their variances add'.]

Noisy-channel coding theorem for the Gaussian channel

We have evaluated a maximal mutual information. Does it correspond to a maximum possible rate of error-free information transmission? One way of proving that this is so is to define a sequence of discrete channels, all derived from the Gaussian channel, with increasing numbers of inputs and outputs, and prove that the maximum mutual information of these channels tends to the asserted C. The noisy-channel coding theorem for discrete channels applies to each of these derived channels, thus we obtain a coding theorem for the continuous channel. Alternatively, we can make an intuitive argument for the coding theorem specific for the Gaussian channel.

Geometrical view of the noisy-channel coding theorem: sphere packing

Consider a sequence $\mathbf{x} = (x_1, \ldots, x_N)$ of inputs, and the corresponding output $\mathbf{y}$, as defining two points in an N dimensional space. For large N, the noise power is very likely to be close (fractionally) to $N\sigma^2$. The output $\mathbf{y}$ is therefore very likely to be close to the surface of a sphere of radius $\sqrt{N\sigma^2}$ centred on $\mathbf{x}$. Similarly, if the original signal $\mathbf{x}$ is generated at random subject to an average power constraint $\overline{x^2} = v$, then $\mathbf{x}$ is likely to lie close to a sphere, centred on the origin, of radius $\sqrt{Nv}$; and because the total average power of $\mathbf{y}$ is $v + \sigma^2$, the received signal $\mathbf{y}$ is likely to lie on the surface of a sphere of radius $\sqrt{N(v + \sigma^2)}$, centred on the origin.

The volume of an N-dimensional sphere of radius r is

$$V(r, N) = \tfrac{\pi^{N/2}}{\Gamma(N/2+1)} r^N. \qquad (11.31)$$

Now consider making a communication system based on non-confusable inputs $\mathbf{x}$, that is, inputs whose spheres do not overlap significantly. The maximum number S of non-confusable inputs is given by dividing the volume of the sphere of probable $\mathbf{y}$s by the volume of the sphere for $\mathbf{y}$ given $\mathbf{x}$:

$$S \leq \left(\frac{\sqrt{N(v + \sigma^2)}}{\sqrt{N\sigma^2}} \right)^N \qquad (11.32)$$

Thus the capacity is bounded by:

$$C = \frac{1}{N} \log M \leq \frac{1}{2} \log \left(1 + \frac{v}{\sigma^2} \right). \qquad (11.33)$$

A more detailed argument like the one used in the previous chapter can establish equality.

Back to the continuous channel

Recall that the use of a real continuous channel with bandwidth W, noise spectral density N_0 and power P is equivalent to $N/T = 2W$ uses per second of a Gaussian channel with $\sigma^2 = N_0/2$ and subject to the constraint $\overline{x_n^2} \leq P/2W$. Substituting the result for the capacity of the Gaussian channel, we find the capacity of the continuous channel to be:

$$C = W \log \left(1 + \frac{P}{N_0 W} \right) \text{ bits per second.} \qquad (11.34)$$

This formula gives insight into the tradeoffs of practical communication. Imagine that we have a fixed power constraint. What is the best bandwidth to make use of that power? Introducing $W_0 = P/N_0$, i.e., the bandwidth for which the signal-to-noise ratio is 1, figure 11.5 shows $C/W_0 = W/W_0 \log(1 + W_0/W)$ as a function of W/W_0. The capacity increases to an asymptote of $W_0 \log e$. It is dramatically better (in terms of capacity for fixed power) to transmit at a low signal-to-noise ratio over a large bandwidth, than with high signal-to-noise in a narrow bandwidth; this is one motivation for wideband communication methods such as the 'direct sequence spread-spectrum' approach used in 3G mobile phones. Of course, you are not alone, and your electromagnetic neighbours may not be pleased if you use a large bandwidth, so for social reasons, engineers often have to make do with higher-power, narrow-bandwidth transmitters.

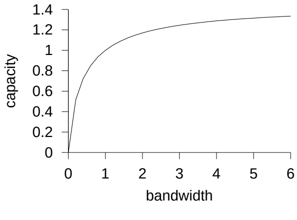


Figure 11.5. Capacity versus bandwidth for a real channel: $C/W_0 = W/W_0 \log(1 + W_0/W)$ as a function of W/W_0.

▶ 11.4 What are the capabilities of practical error-correcting codes?

Nearly all codes are good, but nearly all codes require exponential look-up tables for practical implementation of the encoder and decoder – exponential in the blocklength N. And the coding theorem required N to be large.

By a *practical* error-correcting code, we mean one that can be encoded and decoded in a reasonable amount of time, for example, a time that scales as a polynomial function of the blocklength N – preferably linearly.

The Shannon limit is not achieved in practice

The non-constructive proof of the noisy-channel coding theorem showed that good block codes exist for any noisy channel, and indeed that nearly all block codes are good. But writing down an explicit and practical encoder and decoder that are as good as promised by Shannon is still an unsolved problem.

Very good codes. Given a channel, a family of block codes that achieve arbitrarily small probability of error at any communication rate up to the capacity of the channel are called 'very good' codes for that channel.

Good codes are code families that achieve arbitrarily small probability of error at non-zero communication rates up to some maximum rate that may be *less than* the capacity of the given channel.

Bad codes are code families that cannot achieve arbitrarily small probability of error, or that can achieve arbitrarily small probability of error only by decreasing the information rate to zero. Repetition codes are an example of a bad code family. (Bad codes are not necessarily useless for practical purposes.)

Practical codes are code families that can be encoded and decoded in time and space polynomial in the blocklength.

Most established codes are linear codes

Let us review the definition of a block code, and then add the definition of a linear block code.

An (N, K) **block code** for a channel Q is a list of $S = 2^K$ codewords $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(2^K)}\}$, each of length N: $\mathbf{x}^{(s)} \in \mathcal{A}_X^N$. The signal to be encoded, s, which comes from an alphabet of size 2^K, is encoded as $\mathbf{x}^{(s)}$.

A linear (N, K) **block code** is a block code in which the codewords $\{\mathbf{x}^{(s)}\}$ make up a K-dimensional subspace of $\mathcal{A}_X^N$. The encoding operation can be represented by an $N \times K$ binary matrix $\mathbf{G}^{\mathsf{T}}$ such that if the signal to be encoded, in binary notation, is $\mathbf{s}$ (a vector of length K bits), then the encoded signal is $\mathbf{t} = \mathbf{G}^{\mathsf{T}}\mathbf{s}$ modulo 2.

The codewords $\{\mathbf{t}\}$ can be defined as the set of vectors satisfying $\mathbf{Ht} = \mathbf{0} \bmod 2$, where $\mathbf{H}$ is the *parity-check matrix* of the code.

For example the $(7, 4)$ Hamming code of section 1.2 takes $K = 4$ signal bits, $\mathbf{s}$, and transmits them followed by three parity-check bits. The $N = 7$ transmitted symbols are given by $\mathbf{G}^{\mathsf{T}}\mathbf{s} \bmod 2$.

