Information Theory

http://www.inf.ed.ac.uk/teaching/courses/it/

Week 1
Introduction to Information Theory

Iain Murray, 2012

School of Informatics, University of Edinburgh

Course structure

Constituents:
— ~17 lectures
— Tutorials starting in week 3
— 1 assignment (20% marks)

Website:
http://tinyurl.com/itmsc
http://www.inf.ed.ac.uk/teaching/courses/it/
Notes, assignments, tutorial material, news (optional RSS feed)

Prerequisites: some maths, some programming ability

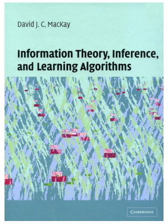
Maths background: This is a theoretical course so some general mathematical ability is essential. Be very familiar with logarithms, mathematical notation (such as sums) and some calculus.

Probabilities are used extensively: Random variables; expectation; Bernoulli, Binomial and Gaussian distributions; joint and conditional probabilities. There will be some review, but expect to work hard if you don't have the background.

Programming background: by the end of the course you are expected to be able to implement algorithms involving probability distributions over many variables. However, I am not going to teach you a programming language. I can discuss programming issues in the tutorials. I won't mark code, only its output, so you are free to pick a language. Pick one that's quick and easy to use.

The scope of this course is to understand the applicability and properties of methods. Programming will be exploratory: slow, high-level but clear code is fine. We will not be writing the final optimized code to sit on a hard-disk controller!

Resources / Acknowledgements



Recommended course text book

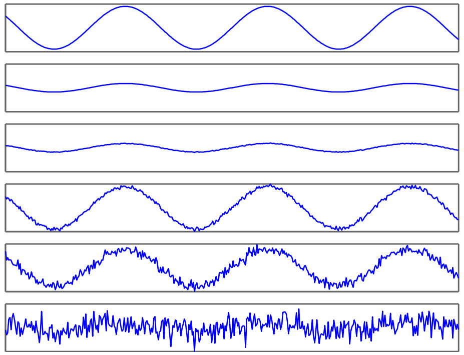
Inexpensive for a hardback textbook
(Stocked in Blackwells, Amazon currently cheaper)

Also free online:
http://www.inference.phy.cam.ac.uk/mackay/itila/

Those preferring a theorem-lemma style book could check out:
Elements of information theory, Cover and Thomas

I made use of course notes by MacKay and from CSC310 at the University of Toronto (Radford Neal, 2004; Sam Roweis, 2006)

Communicating with noise



Consider sending an audio signal by *amplitude modulation*: the desired speaker-cone position is the height of the signal. The figure shows an encoding of a pure tone.

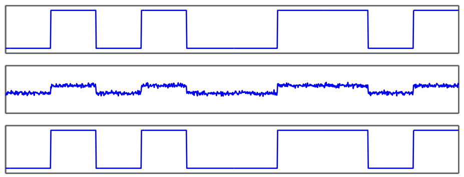
A classical problem with this type of communication channel is attenuation: the amplitude of the signal decays over time. (The details of this in a real system could be messy.) Assuming we could regularly boost the signal, we would also amplify any noise that has been added to the signal. After several cycles of attenuation, noise addition and amplification, corruption can be severe.

A variety of analogue encodings are possible, but whatever is used, no 'boosting' process can ever return a corrupted signal exactly to its original form. In digital communication the sent message comes from a discrete set. If the message is corrupted we can 'round' to the nearest discrete message. It is possible, but not guaranteed, we'll restore the message to exactly the one sent.

Digital communication

Encoding: amplitude modulation not only choice.
Can re-represent messages to improve signal-to-noise ratio

Digital encodings: signal takes on discrete values



Communication channels

modem → phone line → modem

Galileo → radio waves → Earth

finger tips → nerves → brain

parent cell → daughter cells

computer memory → disk drive → computer memory

The challenge

Real channels are error prone

Physical solutions:

£ £ £

Change the system to reduce probability of error.

Cool system, increase power, . . .

System solution

Send more robust encodings over *existing* channel

message
↓
encoded message
↓
corrupted encoding
↓
decoded message

But how is reliable communication possible at all?

Repetition codes

Repetition code R_3:

`hello there` → `hhheeellllllooo   ttthhheeerrreee`

↓ (noise added)

`hello thfr?` ← `hhkgeeslllllqooc m qttzhhfferrrBme`

Possible results of errors:

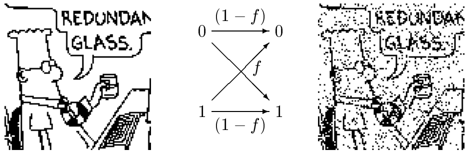
— Corrected
— Detected
— Undetected

Binary symmetric channel

Binary messages: 0010100111001...

Each 0 or 1 is flipped with probability $f = 0.1$

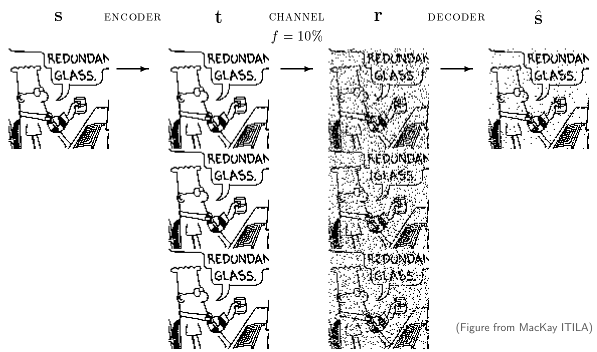
$$x \quad \begin{matrix} 0 \to 0 \\ 1 \to 1 \end{matrix} \quad y \qquad \begin{aligned} P(y=0 \mid x=0) &= 1-f; & P(y=0 \mid x=1) &= f; \\ P(y=1 \mid x=0) &= f; & P(y=1 \mid x=1) &= 1-f. \end{aligned}$$



(Figure from MacKay ITILA)

Can repetition codes give reliable communication?

Repetition code performance



(Figure from MacKay ITILA)

Probability of error per bit ≈ 0.03. What's good enough?

Consider a single 0 transmitted using R_3 as 000

Eight possible messages could be received:
000 100 010 001 110 101 011 111

Majority vote decodes the first four correctly but the next four result in errors. Fortunately the first four are more probable than the rest!

Probability of 111 is small: $f^3 = 0.1^3 = 10^{-3}$
Probability of two bit errors is $3f^2(1-f) = 0.03 \times 0.9$
Total probability of error is a bit less than 3%

How to reduce probability of error further? Repeat more! (N times)

Probability of bit error = Probability > half of bits are flipped:

$$p_b = \sum_{r=\frac{N+1}{2}}^{N} \binom{N}{r} f^r (1-f)^{N-r}$$

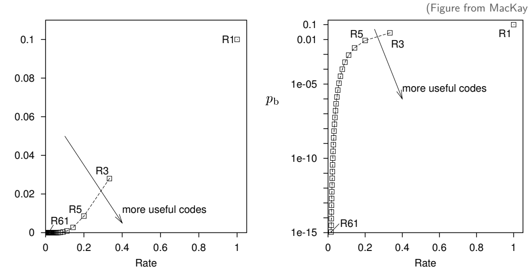
But transmit symbols N times slower! *Rate* is $1/N$.

Repetition code performance

Binary messages: 0010100111001...

Each 0 or 1 is flipped with probability $f = 0.1$

(Figure from MacKay ITILA)



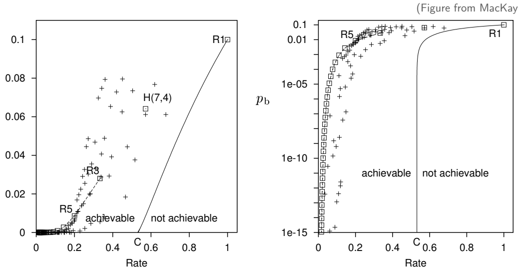
p_b = probability of error at each bit of message

What is achievable?

Binary messages: 0010100111001...

Each 0 or 1 is flipped with probability $f = 0.1$

(Figure from MacKay ITILA)



p_b = probability of error at each bit of message

Course content

Theoretical content

— Shannon's noisy channel and source coding theorems
— Much of the theory is non-constructive
— However bounds are useful and approachable

Practical coding algorithms

— Reliable communication
— Compression

Tools and related material

— Probabilistic modelling and machine learning

Storage capacity

3 decimal digits allow $10^3 = 1{,}000$ numbers: 000–999

3 **binary digits or bits** allow $2^3 = 8$ numbers:
000, 001, 010, 011, 100, 101, 110, 111

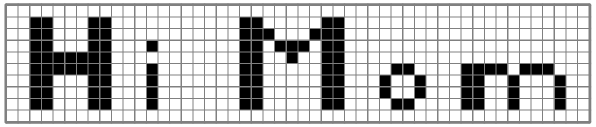
8 bits, a 'byte', can store one of $2^8 = 256$ characters

Indexing I items requires at least
$\log_{10} I$ decimal digits or $\log_2 I$ bits

Reminder: $b = \log_2 I \;\Rightarrow\; 2^b = I \;\Rightarrow\; b \log 2 = \log I \;\Rightarrow\; b = \frac{\log I}{\log 2}$

Representing data / coding

Example: a 10×50 binary image



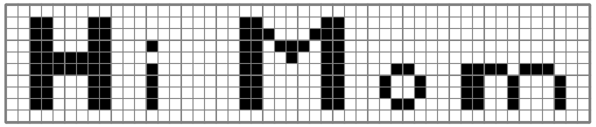
Assume image dimensions are known

Pixels could be represented with 1s and 0s

This encoding takes **500 bits** (binary digits)

2^{500} images can be encoded. The universe is $\approx 2^{98}$ picoseconds old.

Exploit sparseness



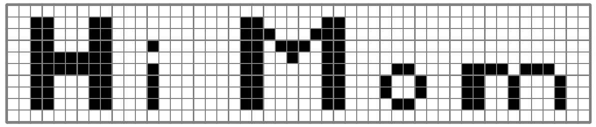
As there are fewer black pixels we send just them.
Encode row + start/end column for each run in binary.

Requires $(4{+}6{+}6){=}16$ bits per run (can you see why?)
There are 54 black runs ⇒ 54×16 = **864 bits**

That's worse than the 500 bit encoding we started with!

Scan columns instead: 33 runs, $(6{+}4{+}4){=}14$ bits each. **462 bits**.

Run-length encoding

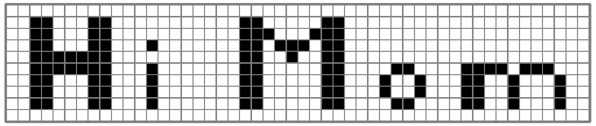


Common idea: store lengths of runs of pixels

Longest possible run = 500 pixels, need 9 bits for run length
Use 1 bit to store colour of first run (should we?)

Scanning along rows: 109 runs ⇒ **982 bits**(!)
Scanning along cols: 67 runs ⇒ **604 bits**

Adapting run-length encoding

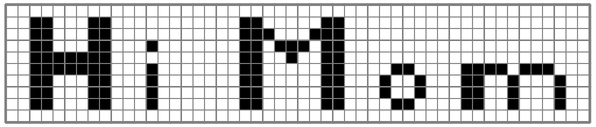


Store number of bits actually needed for runs in a header.
$4{+}4{=}8$ bits give sizes needed for black and white runs.

Scanning along rows: **501 bits** (includes 8+1=9 header bits)
55 white runs up to 52 long, 55×6 = 330 bits
54 black runs up to 7 long, 54×3 = 162 bits

Scanning along cols: **249 bits**
34 white runs up to 72 long, 24×7 = 168 bits
33 black runs up to 8 long, 24×3 = 72 bits (3 bits/run if no zero-length runs; we did need the first-run-colour header bit!)

Rectangles

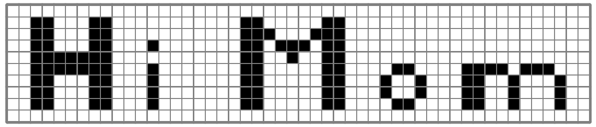


Exploit spatial structure: represent image as 20 rectangles

Version 1:
Each rectangle: (x_1, y_1, x_2, y_2), 4+6+4+6 = 20 bits
Total size: 20×20 = **400 bits**

Version 2:
Header for max rectangle size: 2+3 = 5 bits
Each rectangle: (x_1, y_1, w, h), 4+6+3+3 = 16 bits
Total size: 20×16 + 5 = **325 bits**

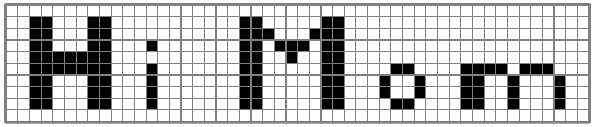
Off-the-shelf solutions?



Established image compressors:
Use PNG: 128 bytes = **1024 bits**
Use GIF: 98 bytes = **784 bits**
JBIG2: 108 bytes = 864 bits
DjVu: 124 bytes = 992 bits

Unfair: image is tiny, file format overhead: headers, image dims
Smallest possible GIF file is about 35 bytes. Smallest possible PNG file is about 67 bytes.
Not strictly meaningful, but: (98-35)×8 = 504 bits. (128-67)×8 = 488 bits

Store as text



Assume we know the font

Encode six characters in a 64 character alphabet (say)

Total size: $6\times \log_2 64$ = **36 bits**

"Overfitting"

We can compress the '`Hi Mom`' image down to **1 bit**:

Represent '`Hi Mom`' image with a single '1'

All other files encoded with '0' and a naive encoding of the image.

. . . the actual message is one selected from a set of possible messages. The system must be designed to operate for each possible selection, not just the one which will actually be chosen since this is unknown at the time of design.

— Shannon, 1948

Summary of lecture 1 (slide 1/2)

Digital communication can work reliably over a noisy channel. We add *redundancy* to a message, so that we can try to infer what corruption occurred and undo it.

Repetition codes simply repeat each message symbol N times. A majority vote at the receiving end removes errors unless more than half of the repetitions were corrupted. Increasing N reduces the error rate, but the *rate* of the code is $1/N$: transmission is slower, or more storage space is used. For the Binary Symmetric Channel the error probability is: $\sum_{r=(N+1)/2}^{N} \binom{N}{r} f^r (1-f)^{N-r}$

Amazing claim: it is possible to get arbitrarily small errors at a fixed rate known as the *capacity* of the channel. *Aside:* codes that approach the capacity send a more complicated message than simple repetitions. Inferring what corruptions must have occurred (occurred with overwhelmingly high probability) is more complex than a majority vote. The algorithms are related to how some groups perform inference in machine learning.

Summary of lecture 1 (slide 2/2)

First task: represent data optimally when there is no noise

Representing files as (binary) numbers:

C bits (binary digits) can index $I = 2^C$ objects.

$\log I = C \log 2$, $C = \frac{\log I}{\log 2}$ for logs of any base, $C = \log_2 I$

In information theory textbooks "log" often means "$\log_2$".

Experiences with the `Hi Mom` **image:**

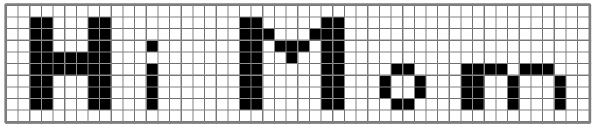
Unless we're careful, we can expand the file dramatically.
When developing a fancy method, always consider simple baselines.

The bit encodings and header bits I used were inelegant.
We'd like more principled and better ways to proceed. (See later).

Summarizing groups of bits (rectangles, runs, etc.) can lead to fewer objects to index. Structure in the image allows compression.

Cheating: add whole image as a "word" in our dictionary.
Schemes should work on future data that the receiver hasn't seen.

Where now



What are the fundamental limits to compression?
Can we avoid all the hackery?
Or at least make it clearer how to proceed?

This course: Shannon's information theory relates compression to *probabilistic modelling*

A simple probabilistic model (predict from three previous neighbouring pixels) and an *arithmetic coder* can compress to about **220 bits**.

Why is compression possible?