Coding theory was born with the work of Hamming, who invented a family of practical error-correcting codes, each able to correct one error in a block of length N, of which the repetition code R_3 and the $(7, 4)$ code are

$$\mathbf{G}^{\mathsf{T}} = \begin{bmatrix} 1 & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & 1 \\ 1 & 1 & 1 & \cdot \\ \cdot & 1 & 1 & 1 \\ 1 & \cdot & 1 & 1 \end{bmatrix}$$

the simplest. Since then most established codes have been generalizations of Hamming's codes: Bose–Chaudhury–Hocquenhem codes, Reed–Müller codes, Reed–Solomon codes, and Goppa codes, to name a few.

Convolutional codes

Another family of linear codes are *convolutional codes*, which do not divide the source stream into blocks, but instead read and transmit bits continuously. The transmitted bits are a linear function of the past source bits. Usually the rule for generating the transmitted bits involves feeding the present source bit into a linear-feedback shift-register of length k, and transmitting one or more linear functions of the state of the shift register at each iteration. The resulting transmitted bit stream is the convolution of the source stream with a linear filter. The impulse-response function of this filter may have finite or infinite duration, depending on the choice of feedback shift-register.

We will discuss convolutional codes in Chapter 48.

Are linear codes 'good'?

One might ask, is the reason that the Shannon limit is not achieved in practice because linear codes are inherently not as good as random codes? The answer is no, the noisy-channel coding theorem can still be proved for linear codes, at least for some channels (see Chapter 14), though the proofs, like Shannon's proof for random codes, are non-constructive.

Linear codes are easy to implement at the encoding end. Is decoding a linear code also easy? Not necessarily. The general decoding problem (find the maximum likelihood $\mathbf{s}$ in the equation $\mathbf{G}^{\mathsf{T}}\mathbf{s}+\mathbf{n}=\mathbf{r}$) is in fact NP-complete (Berlekamp *et al.*, 1978). [NP-complete problems are computational problems that are all equally difficult and which are widely believed to require exponential computer time to solve in general.] So attention focuses on families of codes for which there is a fast decoding algorithm.

Concatenation

One trick for building codes with practical decoders is the idea of concatenation.

$$\mathcal{C}' \rightarrow \underbrace{\mathcal{C} \rightarrow Q \rightarrow \mathcal{D}}_{Q'} \rightarrow \mathcal{D}'$$

An encoder–channel–decoder system $\mathcal{C} \rightarrow Q \rightarrow \mathcal{D}$ can be viewed as defining a super-channel Q' with a smaller probability of error, and with complex correlations among its errors. We can create an encoder $\mathcal{C}'$ and decoder $\mathcal{D}'$ for this super-channel Q'. The code consisting of the outer code $\mathcal{C}'$ followed by the inner code $\mathcal{C}$ is known as a *concatenated code*.

Some concatenated codes make use of the idea of *interleaving*. We read the data in blocks, the size of each block being larger than the blocklengths of the constituent codes $\mathcal{C}$ and $\mathcal{C}'$. After encoding the data of one block using code $\mathcal{C}'$, the bits are reordered within the block in such a way that nearby bits are separated from each other once the block is fed to the second code $\mathcal{C}$. A simple example of an interleaver is a *rectangular code* or *product code* in which the data are arranged in a $K_2 \times K_1$ block, and encoded horizontally using an (N_1, K_1) linear code, then vertically using a (N_2, K_2) linear code.

▷ Exercise 11.3.[3] Show that either of the two codes can be viewed as the inner code or the outer code.

As an example, figure 11.6 shows a product code in which we encode first with the repetition code R_3 (also known as the Hamming code $H(3,1)$)

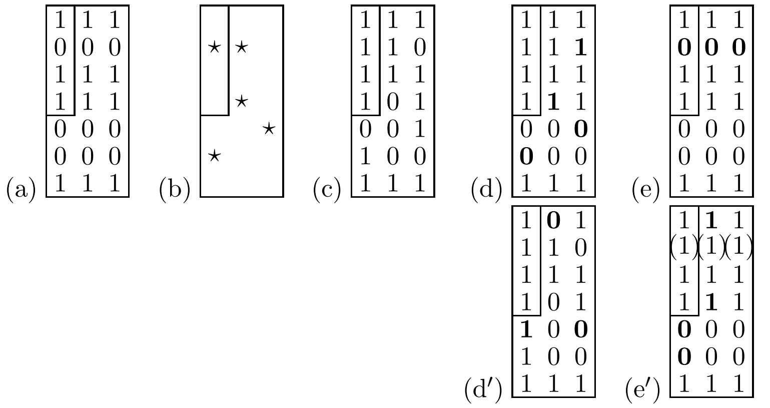


Figure 11.6. A product code. (a) A string 1011 encoded using a concatenated code consisting of two Hamming codes, $H(3,1)$ and $H(7,4)$. (b) a noise pattern that flips 5 bits. (c) The received vector. (d) After decoding using the horizontal $(3,1)$ decoder, and (e) after subsequently using the vertical $(7,4)$ decoder. The decoded vector matches the original.
(d′, e′) After decoding in the other order, three errors still remain.

horizontally then with $H(7,4)$ vertically. The blocklength of the concatenated code is 27. The number of source bits per codeword is four, shown by the small rectangle.

We can decode conveniently (though not optimally) by using the individual decoders for each of the subcodes in some sequence. It makes most sense to first decode the code which has the lowest rate and hence the greatest error-correcting ability.

Figure 11.6(c–e) shows what happens if we receive the codeword of figure 11.6a with some errors (five bits flipped, as shown) and apply the decoder for $H(3,1)$ first, and then the decoder for $H(7,4)$. The first decoder corrects three of the errors, but erroneously modifies the third bit in the second row where there are two bit errors. The $(7,4)$ decoder can then correct all three of these errors.

Figure 11.6(d′–e′) shows what happens if we decode the two codes in the other order. In columns one and two there are two errors, so the $(7,4)$ decoder introduces two extra errors. It corrects the one error in column 3. The $(3,1)$ decoder then cleans up four of the errors, but erroneously infers the second bit.

Interleaving

The motivation for interleaving is that by spreading out bits that are nearby in one code, we make it possible to ignore the complex correlations among the errors that are produced by the inner code. Maybe the inner code will mess up an entire codeword; but that codeword is spread out one bit at a time over several codewords of the outer code. So we can treat the errors introduced by the inner code as if they are independent.

Other channel models

In addition to the binary symmetric channel and the Gaussian channel, coding theorists keep more complex channels in mind also.

Burst-error channels are important models in practice. Reed–Solomon codes use Galois fields (see Appendix C.1) with large numbers of elements (e.g. 2^{16}) as their input alphabets, and thereby automatically achieve a degree of burst-error tolerance in that even if 17 successive bits are corrupted, only 2 successive symbols in the Galois field representation are corrupted. Concatenation and interleaving can give further protection against burst errors. The concatenated Reed–Solomon codes used on digital compact discs are able to correct bursts of errors of length 4000 bits.

▷ Exercise 11.4.[2, p.189] The technique of interleaving, which allows bursts of errors to be treated as independent, is widely used, but is theoretically a poor way to protect data against burst errors, in terms of the amount of redundancy required. Explain why interleaving is a poor method, using the following burst-error channel as an example. Time is divided into chunks of length $N = 100$ clock cycles; during each chunk, there is a burst with probability $b = 0.2$; during a burst, the channel is a binary symmetric channel with $f = 0.5$. If there is no burst, the channel is an error-free binary channel. Compute the capacity of this channel and compare it with the maximum communication rate that could conceivably be achieved if one used interleaving and treated the errors as independent.