Try to compress *all* b bit files to $<b$ bits

There are 2^b possible files but only $(2^b - 1)$ codewords

Theorem: if we compress some files we must expand others
(or fail to represent some files unambiguously)

Search for the `comp.compression` FAQ currently available at:
`http://www.faqs.org/faqs/compression-faq/`

Which files to compress?

We choose to compress the **more probable** files

Example: compress 28×28 binary images like this:

At the expense of longer encodings for files like this:

There are 2^{784} binary images. I think $< 2^{125}$ are like the digits

Sparse file model

Long binary vector $\mathbf{x}$, mainly zeros

Assume bits drawn independently

Bernoulli distribution, a single "bent coin" flip

$$P(x_i \,|\, p) = \begin{cases} p & \text{if } x_i = 1 \\ (1-p) \equiv p_0 & \text{if } x_i = 0 \end{cases}$$

How would we compress a large file for $p = 0.1$?

Idea: encode blocks of N bits at a time

Intuitions:

'Blocks' of lengths $N = 1$ give naive encoding: 1 bit / symbol
Blocks of lengths $N = 2$ aren't going to help
... *maybe* we want long blocks

For large N, some blocks won't appear in the file, e.g. `11111111111...`
The receiver won't know exactly which blocks will be used
Don't want a header listing blocks: expensive for large N.

Instead we use our probabilistic model of the source to guide which blocks will be useful. For $N = 5$ the 6 most probable blocks are:

`00000 00001 00010 00100 01000 10000`

3 bits can encode these as 0–5 in binary: `000 001 010 011 100 101`

Use spare codewords (`110 111`) followed by 4 more bits to encode remaining blocks. Expected length of this code $= 3 + 4\,P(\text{need 4 more})$ $= 3 + 4(1-(1-p)^5 - 5p(1-p)^4) \approx 3.3$ bits $\Rightarrow 3.3/5 \approx 0.67$ bits/symbol

Quick quiz

Q1. Toss a fair coin 20 times. (Block of $N = 20$, $p = 0.5$)
What's the probability of all heads?

Q2. What's the probability of '`TTHTTHHTTTHTTHTHHTTT`'?

Q3. What's the probability of 7 heads and 13 tails?

you'll be waiting forever	**A**	$\approx 10^{-100}$
about one in a million	**B**	$\approx 10^{-6}$
about one in ten	**C**	$\approx 10^{-1}$
about a half	**D**	≈ 0.5
very probable	**E**	$\approx 1 - 10^{-6}$
don't know	**Z**	???

Binomial distribution

How many 1's will be in our block?

Binomial distribution, the sum of N Bernoulli outcomes

$$k = \sum_{n=1}^{N} x_n, \quad x_n \sim \text{Bernoulli}(p)$$

$$\Rightarrow \ k \sim \text{Binomial(N, p)}$$

$$P(k \,|\, N, p) = \binom{N}{k} p^k (1-p)^{N-k}$$

$$= \frac{N!}{(N-k)!\,k!} p^k (1-p)^{N-k}$$

Reviewed by MacKay, p1

Distribution over blocks

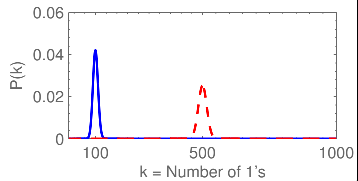
total number of bits: N (= 1000 in examples here)
probability of a 1: $p = P(x_i = 1)$
number of 1's: $k = \sum_i x_i$

Every block is improbable!

$$P(\mathbf{x}) = p^k (1-p)^{N-k}, \quad \text{(at most } (1-p)^N \approx 10^{-45} \text{ for } p = 0.1)$$

How many 1's will we see?

$$P(k) = \binom{N}{k} p^k (1-p)^{N-k}$$



Solid: $p = 0.1$

Dashed: $p = 0.5$

Intuitions: If we sample uniformly at random, the number of 1s is distributed according to the dashed curve. That bump is where almost all of the bit-strings of length $N\!=\!1000$ are.

When $p\!=\!0.1$, the blocks with the most zeros are the most probable. However, there is only one block with zero ones, and not many with only a few ones. As a result, there isn't much probability mass on states with only a few ones. In fact, most of the probability mass is on blocks with around Np ones, so they are the ones we are likely to see. The most probable block is not a typical block, and we'd be surprised to see it!

Evaluating the numbers

$$\binom{N}{k} = \frac{N!}{(N-k)!\,k!}\,,$$ what happens for $N\!=\!1000$, $k\!=\!500$?
(or $N\!=\!10{,}000$, $k\!=\!5{,}000$)

Knee-jerk reaction: try taking logs

Explicit summation: $\log x! = \sum_{n=2}^{x} \log n$

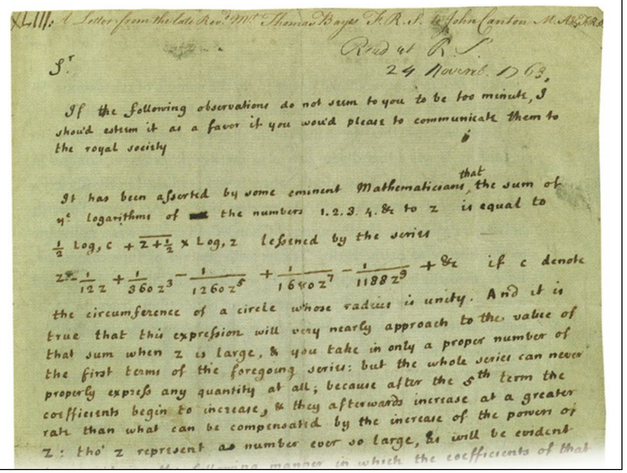
Library routines: $\ln x! = \ln \Gamma(x+1)$, e.g. `gammaln`

Stirling's approx: $\ln x! \approx x \ln x - x + \frac{1}{2}\ln 2\pi x \;\ldots$

Care: Stirling's series gets *less* accurate if you add lots terms(!), but the relative error disappears for large x with just the terms shown.

There is also (now) a convergent version (see Wikipedia).

See also: more specialist routines. Matlab/Octave: `binopdf`, `nchoosek`


XLIII. A Letter from the late Revd Mr Thomas Bayes F.R.S. to John Canton M.A. & F.R.S.

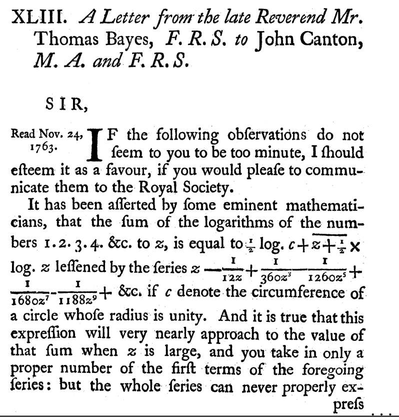
Read at R.S. 24 Novr. 1763.

Sr

If the following observations do not seem to you to be too minute, I should esteem it as a favor if you would please to communicate them to the royal society

It has been asserted by some eminent Mathematicians, that the sum of ye logarithms of the numbers 1. 2. 3. 4. &c to z is equal to ½ Log, c + z+½ × Log, z lessened by the series z − 1/12z + 1/360z³ − 1/1260z⁵ + 1/1680z⁷ − 1/1188z⁹ + &c if c denote the circumference of a circle whose radius is unity. And it is true that this expression will very nearly approach to the value of that sum when z is large, & you take in only a proper number of the first terms of the foregoing series: but the whole series can never properly express any quantity at all; because after the 5th term the coefficients begin to increase, & they afterwards increase at a greater rate than what can be compensated by the increase of the powers of z: tho' z represent as number ever so large, &c will be evident ...

Philosophical Transactions (1683-1775) Vol. 53, (1763), pp. 269–271.
The Royal Society. http://www.jstor.org/stable/105732

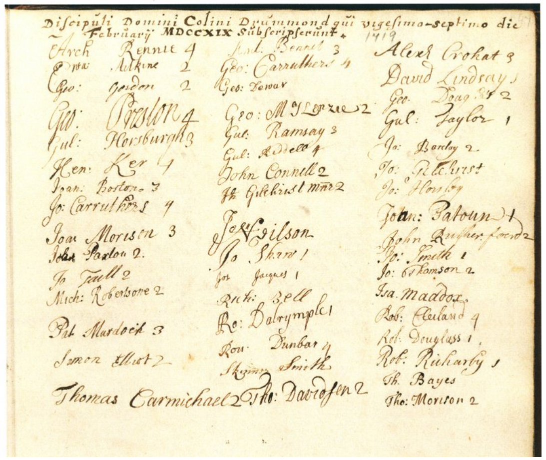

XLIII. *A Letter from the late Reverend Mr.* Thomas Bayes, *F. R. S. to* John Canton, *M. A. and F. R. S.*

SIR,

Read Nov. 24, 1763.

IF the following obſervations do not ſeem to you to be too minute, I ſhould eſteem it as a favour, if you would pleaſe to communicate them to the Royal Society.

It has been aſſerted by ſome eminent mathematicians, that the ſum of the logarithms of the numbers 1. 2. 3. 4. &c. to z, is equal to $\frac{1}{2}$ log. $c+z+\frac{1}{2}\times$ log. z leſſened by the ſeries $z-\frac{1}{12z}+\frac{1}{360z^3}-\frac{1}{1260z^5}+\frac{1}{1680z^7}-\frac{1}{1188z^9}+$ &c. if c denote the circumference of a circle whoſe radius is unity. And it is true that this expreſſion will very nearly approach to the value of that ſum when z is large, and you take in only a proper number of the firſt terms of the foregoing ſeries: but the whole ſeries can never properly expreſs . . .


Diſcipuli Domini Colini Drummond qui vigeſimo-septimo die Februarii MDCCXIX Subſcripſerunt. 1719

Familiarity with extreme numbers: when counting sets of possible strings or images the numbers are *enormous*. Similarly the probabilities of any such objects must be *tiny*, if the probabilities are to sum to one.

Learn to guess: before computing a number, force yourself to guess a rough range of where it will be. Guessing improves intuition and might catch errors.

Numerical experiments: we will derive the asymptotic behaviour of large block codes. Seeing how finite-sized blocks behave empirically is also useful. Take the logs of extreme positive numbers when implementing code.

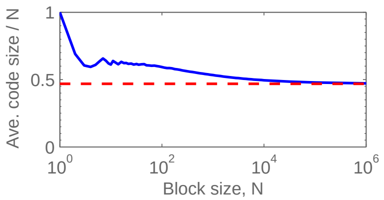
Bayes and Stirling's series: approximations of functions can be useful for analytically work. The images show copies of Bayes's letter about Stirling's series to John Canton, both handwritten and the original typeset version. Bayes studied at what became the University of Edinburgh. I've included a copy of a class list with his name (possibly not his signature) second from the end.

Compression for N-bit blocks

Strategy:
— Encode N-bit blocks with $\leq t$ ones with $C_1(t)$ bits.
— Use remaining codewords followed by $C_2(t)$ bits for other blocks.

Set $C_1(t)$ and $C_2(t)$ to minimum values required.

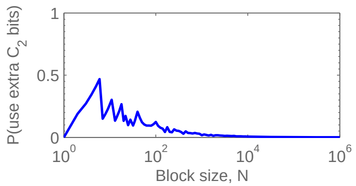
Set t to minimize average length: $C_1(t) + P\big(t < \sum_{n=1}^{N} x_n\big)\, C_2(t)$



Can we do better?

We took a simple, greedy strategy:
Assume one code-length C_1, add another C_2 bits if that doesn't work.

First observation for large N:
The first C_1 bits index almost every block we will see.



With high probability we can compress a large-N block into a fixed number of bits. Empirically $\approx 0.47\,N$ for $p\!=\!0.1$.

Can we do better?

We took a simple, greedy strategy:
Assume one code-length C_1, add another C_2 bits if that doesn't work.

Second observation for large N:
Trying to use $< C_1$ bits means we *always* use more bits

At $N\!=\!10^6$, trying to use 0.95 the optimal C_1 initial bits
$\Rightarrow P(\text{need more bits}) \approx 1 - 10^{-100}$

It is very unlikely a file can be compressed into fewer bits.

Summary of lecture 2 (slide 1/2)

If some files are shrunk others must grow:

\# files length b bits $= 2^b$
\# files $<b$ bits $= \sum_{c=0}^{b-1} 2^c = 1+2+4+8+\cdots+2^{b-1} = 2^b - 1$

(We'll see that things are even worse for encoding blocks in a stream.
Consider using bit strings up to length 2 to index symbols:
`A=0, B=1, C=00, D=01, E=11`
If you receive `111`, what was sent? `BBB`, `BE`, `EB`?)

We temporarily focus on sparse binary files:

Encode blocks of N bits, $\mathbf{x}$ =`00010000001000...000`

Assume model: $P(\mathbf{x}) = p^k(1-p)^{N-k}$, where $k = \sum_i x_i =$ "# 1's"

Key idea: give short encoding to most probable blocks:
Most probable block has $k=0$. Next N most probable blocks have $k=1$
Let's encode all blocks with $k \le t$, for some threshold t.
This set has $I_1 = \sum_{k=0}^{t} \binom{N}{k}$ items. Can index with $C_1 = \lceil \log_2 I_1 \rceil$ bits.

Summary of lecture 2 (slide 2/2)

Can make a lossless compression scheme:
Actually transmit $C_1 = \lceil \log_2(I_1+1) \rceil$ bits
Spare code word(s) are used to signal C_2 more bits should be read, where $C_2 \le N$ can index the other blocks with $k>t$.
Expected/average code length $= C_1 + P(k>t)\, C_2$

Empirical results for large block-lengths N
— The best codes (best t, C_1, C_2) had code length $\approx 0.47N$
— these had tiny $P(k>t)$; it doesn't matter how we encode $k>t$
— Setting $C_1 = 0.95 \times 0.47N$ made $P(k>t) \approx 1$

$\approx 0.47N$ bits are sufficient and necessary to encode long blocks (with our model, $p=0.1$) almost all the time and on average

No scheme can compress binary variables with $p=0.1$ into less than 0.47 bits on average, or we could contradict the above result.

Other schemes will be more practical (they'd better be!) and will be closer to the $0.47N$ limit for small N.

Information Theory

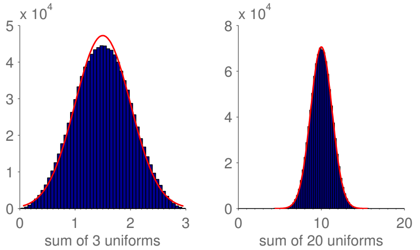
http://www.inf.ed.ac.uk/teaching/courses/it/

Week 2
Information and Entropy

Iain Murray, 2013
School of Informatics, University of Edinburgh

Central Limit theorem

The sum or mean of independent variables with bounded mean and variance tends to a Gaussian (normal) distribution.




```
N=1e6;  hist(sum(rand(3,N),1));  hist(sum(rand(20,N),1));
```

There are a few forms of the Central Limit Theorem (CLT), we are just noting a vague statement as we won't make extensive use of it.

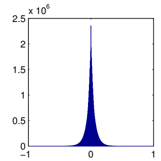
CLT behaviour can occur unreasonably quickly when the assumptions hold. Some old random-number libraries used to use the following method for generating a sample from a unit-variance, zero-mean Gaussian: a) generate 12 samples uniformly between zero and one; b) add them up and subtract 6. It isn't that far off!

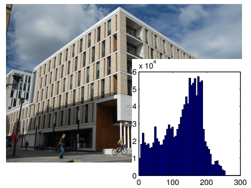
Data from a natural source will usually *not* be Gaussian.
The next slide gives examples. Reasons: extreme outliers often occur; there may be lots of strongly dependent variables underlying the data; there may be mixtures of small numbers of effects with very different means or variances.