Fading channels are real channels like Gaussian channels except that the received power is assumed to vary with time. A moving mobile phone is an important example. The incoming radio signal is reflected off nearby objects so that there are interference patterns and the intensity of the signal received by the phone varies with its location. The received power can easily vary by 10 decibels (a factor of ten) as the phone's antenna moves through a distance similar to the wavelength of the radio signal (a few centimetres).

▶ 11.5 The state of the art

What are the best known codes for communicating over Gaussian channels? All the practical codes are linear codes, and are either based on convolutional codes or block codes.

Convolutional codes, and codes based on them

Textbook convolutional codes. The 'de facto standard' error-correcting code for satellite communications is a convolutional code with constraint length 7. Convolutional codes are discussed in Chapter 48.

Concatenated convolutional codes. The above convolutional code can be used as the inner code of a concatenated code whose outer code is a Reed–Solomon code with eight-bit symbols. This code was used in deep space communication systems such as the Voyager spacecraft. For further reading about Reed–Solomon codes, see Lin and Costello (1983).

The code for Galileo. A code using the same format but using a longer constraint length – 15 – for its convolutional code and a larger Reed–Solomon code was developed by the Jet Propulsion Laboratory (Swanson, 1988). The details of this code are unpublished outside JPL, and the decoding is only possible using a room full of special-purpose hardware. In 1992, this was the best code known of rate 1/4.

Turbo codes. In 1993, Berrou, Glavieux and Thitimajshima reported work on *turbo codes*. The encoder of a turbo code is based on the encoders of two convolutional codes. The source bits are fed into each encoder, the order of the source bits being permuted in a random way, and the resulting parity bits from each constituent code are transmitted.

The decoding algorithm involves iteratively decoding each constituent code using its standard decoding algorithm, then using the output of the decoder as the input to the other decoder. This decoding algorithm

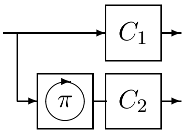


Figure 11.7. The encoder of a turbo code. Each box C_1, C_2, contains a convolutional code. The source bits are reordered using a permutation π before they are fed to C_2. The transmitted codeword is obtained by concatenating or interleaving the outputs of the two convolutional codes. The random permutation is chosen when the code is designed, and fixed thereafter.

is an instance of a *message-passing* algorithm called the *sum–product algorithm*.

Turbo codes are discussed in Chapter 48, and message passing in Chapters 16, 17, 25, and 26.

Block codes

Gallager's low-density parity-check codes. The best block codes known for Gaussian channels were invented by Gallager in 1962 but were promptly forgotten by most of the coding theory community. They were rediscovered in 1995 and shown to have outstanding theoretical and practical properties. Like turbo codes, they are decoded by message-passing algorithms.

We will discuss these beautifully simple codes in Chapter 47.

The performances of the above codes are compared for Gaussian channels in figure 47.17, p.568.

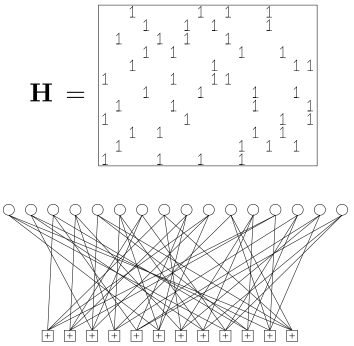


Figure 11.8. A low-density parity-check matrix and the corresponding graph of a rate-1/4 low-density parity-check code with blocklength $N = 16$, and $M = 12$ constraints. Each white circle represents a transmitted bit. Each bit participates in $j = 3$ constraints, represented by ⊞ squares. Each constraint forces the sum of the $k = 4$ bits to which it is connected to be even. This code is a $(16, 4)$ code. Outstanding performance is obtained when the blocklength is increased to $N \simeq 10\,000$.

▶ 11.6 Summary

Random codes are good, but they require exponential resources to encode and decode them.

Non-random codes tend for the most part not to be as good as random codes. For a non-random code, encoding may be easy, but even for simply-defined linear codes, the decoding problem remains very difficult.

The best practical codes (a) employ very large block sizes; (b) are based on semi-random code constructions; and (c) make use of probability-based decoding algorithms.

▶ 11.7 Nonlinear codes

Most practically used codes are linear, but not all. Digital soundtracks are encoded onto cinema film as a binary pattern. The likely errors affecting the film involve dirt and scratches, which produce large numbers of 1s and 0s respectively. We want none of the codewords to look like all-1s or all-0s, so that it will be easy to detect errors caused by dirt and scratches. One of the codes used in digital cinema sound systems is a nonlinear $(8, 6)$ code consisting of 64 of the $\binom{8}{4}$ binary patterns of weight 4.

▶ 11.8 Errors other than noise

Another source of uncertainty for the receiver is uncertainty about the *timing* of the transmitted signal $x(t)$. In ordinary coding theory and information theory, the transmitter's time t and the receiver's time u are assumed to be perfectly synchronized. But if the receiver receives a signal $y(u)$, where the receiver's time, u, is an imperfectly known function $u(t)$ of the transmitter's time t, then the capacity of this channel for communication is reduced. The theory of such channels is incomplete, compared with the synchronized channels we have discussed thus far. Not even the *capacity* of channels with synchronization errors is known (Levenshtein, 1966; Ferreira *et al.*, 1997); codes for reliable communication over channels with synchronization errors remain an active research area (Davey and MacKay, 2001).

Further reading

For a review of the history of spread-spectrum methods, see Scholtz (1982).

▶ 11.9 Exercises

The Gaussian channel

▷ Exercise 11.5.$^{[2, \text{ p.190}]}$ Consider a Gaussian channel with a real input x, and signal to noise ratio v/σ^2.

(a) What is its capacity C?

(b) If the input is constrained to be binary, $x \in \{\pm\sqrt{v}\}$, what is the capacity C' of this constrained channel?

(c) If in addition the output of the channel is thresholded using the mapping

$$y \rightarrow y' = \begin{cases} 1 & y > 0 \\ 0 & y \leq 0, \end{cases} \tag{11.35}$$

what is the capacity C'' of the resulting channel?

(d) Plot the three capacities above as a function of v/σ^2 from 0.1 to 2. [You'll need to do a numerical integral to evaluate C'.]

▷ Exercise 11.6.$^{[3]}$ For large integers K and N, what fraction of all binary error-correcting codes of length N and rate $R = K/N$ are *linear* codes? [The answer will depend on whether you choose to define the code to be an *ordered* list of 2^K codewords, that is, a mapping from $s \in \{1, 2, \ldots, 2^K\}$ to $\mathbf{x}^{(s)}$, or to define the code to be an unordered list, so that two codes consisting of the same codewords are identical. Use the latter definition: a code is a set of codewords; how the encoder operates is not part of the definition of the code.]

Erasure channels

▷ Exercise 11.7.$^{[4]}$ Design a code for the binary erasure channel, and a decoding algorithm, and evaluate their probability of error. [The design of good codes for erasure channels is an active research area (Spielman, 1996; Byers *et al.*, 1998); see also Chapter 50.]