An example random variable with unbounded mean is given by the payout of the game in the *St. Petersburg Paradox*. A fair coin is tossed repeatedly until it comes up tails. The game pays out $2^{\#\text{heads}}$ pounds. How much would you pay to play? The 'expected' payout is infinite: $\frac{1}{2}\times1 + \frac{1}{4}\times2 + \frac{1}{8}\times4 + \frac{1}{16}\times8 + \ldots = \frac{1}{2}+\frac{1}{2}+\frac{1}{2}+\frac{1}{2}+\ldots$

Gaussians are not the only fruit

```
xx = importdata('Holst_-_Mars.wav');
hist(double(xx(:)), 400);
```






```
xx = importdata('forum.jpg');
hist(xx(:), 50);
```

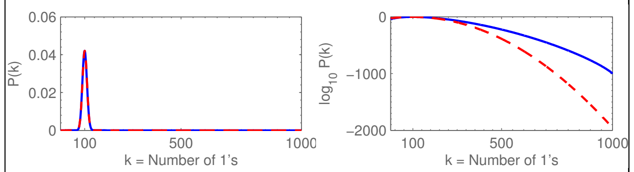
How many 1's will we see?

How many 1's will we see? $P(k) = \binom{N}{k} p^k (1-p)^{N-k}$

Gaussian fit (dashed lines):

$P(k) \approx \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(k-\mu)^2\right), \quad \mu = Np, \quad \sigma^2 = Np(1-p)$

(Binomial mean and variance, MacKay p1)



The log-probability plot on the previous slide illustrates how one must be careful with the Central Limit Theorem. Even though the assumptions hold, convergence of the tails is very slow. (The theory gives only "*convergence in distribution*" which makes weak statements out there.) While k, the number of ones, closely follows a Gaussian near the mean, we can't use the Gaussian to make precise statements about the tails.

All that we will use for now is that the mass in the tails further out than a few standard deviations (a few σ) will be small. This is correct, we just can't guarantee that the probability will be quite as small as if the whole distribution actually were Gaussian.

Chebyshev's inequality (MacKay p82, Wikipedia, ...) tells us that:
$P(|k-\mu| \ge m\sigma) \le \frac{1}{m^2}$,
a loose bound which will be good enough for what follows.

The fact that as $N \to \infty$ all of the probability mass becomes close to the mean is referred to as the *law of large numbers*.

A weighing problem

Find 1 odd ball out of 12

You have a two-pan balance with three outputs:
"left-pan heavier", "right-pan heavier", or "pans equal"

How many weighings do you need to find the odd ball *and* decide whether it is heavier or lighter?

Unclear? See p66 of MacKay's book, but do not look at his answer until you have had a serious attempt to solve it.

Are you sure your answer is right? Can you prove it?
Can you prove it without an extensive search of the solution space?

Weighing problem: bounds

Find 1 odd ball out of 12 with a two-pan balance

There are 24 hypothesis:
ball 1 heavier, ball 1 lighter, ball 2 heavier, . . .

For K weighings, there are at most 3^K outcomes:
(left, balance, right), (right, right, left), . . .

$3^2 = 9 \;\Rightarrow\;$ 2 weighings not enough
$3^3 = 27 \;\Rightarrow\;$ 3 weighings *might* be enough

Analogy: sorting (review?)

How much does it cost to sort n items?

There are 2^C outcomes of C binary comparisons

There are $n!$ orderings of the items

To pick out the correct ordering must have:

$C \log 2 \geq \log n! \quad \Rightarrow \quad C \geq \mathcal{O}(n \log n)$ (Stirling's series)

Radix sort is "$\mathcal{O}(n)$", gets more information from the items

Weighing problem: strategy

Find 1 odd ball out of 12 with a two-pan balance

Probability of an outcome is: $\frac{\text{\# hypotheses compatible with outcome}}{\text{\# hypotheses}}$

Experiment	Left	Right	Balance
1 vs. 1	2/24	2/24	20/24
2 vs. 2	4/24	4/24	16/24
3 vs. 3	6/24	6/24	12/24
4 vs. 4	8/24	8/24	8/24
5 vs. 5	10/24	10/24	4/24
6 vs. 6	12/24	12/24	0/24

Weighing problem: strategy

8 hypotheses remain. Find a second weighing where:

3 hypotheses ⇒ left pan down
3 hypotheses ⇒ right pan down
2 hypotheses ⇒ balance

It turns out we can always identify one hypothesis with a third weighing (p69 MacKay for details)

Intuition: outcomes with even probability distributions seem *informative* — useful to identify the correct hypothesis

Measuring information

As we read a file, or do experiments, we get **information**

Very probable outcomes are not informative:
⇒ Information is zero if $P(x)=1$
⇒ Information increases with $1/P(x)$

Information of two independent outcomes add
$\Rightarrow f\big(\frac{1}{P(x)P(y)}\big) = f\big(\frac{1}{P(x)}\big) + f\big(\frac{1}{P(y)}\big)$

Shannon information content: $h(x) = \log \frac{1}{P(x)} = -\log P(x)$

The base of the logarithm scales the information content:
base 2: bits
base e: nats
base 10: bans (used at Bletchley park: MacKay, p265)

$\log \frac{1}{P}$ is the only 'natural' measure of information based on probability alone. Derivation non-examinable.

Assume: $f(ab) = f(a) + f(b)$; $f(1) = 0$; f smoothly increases

$f(a(1+\epsilon)) = f(a) + f(1+\epsilon)$

Take limit $\epsilon \to 0$ on both sides:

$f(a) + a\epsilon f'(a) = f(a) + \cancelto{0}{f(1)} + \epsilon f'(1)$

$\Rightarrow f'(a) = f'(1)\frac{1}{a}$

$\int_1^x f'(a)\,\mathrm{d}a = f'(1)\int_1^x \frac{1}{a}\,\mathrm{d}a$

$f(x) = f'(1)\ln x$

Define $b = e^{1/f'(1)}$, which must be >1 as f is increasing.

$f(x) = \log_b x$

We can choose to measure information in any base (>1), as the base is not determined by our assumptions.

Foundations of probability *(very non-examinable)*

The main step justifying information resulted from $P(a,b) = P(a)\,P(b)$ for independent events. Where did *that* come from?

There are various formulations of probability. Kolmogorov provided a measure-theoretic formalization for frequencies of events.

Cox (1946) provided a very readable rationalization for using the standard rules of probability to express beliefs and to incorporate knowledge: `http://dx.doi.org/10.1119/1.1990764`

There's some (I believe misguided) arguing about the details. A sensible response to some of these has been given by Van Horn (2003) `http://dx.doi.org/10.1016/S0888-613X(03)00051-3`

Ultimately for both information and probability, the main justification for using them is that they have proven to be hugely useful. While one can argue forever about choices of axioms, I don't believe that there are other compelling formalisms to be had for dealing with uncertainty and information.

Information content vs. storage

A 'bit' is a symbol that takes on two values.
The 'bit' is also a unit of information content.

Numbers in 0–63, e.g. $47 = 101111$, need $\log_2 64 = 6$ bits

If numbers 0–63 are equally probable, being told the number has information content $-\log \frac{1}{64} = 6$ bits

The binary digits are the answers to six 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 $x \bmod 2 = 1$?

Each question has information content $-\log \frac{1}{2} = 1$ bit

Fractional information

A dull guessing game: (submarine, MacKay p71)

Q. Is the number 36?
A. a_1 = No.
$h(a_1) = \log \frac{1}{P(x \neq 36)} = \log \frac{64}{63} = 0.0227$ bits

Remember: $\log_2 x = \frac{\ln x}{\ln 2}$

Q. Is the number 42?
A. a_2 = No.
$h(a_2) = \log \frac{1}{P(x \neq 42 \,|\, x \neq 36)} = \log \frac{63}{62} = 0.0231$ bits

Q. Is the number 47?
A. a_3 = Yes.
$h(a_3) = \log \frac{1}{P(x = 47 \,|\, x \neq 42, x \neq 36)} = \log \frac{62}{1} = 5.9542$ bits

Total information: $5.9542 + 0.0231 + 0.0227 = 6$ bits

Entropy

Improbable events are very informative, but don't happen very often! How much information can we *expect*?

Discrete sources:

Ensemble: $X = (x, \mathcal{A}_X, \mathcal{P}_X)$
Outcome: $x \in \mathcal{A}_x, \quad p(x\!=\!a_i) = p_i$
Alphabet: $\mathcal{A}_X = \{a_1, a_2, \ldots, a_i, \ldots a_I\}$
Probabilities: $\mathcal{P}_X = \{p_1, p_2, \ldots, p_i, \ldots p_I\}, \qquad p_i > 0, \quad \sum_i p_i = 1$

Information content:

$h(x\!=\!a_i) = \log \frac{1}{p_i}, \qquad h(x) = \log \frac{1}{P(x)}$

Entropy:

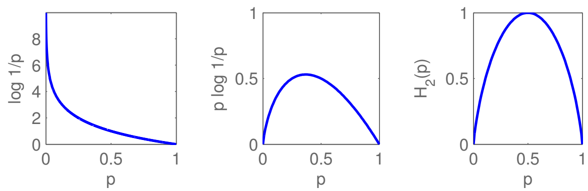
$H(X) = \sum_i p_i \log \frac{1}{p_i} = \mathbb{E}_{\mathcal{P}_X}[h(x)]$

average information content of source, also "the uncertainty of X"

Binary Entropy

Entropy of Bernoulli variable:

$$H(X) = H_2(p) = p_1 \log \frac{1}{p_1} + p_2 \log \frac{1}{p_2}$$
$$= -p \log p - (1-p) \log (1-p)$$



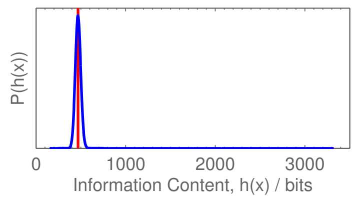
Plots take logs base 2. We define $0 \log 0 = 0$

Distribution of Information

Extended Ensemble X^N: N independent draws from X

$\mathbf{x}$ a length-N vector containing a draw from X^N

Bernoulli example: $N = 10^3$, $p = 0.1$, $H(X) = 0.47$ bits



The information content of each element, $h(x_n)$, is a random variable. This variable has mean $H(X)$, and some finite variance σ^2.

Mean and width of the curve: The total information content of a block: $h(\mathbf{x}) = \sum_n h(x_n)$ is another random variable with mean $NH(X)$, shown in red, and variance $N\sigma^2$ or standard deviation $\sqrt{N}\sigma$. (All of the above is true for general extended ensembles, not just binary streams.)

The range of the plot: The block with maximum information content is the most surprising, or least probable block. In the Bernoulli example with $p\!=\!0.1$, '`1111...111`' is most surprising, with $h(\mathbf{x})\!=\!Nh(1)\!=\!N \log \frac{1}{0.1}$. Similarly the least informative block, is the most probable. In the example $Nh(0)\!=\!N \log \frac{1}{0.9}$. Remember to take logs base 2 to obtain an answer in bits. Neither of these blocks will *ever* be seen in practice, even though `0000...000` *is* the most probable block.

Only blocks with information contents close to the mean are 'typical'.

Define the *typical set*, T, to be all blocks with information contents a few standard deviations away from the mean:

$h(\mathbf{x}) \in [NH\!-\!m\sigma\sqrt{N}, NH\!+\!m\sigma\sqrt{N}]$ for some $m > 0$.

(Actually a family of typical sets for different choices of m.)

We only need to count the typical set: Chebyshev's inequality (see MacKay p82, Wikipedia, ...) bounds the probability that we land outside the typical set.

$$P(|h(\mathbf{x}) - NH| \geq m\sigma\sqrt{N}) \leq \tfrac{1}{m^2}$$

We can pick m so that the typical set is so large that the probability of landing outside it is negligible. Then we can compress almost every file we see into a number of bits that can index the typical set.

How big is the typical set? Number of elements: $|T|$

Probability of landing in set ≤ 1
Probability of landing in set $\geq |T| p_{\min}$, where $p_{\min} = \min_{\mathbf{x} \in T} p(\mathbf{x})$
Therefore, $|T| < \frac{1}{p_{\min}}$

Block with smallest probability $p_{\min}$ has information $NH + m\sigma\sqrt{N}$.

$p_{\min} = 2^{-NH - m\sigma\sqrt{N}}$
$|T| < 2^{NH + m\sigma\sqrt{N}}$

Number of bits to index typical set is $NH + m\sigma\sqrt{N}$.
Dividing by the block length, N we see we need:

$$H + m\sigma/\sqrt{N} \text{ bits/symbol}, \rightarrow H \text{ as } N \rightarrow \infty$$

For any choice m, in the limit of large blocks, we can encode the typical set (and for large enough m, any file we will see in practice) with $H(X)$ bits/symbol.

Can we do better?

Motivation: The above result put a loose bound on the probability of being outside T, so we might have made it bigger than necessary. Then we put a loose bound on the number of items, so we assumed it was even bigger than that. *Maybe* we could use many fewer bits per symbol than we calculated? (Amazingly, the answer is that we can't.)

We assume there is a smaller useful set S, which we could encode with only $(1\!-\!\epsilon)H$ bits/symbol. For example, if $\epsilon\!=\!0.01$ we would be trying to get a 1% saving in the number of bits for strings in this set.

The size of S is $|S| = 2^{N(1-\epsilon)H}$

Some of S will overlap with T, and some might be outside. But we know that the total probability outside of T is negligible (for large m). The probability mass of elements inside T is less than $|S| p_{\max}$, where $p_{\max}$ is the probability of the largest probability element of T.

$p_{\max} = 2^{-NH + m\sigma\sqrt{N}}$
$p(\mathbf{x} \in S) \leq |S| p_{\max} + \text{tail mass outside } T$
$p(\mathbf{x} \in S) \leq 2^{N(-\epsilon H + m\sigma/\sqrt{N})} + \text{tail mass outside } T$

As $N \rightarrow \infty$ the probability of getting a block in S tends to zero for any m. The smaller set is useless.

At least H bits/symbol are required to encode an extended ensemble.

On average, no compressor can use fewer than H bits per symbol (applied to length-N blocks, it wouldn't be using enough bits)

Where now?

A block of variables can be compressed into $H(X)$ bits/symbol, but no less

Where do we get the probabilities from?

How do we *actually* compress the files?
We can't explicitly list 2^{NH} items!
Can we avoid using enormous blocks?

Numerics note: $\log \sum_i \exp(x_i)$

$\binom{N}{k}$ blows up for large N, k; we evaluate $l_{N,k} = \ln \binom{N}{k}$

Common problem: want to find a sum, like $\sum_{k=0}^{t} \binom{N}{k}$

Actually we want its log:

$$\ln \sum_{k=0}^{t} \exp(l_{N,k}) = l_{\max} + \ln \sum_{k=0}^{t} \exp(l_{N,k} - l_{\max})$$

To make it work, set $l_{\max} = \max_k l_{N,k}$. `logsumexp` functions are frequently used

I needed this trick when numerically exploring block codes:

For a range of t we needed to sum up: a) the number of strings with $k = 0..t$; and b) the probability mass associated with those strings. The log of the number of strings says how many bits, C_1 was needed to index them. If the probability mass is close to one, that will also be close to the expected length needed to encode random strings.

For both sums we need the log of the sum of some terms, where each term is available in log form. The next slide demonstrates this for problem a), but the technique readily applies to problem b) too.

The bumps are very well behaved: to what extent can we assume they are Gaussian due to central limit arguments?

Information Theory

http://www.inf.ed.ac.uk/teaching/courses/it/

Week 3
Symbol codes

Iain Murray, 2012

School of Informatics, University of Edinburgh

(Binary) Symbol Codes

For strings of symbols from alphabet e.g.,
$x_i \in \mathcal{A}_X = \{A, C, G, T\}$

Binary codeword assigned to each symbol

$CGTAGATTACAGG$

$\downarrow$

10111110011101101100100111111

A	0
C	10
G	111
T	110

Codewords are concatenated without punctuation

Uniquely decodable

We'd like to make all codewords short

But some codes are not **uniquely decodable**

$CGTAGATTACAGG$

$\downarrow$

111111001110110110010111111

$\downarrow$

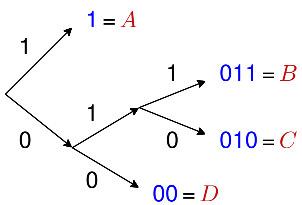
$CGTAGATTACAGG$
$CCCCCCAACCCACCACCAACACCCCCC$
$CCGCAACCCATCCAACAGCCC$
$GGAAGATTACAGG$
???