▷ Exercise 11.8.$^{[5]}$ Design a code for the q-ary erasure channel, whose input x is drawn from $0, 1, 2, 3, \ldots, (q-1)$, and whose output y is equal to x with probability $(1-f)$ and equal to `?` otherwise. [This erasure channel is a good model for packets transmitted over the internet, which are either received reliably or are lost.]

Exercise 11.9.$^{[3, \text{ p.190}]}$ How do redundant arrays of independent disks (RAID) work? These are information storage systems consisting of about ten disk drives, of which any two or three can be disabled and the others are able to still able to reconstruct any requested file. What codes are used, and how far are these systems from the Shannon limit for the problem they are solving? How would *you* design a better RAID system? Some information is provided in the solution section. See `http://www.acnc.com/raid2.html`; see also Chapter 50.

[Some people say RAID stands for 'redundant array of inexpensive disks', but I think that's silly – RAID would still be a good idea even if the disks were expensive!]

▶ 11.10 Solutions

Solution to exercise 11.1 (p.181). Introduce a Lagrange multiplier λ for the power constraint and another, μ, for the constraint of normalization of $P(x)$.

$$\begin{aligned} F &= I(X;Y) - \lambda \textstyle\int \mathrm{d}x\, P(x)x^2 - \mu \textstyle\int \mathrm{d}x\, P(x) & (11.36) \\ &= \int \mathrm{d}x\, P(x) \left[\int \mathrm{d}y\, P(y \,|\, x) \ln \frac{P(y \,|\, x)}{P(y)} - \lambda x^2 - \mu \right]. & (11.37) \end{aligned}$$

Make the functional derivative with respect to $P(x^*)$.

$$\begin{aligned} \frac{\delta F}{\delta P(x^*)} = & \int \mathrm{d}y\, P(y \,|\, x^*) \ln \frac{P(y \,|\, x^*)}{P(y)} - \lambda x^{*2} - \mu \\ & - \int \mathrm{d}x\, P(x) \int \mathrm{d}y\, P(y \,|\, x) \frac{1}{P(y)} \frac{\delta P(y)}{\delta P(x^*)}. & (11.38) \end{aligned}$$

The final factor $\delta P(y)/\delta P(x^*)$ is found, using $P(y) = \int \mathrm{d}x\, P(x) P(y \,|\, x)$, to be $P(y \,|\, x^*)$, and the whole of the last term collapses in a puff of smoke to 1, which can be absorbed into the μ term.

Substitute $P(y \,|\, x) = \exp(-(y-x)^2/2\sigma^2)/\sqrt{2\pi\sigma^2}$ and set the derivative to zero:

$$\int \mathrm{d}y\, P(y \,|\, x) \ln \frac{P(y \,|\, x)}{P(y)} - \lambda x^2 - \mu' = 0 \qquad (11.39)$$

$$\Rightarrow \int \mathrm{d}y\, \frac{\exp(-(y-x)^2/2\sigma^2)}{\sqrt{2\pi\sigma^2}} \ln\left[P(y)\sigma\right] = -\lambda x^2 - \mu' - \frac{1}{2}. \qquad (11.40)$$

This condition must be satisfied by $\ln[P(y)\sigma]$ for all x.

Writing a Taylor expansion of $\ln[P(y)\sigma] = a + by + cy^2 + \cdots$, only a quadratic function $\ln[P(y)\sigma] = a + cy^2$ would satisfy the constraint (11.40). (Any higher order terms y^p, $p > 2$, would produce terms in x^p that are not present on the right-hand side.) Therefore $P(y)$ is Gaussian. We can obtain this optimal output distribution by using a Gaussian input distribution $P(x)$.

Solution to exercise 11.2 (p.181). Given a Gaussian input distribution of variance v, the output distribution is $\mathrm{Normal}(0, v + \sigma^2)$, since x and the noise are independent random variables, and variances add for independent random variables. The mutual information is:

$$\begin{aligned} I(X;Y) &= \int \mathrm{d}x\, \mathrm{d}y\, P(x) P(y \,|\, x) \log P(y \,|\, x) - \int \mathrm{d}y\, P(y) \log P(y) & (11.41) \\ &= \frac{1}{2} \log \frac{1}{\sigma^2} - \frac{1}{2} \log \frac{1}{v + \sigma^2} & (11.42) \\ &= \frac{1}{2} \log \left(1 + \frac{v}{\sigma^2}\right). & (11.43) \end{aligned}$$

Solution to exercise 11.4 (p.186). The capacity of the channel is one minus the information content of the noise that it adds. That information content is, per chunk, the entropy of the selection of whether the chunk is bursty, $H_2(b)$, plus, with probability b, the entropy of the flipped bits, N, which adds up to $H_2(b) + Nb$ per chunk (roughly; accurate if N is large). So, per bit, the capacity is, for $N = 100$,

$$C = 1 - \left(\frac{1}{N} H_2(b) + b\right) = 1 - 0.207 = 0.793. \qquad (11.44)$$

In contrast, interleaving, which treats bursts of errors as independent, causes the channel to be treated as a binary symmetric channel with $f = 0.2 \times 0.5 = 0.1$, whose capacity is about 0.53.

Interleaving throws away the useful information about the correlatedness of the errors. Theoretically, we should be able to communicate about $(0.79/0.53) \simeq 1.6$ times faster using a code and decoder that explicitly treat bursts as bursts.

Solution to exercise 11.5 (p.188).

(a) Putting together the results of exercises 11.1 and 11.2, we deduce that a Gaussian channel with real input x, and signal to noise ratio v/σ^2 has capacity

$$C = \frac{1}{2} \log \left(1 + \frac{v}{\sigma^2}\right). \tag{11.45}$$

(b) If the input is constrained to be binary, $x \in \{\pm\sqrt{v}\}$, the capacity is achieved by using these two inputs with equal probability. The capacity is reduced to a somewhat messy integral,

$$C'' = \int_{-\infty}^{\infty} \mathrm{d}y \, N(y;0) \log N(y;0) - \int_{-\infty}^{\infty} \mathrm{d}y \, P(y) \log P(y), \tag{11.46}$$

where $N(y;x) \equiv (1/\sqrt{2\pi}) \exp[(y-x)^2/2]$, $x \equiv \sqrt{v}/\sigma$, and $P(y) \equiv [N(y;x) + N(y;-x)]/2$. This capacity is smaller than the unconstrained capacity (11.45), but for small signal-to-noise ratio, the two capacities are close in value.

(c) If the output is thresholded, then the Gaussian channel is turned into a binary symmetric channel whose transition probability is given by the error function Φ defined on page 156. The capacity is

$$C'' = 1 - H_2(f), \text{ where } f = \Phi(\sqrt{v}/\sigma). \tag{11.47}$$

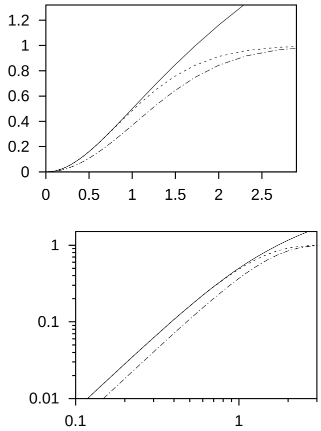


Figure 11.9. Capacities (from top to bottom in each graph) C, C', and C''', versus the signal-to-noise ratio $(\sqrt{v}/\sigma)$. The lower graph is a log–log plot.