A	0
C	1
G	111
T	110

Instantaneous/Prefix Codes

Attach symbols to leaves of a binary tree

Codeword gives path to get to leaf



"Prefix code" because no codeword is a prefix of another

Decoding: follow tree while reading stream until hit leaf

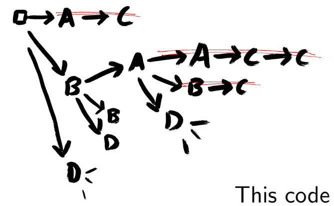
Symbol is *instantly* identified. Return to root of tree.

Non-instantaneous Codes

The last code was **instantaneously decodable**:
We knew as soon as we'd finished receiving a symbol

101100000101100

$\downarrow$



A	1
B	10
C	000
D	100

This code *is* uniquely decodable, but not instantaneous or pleasant!

Expected length/symbol, $\bar{L}$

Code lengths: $\{\ell_i\} = \{\ell_1, \ell_2, \ldots, \ell_I\}$

$$\text{Average, } \bar{L} = \sum_i p_i \ell_i$$

Compare to Entropy:

$$H(X) = \sum_i p_i \log \frac{1}{p_i}$$

If $\ell_i = \log \frac{1}{p_i}$ **or** $p_i = 2^{-\ell_i}$ **we compress to the entropy**

An optimal symbol code

An example code with:

$$\bar{L} = \sum_i p_i \ell_i = H(X) = \sum_i p_i \log \frac{1}{p_i}$$

x	$p(x)$	codeword
A	$1/2$	0
B	$1/4$	10
C	$1/8$	110
D	$1/8$	111

Entropy: decomposability

Flip a coin:
Heads $\rightarrow A$
Tails $\rightarrow$ **flip again:**
Heads $\rightarrow B$
Tails $\rightarrow C$

$\mathcal{A}_X = \{A,\ B,\ C\}$
$\mathcal{P}_X = \{0.5,\ 0.25,\ 0.25\}$

$H(X) = 0.5 \log \frac{1}{0.5} + 0.25 \log \frac{1}{0.25} + 0.25 \log \frac{1}{0.25} = 1.5$ bits

Or: $H(X) = H_2(0.5) + 0.5\, H_2(0.5) = 1.5$ bits

Shannon's 1948 paper §6. MacKay §2.5, p33

Why look at the decomposability of Entropy?

Mundane, but useful: it can make your algebra a lot neater. Decomposing computations on graphs is ubiquitous in computer science.

Philosophical: we expect that the expected amount of information from a source should be the same if the same basic facts are represented in different ways and/or reported in a different order.

Shannon's 1948 paper used the desired decomposability of entropy to derive what form it must take, section 6. This is similar to how we intuited the information content from simple assumptions.

Limit on code lengths

Imagine coding under an implicit distribution:

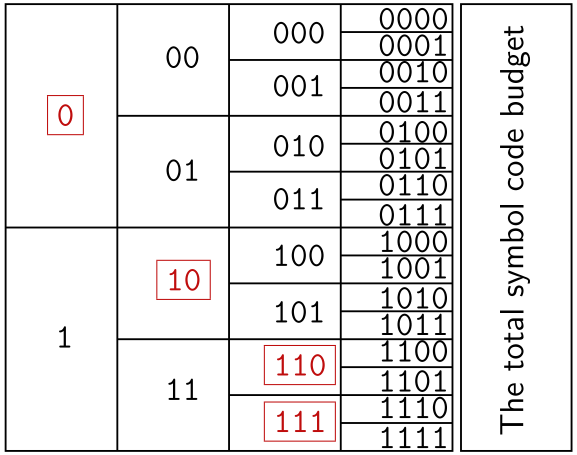
$$q_i = \frac{1}{Z} 2^{-\ell_i}, \quad Z = \sum_i 2^{-\ell_i}.$$

$$H = \sum_i q_i \log \frac{1}{q_i} = \sum_i q_i \left(\ell_i + \log Z\right) = \bar{L} + \log Z$$

$$\Rightarrow \log Z \leq 0, \quad Z \leq 1$$

Kraft–McMillan Inequality $\boxed{\sum_i 2^{-\ell_i} \leq 1}$ (if uniquely-decodable)

Proof without invoking entropy bound: p95 of MacKay, or p116 Cover & Thomas 2nd Ed.



Kraft Inequality

If height of budget is 1, codeword has height $= 2^{-\ell_i}$

Pick codes of required lengths in order from shortest–largest

Choose heighest codeword of required length beneath previously-chosen code (There won't be a gap because of sorting)

Can always pick codewords if total height, $\sum_i 2^{-\ell_i} \leq 1$

Kraft–McMillan Inequality $\boxed{\sum_i 2^{-\ell_i} \leq 1}$ (instantaneous code possible)

Corollary: there's probably no point using a non-instantaneous code.
Can always make **complete code** $\sum_i 2^{-\ell_i} = 1$: slide last codeword left.

Summary of Lecture 5

Symbol codes assign each symbol in an alphabet a codeword.
(We only considered binary symbol codes, which have binary codewords.)
Messages are sent by concatenating codewords with no punctuation.

Uniquely decodable: the original message is unambiguous

Instantaneously decodable: the original symbol can always be determined as soon as the last bit of its codeword is received.

Codeword lengths must satisfy $\sum_i 2^{-\ell_i} \leq 1$ for unique decodability

Instantaneous prefix codes can always be found (if $\sum_i 2^{-\ell_i} \leq 1$)

Complete codes have $\sum_i 2^{-\ell_i} = 1$, as realized by prefix codes made from binary trees with a codeword at every leaf.

If (big if) symbols are drawn i.i.d. with probabilities $\{p_i\}$, *and* $\ell_i = \log \frac{1}{p_i}$, then a prefix code exists that offers optimal compression.

Next lecture: how to form the best symbol code when $\{\log \frac{1}{p_i}\}$ are not integers.

Performance of symbol codes

Simple idea: set $\ell_i = \left\lceil \log \frac{1}{p_i} \right\rceil$

These codelengths satisfy the Kraft inequality:

$$\sum_i 2^{-\ell_i} = \sum_i 2^{-\lceil \log 1/p_i \rceil} \leq \sum_i p_i = 1$$

Expected length, $\bar{L}$:

$$\bar{L} = \sum_i p_i \ell_i = \sum_i p_i \lceil \log 1/p_i \rceil < \sum_i p_i \left(\log 1/p_i + 1\right)$$

$$\bar{L} < H(\mathbf{p}) + 1$$

Symbol codes can compress to within 1 bit/symbol of the entropy.

Optimal symbol codes

Encode independent symbols with known probabilities:

E.g., $\mathcal{A}_X = \{A, B, C, D, E\}$
$\mathcal{P}_X = \{0.3, 0.25, 0.2, 0.15, 0.1\}$

We can do better than $\ell_i = \left\lceil \log \frac{1}{p_i} \right\rceil$

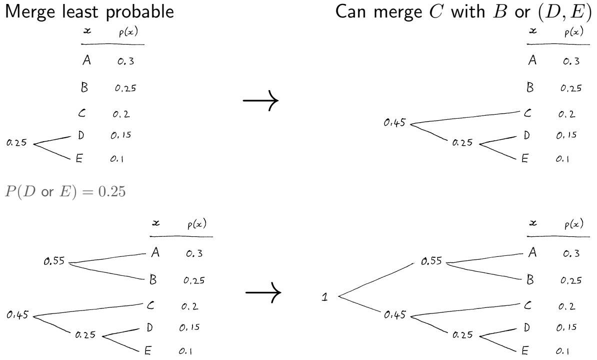
The *Huffman algorithm* gives an optimal symbol code.

Proof: MacKay Exercise 5.16 (with solution).
Cover and Thomas has another version.