Solution to exercise 11.9 (p.188). There are several RAID systems. One of the easiest to understand consists of 7 disk drives which store data at rate 4/7 using a $(7,4)$ Hamming code: each successive four bits are encoded with the code and the seven codeword bits are written one to each disk. Two or perhaps three disk drives can go down and the others can recover the data. The effective channel model here is a binary erasure channel, because it is assumed that we can tell when a disk is dead.

It is not possible to recover the data for *some* choices of the three dead disk drives; can you see why?

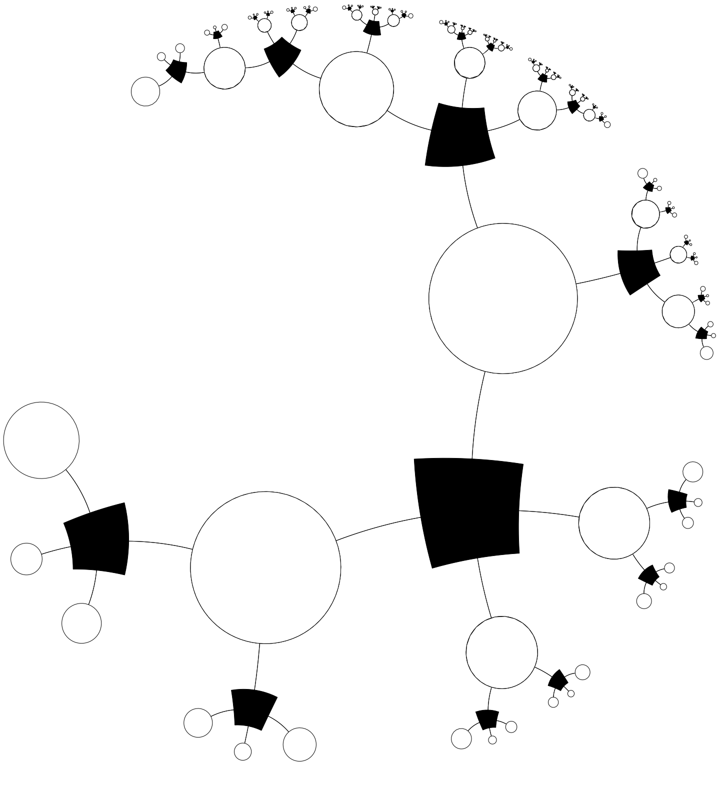
▷ Exercise 11.10.[*2*, p.190] Give an example of three disk drives that, if lost, lead to failure of the above RAID system, and three that can be lost without failure.

Solution to exercise 11.10 (p.190). The $(7,4)$ Hamming code has codewords of weight 3. If any set of three disk drives corresponding to one of those codewords is lost, then the other four disks can recover only 3 bits of information about the four source bits; a fourth bit is lost. [cf. exercise 13.13 (p.220) with $q = 2$: there are no binary MDS codes. This deficit is discussed further in section 13.11.]

Any other set of three disk drives can be lost without problems because the corresponding four by four submatrix of the generator matrix is invertible. A better code would be a digital fountain – see Chapter 50.

Part III

Further Topics in Information Theory



About Chapter 12

In Chapters 1–11, we concentrated on two aspects of information theory and coding theory: source coding – the compression of information so as to make efficient use of data transmission and storage channels; and channel coding – the redundant encoding of information so as to be able to detect and correct communication errors.

In both these areas we started by ignoring practical considerations, concentrating on the question of the theoretical limitations and possibilities of coding. We then discussed practical source-coding and channel-coding schemes, shifting the emphasis towards computational feasibility. But the prime criterion for comparing encoding schemes remained the efficiency of the code in terms of the channel resources it required: the best source codes were those that achieved the greatest compression; the best channel codes were those that communicated at the highest rate with a given probability of error.

In this chapter we now shift our viewpoint a little, thinking of *ease of information retrieval* as a primary goal. It turns out that the random codes which were theoretically useful in our study of channel coding are also useful for rapid information retrieval.

Efficient information retrieval is one of the problems that brains seem to solve effortlessly, and content-addressable memory is one of the topics we will study when we look at neural networks.

12

Hash Codes: Codes for Efficient Information Retrieval

▶ 12.1 The information-retrieval problem

A simple example of an information-retrieval problem is the task of implementing a phone directory service, which, in response to a person's *name*, returns (a) a confirmation that that person is listed in the directory; and (b) the person's phone number and other details. We could formalize this problem as follows, with S being the number of names that must be stored in the directory.

string length	$N \simeq 200$
number of strings	$S \simeq 2^{23}$
number of possible strings	$2^N \simeq 2^{200}$

Figure 12.1. Cast of characters.

You are given a list of S binary strings of length N bits, $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(S)}\}$, where S is considerably smaller than the total number of possible strings, 2^N. We will call the superscript 's' in $\mathbf{x}^{(s)}$ the *record number* of the string. The idea is that s runs over customers in the order in which they are added to the directory and $\mathbf{x}^{(s)}$ is the name of customer s. We assume for simplicity that all people have names of the same length. The name length might be, say, $N = 200$ bits, and we might want to store the details of ten million customers, so $S \simeq 10^7 \simeq 2^{23}$. We will ignore the possibility that two customers have identical names.

The task is to construct the inverse of the mapping from s to $\mathbf{x}^{(s)}$, i.e., to make a system that, given a string $\mathbf{x}$, returns the value of s such that $\mathbf{x} = \mathbf{x}^{(s)}$ if one exists, and otherwise reports that no such s exists. (Once we have the record number, we can go and look in memory location s in a separate memory full of phone numbers to find the required number.) The aim, when solving this task, is to use minimal computational resources in terms of the amount of memory used to store the inverse mapping from $\mathbf{x}$ to s and the amount of time to compute the inverse mapping. And, preferably, the inverse mapping should be implemented in such a way that further new strings can be added to the directory in a small amount of computer time too.

Some standard solutions

The simplest and dumbest solutions to the information-retrieval problem are a look-up table and a raw list.

The look-up table is a piece of memory of size $2^N \log_2 S$, $\log_2 S$ being the amount of memory required to store an integer between 1 and S. In each of the 2^N locations, we put a zero, except for the locations $\mathbf{x}$ that correspond to strings $\mathbf{x}^{(s)}$, into which we write the value of s.

The look-up table is a simple and quick solution, but only if there is sufficient memory for the table, and if the cost of looking up entries in memory is independent of the memory size. But in our definition of the task, we assumed that N is about 200 bits or more, so the amount of memory required would be of size 2^{200}; this solution is completely out of the question. Bear in mind that the number of particles in the solar system is only about 2^{190}.

The raw list is a simple list of ordered pairs $(s, \mathbf{x}^{(s)})$ ordered by the value of s. The mapping from $\mathbf{x}$ to s is achieved by searching through the list of strings, starting from the top, and comparing the incoming string $\mathbf{x}$ with each record $\mathbf{x}^{(s)}$ until a match is found. This system is very easy to maintain, and uses a small amount of memory, about SN bits, but is rather slow to use, since on average five million pairwise comparisons will be made.

▷ Exercise 12.1.[2, p.202] Show that the average time taken to find the required string in a raw list, assuming that the original names were chosen at random, is about $S + N$ binary comparisons. (Note that you don't have to compare the whole string of length N, since a comparison can be terminated as soon as a mismatch occurs; show that you need on average two binary comparisons per incorrect string match.) Compare this with the worst-case search time – assuming that the devil chooses the set of strings and the search key.

The standard way in which phone directories are made improves on the look-up table and the raw list by using an *alphabetically-ordered list*.

Alphabetical list. The strings $\{\mathbf{x}^{(s)}\}$ are sorted into alphabetical order. Searching for an entry now usually takes less time than was needed for the raw list because we can take advantage of the sortedness; for example, we can open the phonebook at its middle page, and compare the name we find there with the target string; if the target is 'greater' than the middle string then we know that the required string, if it exists, will be found in the second half of the alphabetical directory. Otherwise, we look in the first half. By iterating this splitting-in-the-middle procedure, we can identify the target string, or establish that the string is not listed, in $\lceil \log_2 S \rceil$ string comparisons. The expected number of binary comparisons per string comparison will tend to increase as the search progresses, but the total number of binary comparisons required will be no greater than $\lceil \log_2 S \rceil N$.

The amount of memory required is the same as that required for the raw list.

Adding new strings to the database requires that we insert them in the correct location in the list. To find that location takes about $\lceil \log_2 S \rceil$ binary comparisons.

Can we improve on the well-established alphabetized list? Let us consider our task from some new viewpoints.

The task is to construct a mapping $\mathbf{x} \rightarrow s$ from N bits to $\log_2 S$ bits. This is a pseudo-invertible mapping, since for any $\mathbf{x}$ that maps to a non-zero s, the customer database contains the pair $(s, \mathbf{x}^{(s)})$ that takes us back. Where have we come across the idea of mapping from N bits to M bits before?

We encountered this idea twice: first, in source coding, we studied block codes which were mappings from strings of N symbols to a selection of one label in a list. The task of information retrieval is similar to the task (which we never actually solved) of making an encoder for a typical-set compression code.

The second time that we mapped bit strings to bit strings of another dimensionality was when we studied channel codes. There, we considered codes that mapped from K bits to N bits, with N greater than K, and we made theoretical progress using *random* codes.

In hash codes, we put together these two notions. We will study random codes that map from N bits to M bits where M is *smaller* than N.

The idea is that we will map the original high-dimensional space down into a lower-dimensional space, one in which it is feasible to implement the dumb look-up table method which we rejected a moment ago.

▶ 12.2 Hash codes

string length	$N \simeq 200$
number of strings	$S \simeq 2^{23}$
size of hash function	$M \simeq 30$ bits
size of hash table	$T = 2^M$ $\simeq 2^{30}$

Figure 12.2. Revised cast of characters.

First we will describe how a hash code works, then we will study the properties of idealized hash codes. A hash code implements a solution to the information-retrieval problem, that is, a mapping from $\mathbf{x}$ to s, with the help of a pseudo-random function called a *hash function*, which maps the N-bit string $\mathbf{x}$ to an M-bit string $\mathbf{h}(\mathbf{x})$, where M is smaller than N. M is typically chosen such that the 'table size' $T \simeq 2^M$ is a little bigger than S – say, ten times bigger. For example, if we were expecting S to be about a million, we might map $\mathbf{x}$ into a 30-bit hash $\mathbf{h}$ (regardless of the size N of each item $\mathbf{x}$). The hash function is some fixed deterministic function which should ideally be indistinguishable from a fixed random code. For practical purposes, the hash function must be quick to compute.

Two simple examples of hash functions are:

Division method. The table size T is a prime number, preferably one that is not close to a power of 2. The hash value is the remainder when the integer $\mathbf{x}$ is divided by T.

Variable string addition method. This method assumes that $\mathbf{x}$ is a string of bytes and that the table size T is 256. The characters of $\mathbf{x}$ are added, modulo 256. This hash function has the defect that it maps strings that are anagrams of each other onto the same hash.

It may be improved by putting the running total through a fixed pseudorandom permutation after each character is added. In the *variable string exclusive-or method* with table size $\leq 65\,536$, the string is hashed twice in this way, with the initial running total being set to 0 and 1 respectively (algorithm 12.3). The result is a 16-bit hash.

Having picked a hash function $\mathbf{h}(\mathbf{x})$, we implement an information retriever as follows. (See figure 12.4.)

Encoding. A piece of memory called the *hash table* is created of size $2^M b$ memory units, where b is the amount of memory needed to represent an integer between 0 and S. This table is initially set to zero throughout. Each memory $\mathbf{x}^{(s)}$ is put through the hash function, and at the location in the hash table corresponding to the resulting vector $\mathbf{h}^{(s)} = \mathbf{h}(\mathbf{x}^{(s)})$, the integer s is written – unless that entry in the hash table is already occupied, in which case we have a *collision* between $\mathbf{x}^{(s)}$ and some earlier $\mathbf{x}^{(s')}$ which both happen to have the same hash code. Collisions can be handled in various ways – we will discuss some in a moment – but first let us complete the basic picture.

```
unsigned char Rand8[256];     // This array contains a random
                                 permutation from 0..255 to 0..255
int Hash(char *x) {           // x is a pointer to the first char;
     int h;                   //    *x is the first character
     unsigned char h1, h2;

     if (*x == 0) return 0;   // Special handling of empty string
     h1 = *x; h2 = *x + 1;    // Initialize two hashes
     x++;                     // Proceed to the next character
     while (*x) {
         h1 = Rand8[h1 ^ *x]; // Exclusive-or with the two hashes
         h2 = Rand8[h2 ^ *x]; //    and put through the randomizer
         x++;
     }                        // End of string is reached when *x=0
     h = ((int)(h1)<<8) |     // Shift h1 left 8 bits and add h2
          (int) h2 ;
     return h ;               // Hash is concatenation of h1 and h2
}
```

Algorithm 12.3. C code implementing the variable string exclusive-or method to create a hash h in the range $0\ldots65\,535$ from a string x. Author: Thomas Niemann.

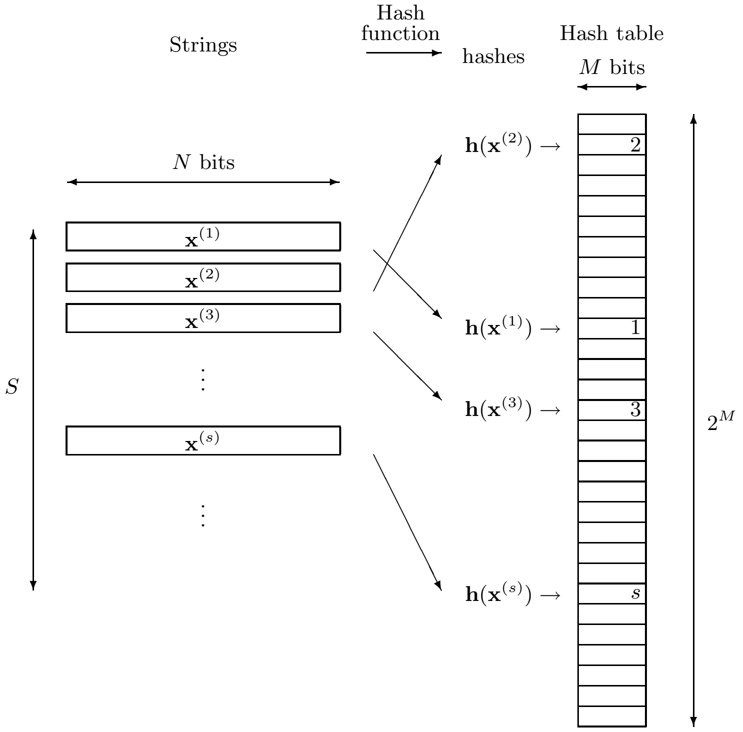

Strings
Hash function
hashes
Hash table
M bits
N bits
$\mathbf{x}^{(1)}$
$\mathbf{x}^{(2)}$
$\mathbf{x}^{(3)}$
$\mathbf{x}^{(s)}$
S
$\mathbf{h}(\mathbf{x}^{(2)}) \rightarrow$
$\mathbf{h}(\mathbf{x}^{(1)}) \rightarrow$
$\mathbf{h}(\mathbf{x}^{(3)}) \rightarrow$
$\mathbf{h}(\mathbf{x}^{(s)}) \rightarrow$
2
1
3
s
2^M

Figure 12.4. Use of hash functions for information retrieval. For each string $\mathbf{x}^{(s)}$, the hash $\mathbf{h} = \mathbf{h}(\mathbf{x}^{(s)})$ is computed, and the value of s is written into the $\mathbf{h}$th row of the hash table. Blank rows in the hash table contain the value zero. The table size is $T = 2^M$.

Decoding. To retrieve a piece of information corresponding to a target vector $\mathbf{x}$, we compute the hash $\mathbf{h}$ of $\mathbf{x}$ and look at the corresponding location in the hash table. If there is a zero, then we know immediately that the string $\mathbf{x}$ is not in the database. The cost of this answer is the cost of one hash-function evaluation and one look-up in the table of size 2^M. If, on the other hand, there is a non-zero entry s in the table, there are two possibilities: either the vector $\mathbf{x}$ is indeed equal to $\mathbf{x}^{(s)}$; or the vector $\mathbf{x}^{(s)}$ is another vector that happens to have the same hash code as the target $\mathbf{x}$. (A third possibility is that this non-zero entry might have something to do with our yet-to-be-discussed collision-resolution system.)

To check whether $\mathbf{x}$ is indeed equal to $\mathbf{x}^{(s)}$, we take the tentative answer s, look up $\mathbf{x}^{(s)}$ in the original forward database, and compare it bit by bit with $\mathbf{x}$; if it matches then we report s as the desired answer. This successful retrieval has an overall cost of one hash-function evaluation, one look-up in the table of size 2^M, another look-up in a table of size S, and N binary comparisons – which may be much cheaper than the simple solutions presented in section 12.1.

▷ Exercise 12.2.[*2*, p.202] If we have checked the first few bits of $\mathbf{x}^{(s)}$ with $\mathbf{x}$ and found them to be equal, what is the probability that the correct entry has been retrieved, if the alternative hypothesis is that $\mathbf{x}$ is actually not in the database? Assume that the original source strings are random, and the hash function is a random hash function. How many binary evaluations are needed to be sure with odds of a billion to one that the correct entry has been retrieved?

The hashing method of information retrieval can be used for strings $\mathbf{x}$ of arbitrary length, if the hash function $\mathbf{h}(\mathbf{x})$ can be applied to strings of any length.

▶ 12.3 Collision resolution

We will study two ways of resolving collisions: appending in the table, and storing elsewhere.

Appending in table

When encoding, if a collision occurs, we continue down the hash table and write the value of s into the next available location in memory that currently contains a zero. If we reach the bottom of the table before encountering a zero, we continue from the top.

When decoding, if we compute the hash code for $\mathbf{x}$ and find that the s contained in the table doesn't point to an $\mathbf{x}^{(s)}$ that matches the cue $\mathbf{x}$, we continue down the hash table until we either find an s whose $\mathbf{x}^{(s)}$ does match the cue $\mathbf{x}$, in which case we are done, or else encounter a zero, in which case we know that the cue $\mathbf{x}$ is not in the database.

For this method, it is essential that the table be substantially bigger in size than S. If $2^M < S$ then the encoding rule will become stuck with nowhere to put the last strings.

Storing elsewhere

A more robust and flexible method is to use *pointers* to additional pieces of memory in which collided strings are stored. There are many ways of doing this. As an example, we could store in location $\mathbf{h}$ in the hash table a pointer (which must be distinguishable from a valid record number s) to a 'bucket' where all the strings that have hash code $\mathbf{h}$ are stored in a *sorted list*. The encoder sorts the strings in each bucket alphabetically as the hash table and buckets are created.

The decoder simply has to go and look in the relevant bucket and then check the short list of strings that are there by a brief alphabetical search.

This method of storing the strings in buckets allows the option of making the hash table quite small, which may have practical benefits. We may make it so small that almost all strings are involved in collisions, so all buckets contain a small number of strings. It only takes a small number of binary comparisons to identify which of the strings in the bucket matches the cue $\mathbf{x}$.

12.4 Planning for collisions: a birthday problem

Exercise 12.3.[2, p.202] If we wish to store S entries using a hash function whose output has M bits, how many collisions should we expect to happen, assuming that our hash function is an ideal random function? What size M of hash table is needed if we would like the expected number of collisions to be smaller than 1?

What size M of hash table is needed if we would like the expected number of collisions to be a small fraction, say 1%, of S?

[Notice the similarity of this problem to exercise 9.20 (p.156).]

12.5 Other roles for hash codes

Checking arithmetic

If you wish to check an addition that was done by hand, you may find useful the method of *casting out nines*. In casting out nines, one finds the sum, modulo nine, of all the *digits* of the numbers to be summed and compares it with the sum, modulo nine, of the digits of the putative answer. [With a little practice, these sums can be computed much more rapidly than the full original addition.]

Example 12.4. In the calculation shown in the margin the sum, modulo nine, of the digits in 189+1254+238 is 7, and the sum, modulo nine, of 1+6+8+1 is 7. The calculation thus passes the casting-out-nines test.

```
  189
+1254
+ 238
-----
 1681
```

Casting out nines gives a simple example of a hash function. For any addition expression of the form $a + b + c + \cdots$, where $a, b, c, \ldots$ are decimal numbers we define $h \in \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$ by

$$h(a + b + c + \cdots) = \text{ sum modulo nine of all digits in } a, b, c \; ; \tag{12.1}$$

then it is nice property of decimal arithmetic that if

$$a + b + c + \cdots = m + n + o + \cdots \tag{12.2}$$

then the hashes $h(a + b + c + \cdots)$ and $h(m + n + o + \cdots)$ are equal.

▷ Exercise 12.5.[1, p.203] What evidence does a correct casting-out-nines match give in favour of the hypothesis that the addition has been done correctly?

Error detection among friends

Are two files the same? If the files are on the same computer, we could just compare them bit by bit. But if the two files are on separate machines, it would be nice to have a way of confirming that two files are identical without having to transfer one of the files from A to B. [And even if we did transfer one of the files, we would still like a way to confirm whether it has been received without modifications!]

This problem can be solved using hash codes. Let Alice and Bob be the holders of the two files; Alice sent the file to Bob, and they wish to confirm it has been received without error. If Alice computes the hash of her file and sends it to Bob, and Bob computes the hash of his file, using the same M-bit hash function, and the two hashes match, then Bob can deduce that the two files are almost surely the same.

Example 12.6. What is the probability of a false negative, i.e., the probability, given that the two files do differ, that the two hashes are nevertheless identical?

If we assume that the hash function is random and that the process that causes the files to differ knows nothing about the hash function, then the probability of a false negative is 2^{-M}. □

A 32-bit hash gives a probability of false negative of about 10^{-10}. It is common practice to use a linear hash function called a 32-bit cyclic redundancy check to detect errors in files. (A cyclic redundancy check is a set of 32 parity-check bits similar to the 3 parity-check bits of the $(7,4)$ Hamming code.)

> To have a false-negative rate smaller than one in a billion, $M = 32$ bits is plenty, if the errors are produced by noise.

▷ Exercise 12.7.[*2*, p.203] Such a simple parity-check code only detects errors; it doesn't help correct them. Since error-*correcting* codes exist, why not use one of them to get some error-correcting capability too?

Tamper detection

What if the differences between the two files are not simply 'noise', but are introduced by an adversary, a clever *forger* called Fiona, who modifies the original file to make a forgery that purports to be Alice's file? How can Alice make a digital signature for the file so that Bob can confirm that no-one has tampered with the file? And how can we prevent Fiona from listening in on Alice's signature and attaching it to other files?

Let's assume that Alice computes a hash function for the file and sends it securely to Bob. If Alice computes a simple hash function for the file like the linear cyclic redundancy check, and Fiona knows that this is the method of verifying the file's integrity, Fiona can make her chosen modifications to the file and then easily identify (by linear algebra) a further 32-or-so single bits that, when flipped, restore the hash function of the file to its original value. *Linear hash functions give no security against forgers.*

We must therefore require that the hash function be *hard to invert* so that no-one can construct a tampering that leaves the hash function unaffected. We would still like the hash function to be easy to compute, however, so that Bob doesn't have to do hours of work to verify every file he received. Such a hash function – easy to compute, but hard to invert – is called a *one-way* *hash function*. Finding such functions is one of the active research areas of cryptography.

A hash function that is widely used in the free software community to confirm that two files do not differ is `MD5`, which produces a 128-bit hash. The details of how it works are quite complicated, involving convoluted exclusive-or-ing and if-ing and and-ing.[1]

Even with a good one-way hash function, the digital signatures described above are still vulnerable to attack, if Fiona has access to the hash function. Fiona could take the tampered file and hunt for a further tiny modification to it such that its hash matches the original hash of Alice's file. This would take some time – on average, about 2^{32} attempts, if the hash function has 32 bits – but eventually Fiona would find a tampered file that matches the given hash. To be secure against forgery, digital signatures must either have enough bits for such a random search to take too long, or the hash function itself must be kept secret.

> Fiona has to hash 2^M files to cheat. 2^{32} file modifications is not very many, so a 32-bit hash function is not large enough for forgery prevention.

Another person who might have a motivation for forgery is Alice herself. For example, she might be making a bet on the outcome of a race, without wishing to broadcast her prediction publicly; a method for placing bets would be for her to send to Bob the bookie the hash of her bet. Later on, she could send Bob the details of her bet. Everyone can confirm that her bet is consistent with the previously publicized hash. [This method of secret publication was used by Isaac Newton and Robert Hooke when they wished to establish priority for scientific ideas without revealing them. Hooke's hash function was alphabetization as illustrated by the conversion of *UT TENSIO, SIC VIS* into the anagram `CEIIINOSSSTTUV`.] Such a protocol relies on the assumption that Alice cannot change her bet after the event without the hash coming out wrong. How big a hash function do we need to use to ensure that Alice cannot cheat? The answer is different from the size of the hash we needed in order to defeat Fiona above, because Alice is the author of *both* files. Alice could cheat by searching for two files that have identical hashes to each other. For example, if she'd like to cheat by placing two bets for the price of one, she could make a large number N_1 of versions of bet one (differing from each other in minor details only), and a large number N_2 of versions of bet two, and hash them all. If there's a collision between the hashes of two bets of different types, then she can submit the common hash and thus buy herself the option of placing either bet.

Example 12.8. If the hash has M bits, how big do N_1 and N_2 need to be for Alice to have a good chance of finding two different bets with the same hash?

This is a birthday problem like exercise 9.20 (p.156). If there are N_1 Montagues and N_2 Capulets at a party, and each is assigned a 'birthday' of M bits, the expected number of collisions between a Montague and a Capulet is

$$N_1 N_2 2^{-M}, \tag{12.3}$$

[1] `http://www.freesoft.org/CIE/RFC/1321/3.htm`

so to minimize the number of files hashed, $N_1 + N_2$, Alice should make N_1 and N_2 equal, and will need to hash about $2^{M/2}$ files until she finds two that match. □

> Alice has to hash $2^{M/2}$ files to cheat. [This is the square root of the number of hashes Fiona had to make.]

If Alice has the use of $C = 10^6$ computers for $T = 10$ years, each computer taking $t = 1$ ns to evaluate a hash, the bet-communication system is secure against Alice's dishonesty only if $M \gg 2 \log_2 CT/t \simeq 160$ bits.

Further reading

The Bible for hash codes is volume 3 of Knuth (1968). I highly recommend the story of Doug McIlroy's `spell` program, as told in section 13.8 of *Programming Pearls* (Bentley, 2000). This astonishing piece of software makes use of a 64-kilobyte data structure to store the spellings of all the words of 75 000-word dictionary.

▶ 12.6 Further exercises

Exercise 12.9.$^{[1]}$ What is the shortest the address on a typical international letter could be, if it is to get to a unique human recipient? (Assume the permitted characters are `[A-Z,0-9]`.) How long are typical email addresses?

Exercise 12.10.$^{[2,\text{ p.203}]}$ How long does a piece of text need to be for you to be pretty sure that no human has written that string of characters before? How many notes are there in a new melody that has not been composed before?

▷ Exercise 12.11.$^{[3,\text{ p.204}]}$ Pattern recognition by molecules.

Some proteins produced in a cell have a regulatory role. A regulatory protein controls the transcription of specific genes in the genome. This control often involves the protein's binding to a particular DNA sequence in the vicinity of the regulated gene. The presence of the bound protein either promotes or inhibits transcription of the gene.

(a) Use information-theoretic arguments to obtain a lower bound on the size of a typical protein that acts as a regulator specific to one gene in the whole human genome. Assume that the genome is a sequence of 3×10^9 nucleotides drawn from a four letter alphabet $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$; a protein is a sequence of amino acids drawn from a twenty letter alphabet. [Hint: establish how long the recognized DNA sequence has to be in order for that sequence to be unique to the vicinity of one gene, treating the rest of the genome as a random sequence. Then discuss how big the protein must be to recognize a sequence of that length uniquely.]

(b) Some of the sequences recognized by DNA-binding regulatory proteins consist of a subsequence that is repeated twice or more, for example the sequence

$$\underline{\mathtt{GCCCCC}}\mathtt{CACCCCT}\underline{\mathtt{GCCCCC}} \qquad (12.4)$$