Huffman algorithm

Merge least probable

Can merge C with B or (D, E)

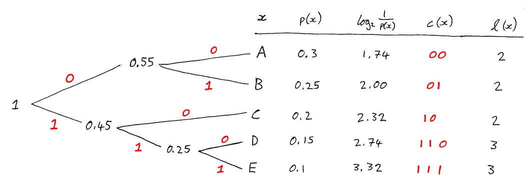


$P(D \text{ or } E) = 0.25$

Continue merging least probable, until root represents all events $P = 1$

Huffman algorithm

Given a tree, label branches with 1s and 0s to get code



Code-lengths are close to the information content
(not just rounded up, some are shorter)

$H(X) \approx 2.23$ bits. Expected length $= 2.25$ bits.

Wow! Despite limitations we will discuss, Huffman codes can be very good. You'll find them inside many systems (e.g., bzip2, jpeg, mp3), although all these schemes do clever stuff to come up with a good symbol representation.

Huffman decoding

Huffman codes are easily and uniquely decodable because they are prefix codes

Reminder on decoding a prefix code stream:
- — Start at root of tree
- — Follow a branch after reading each bit of the stream
- — Emit a symbol upon reaching a leaf of the tree
- — Return to the root after emitting a symbol. . .

An input stream can only give one symbol sequence, the one that was encoded

Building prefix trees 'top-down'

Heuristic: if you're ever building a tree, consider top-down vs. bottom-up (and maybe middle-out)

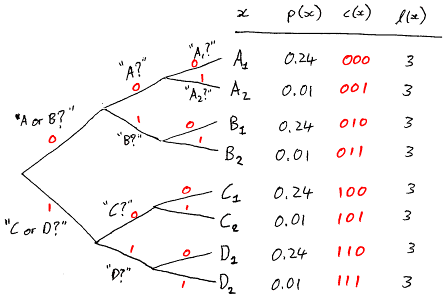
Weighing problem strategy:
Use questions with nearly uniform distribution over the answers.

How well would this work on the ensemble to the right?

x	$P(x)$
A_1	0.24
A_2	0.01
B_1	0.24
B_2	0.01
C_1	0.24
C_2	0.01
D_1	0.24
D_2	0.01

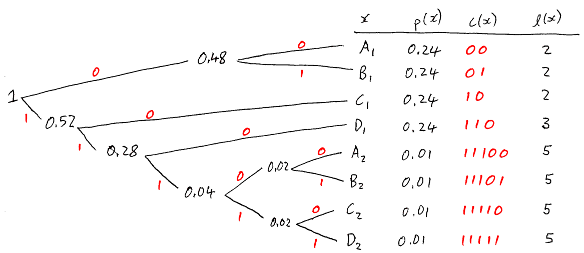
$H(X) = 2.24$ bits (just over $\log 4 = 2$). Fixed-length encoding: 3 bits

Top-down performing badly



Probabilities for answers to first two questions is $(1/2, 1/2)$
Greedy strategy $\Rightarrow$ very uneven distribution at end

Compare to Huffman



Expected length 2.36 bits/symbol

(Symbols reordered for display purposes only)

Relative Entropy / KL

Implicit probabilities: $q_i = 2^{-\ell_i}$
($\sum_i q_i = 1$ because Huffman codes are complete)

Extra cost for using "wrong" probability distribution:

$$\begin{aligned}\Delta L &= \sum_i p_i \ell_i - H(X)\\ &= \sum_i p_i \log 1/q_i - \sum_i p_i \log 1/p_i\\ &= \sum_i p_i \log \frac{p_i}{q_i} \;=\; D_{\mathrm{KL}}(p\,||\,q)\end{aligned}$$

$D_{\mathrm{KL}}(p\,||\,q)$ **is the *Relative Entropy* also known as the *Kullback–Leibler divergence* or *KL-divergence***

Gibbs' inequality

An important result:

$$D_{\mathrm{KL}}(p\,||\,q) \geq 0$$

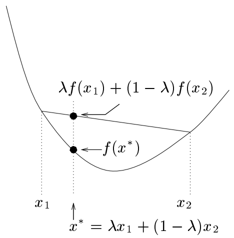
with equality only if $p = q$

"If we encode with the wrong distribution we will do worse than the fundamental limit given by the entropy"

A simple direct proof can be shown using convexity.
(Jensen's inequality)

Convexity

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



Strictly convex functions:
Equality only if λ is 0 or 1, or if $x_1 = x_2$
(non-strictly convex functions contain straight line segments)

Convex vs. Concave

For (strictly) concave functions reverse the inequality

A (con)cave

Photo credit: Kevin Krejci on Flickr

Summary of Lecture 6

The **Huffman Algorithm** gives optimal symbol codes:
Merging event adds to code length for children, so
Huffman always merges least probable events first

A complete code implies negative log probabilities: $q_i = 2^{-\ell_i}$.
If the symbols are generated with these probabilities, the symbol code compresses to the entropy. Otherwise the number of extra bits/symbol is given by the **Relative Entropy** or **KL-divergence**: $D_{\mathrm{KL}}(p\,||\,q) = \sum_i p_i \log \frac{p_i}{q_i}$

Gibbs' inequality says $D_{\mathrm{KL}}(p\,||\,q) \geq 0$ with equality only when the distributions are equal.

Convexity and Concavity are useful properties when proving several inequalities in Information Theory. Next time: the basis of these proofs is **Jensen's inequality**, which can be used to prove Gibbs' inequality.

Information Theory

http://www.inf.ed.ac.uk/teaching/courses/it/

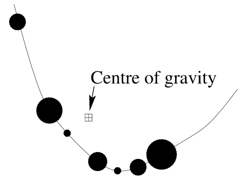
Week 4
Compressing streams

Iain Murray, 2012

School of Informatics, University of Edinburgh

Jensen's inequality

For convex functions: $\mathbb{E}[f(x)] \geq f(\mathbb{E}[x])$



Centre of gravity at $\big(\mathbb{E}[x], \mathbb{E}[f(x)]\big)$, which is above $\big(\mathbb{E}[x], f(\mathbb{E}[x])\big)$

Strictly convex functions:
Equality only if $P(x)$ puts all mass on one value

Remembering Jensen's

The inequality is reversed for concave functions.

Which way around is the inequality?

I draw a picture in the margin.

Alternatively, try 'proof by example':

$f(x) = x^2$ is a convex function

$\mathrm{var}[X] = \mathbb{E}[x^2] - \mathbb{E}[x]^2 \geq 0$

So Jensen's must be: $\mathbb{E}[f(x)] \geq f(\mathbb{E}[x])$ for convex f.

Jensen's: Entropy & Perplexity

Set $u(x) = \dfrac{1}{p(x)}$, $\;p(u(x)) = p(x)$

$\mathbb{E}[u] = \mathbb{E}[\frac{1}{p(x)}] = |\mathcal{A}|$ (Tutorial 1 question)

$H(X) = \mathbb{E}[\log u(x)] \leq \log \mathbb{E}[u]$

$H(X) \leq \log |\mathcal{A}|$

Equality, maximum Entropy, for constant $u \Rightarrow$ uniform $p(x)$

$2^{H(X)}$ = "Perplexity" = "Effective number of choices"

Maximum effective number of choices is $|\mathcal{A}|$

Proving Gibbs' inequality

Idea: use Jensen's inequality

For the idea to work, the proof must look like this:

$$D_{\mathrm{KL}}(p\,||\,q) = \sum_i p_i \log \frac{p_i}{q_i} = \mathbb{E}[f(u)] \geq f\big(\mathbb{E}[u]\big)$$

Define $u_i = \frac{q_i}{p_i}$, with $p(u_i) = p_i$, giving $\mathbb{E}[u] = 1$

Identify $f(x) \equiv \log {}^1\!/\!_x = -\log x$, a convex function

Substituting gives: $\boxed{D_{\mathrm{KL}}(p\,||\,q) \geq 0}$

Huffman code worst case

Previously saw: simple simple code $\ell_i = \lceil \log {}^1\!/\!_{p_i} \rceil$

Always compresses with $\mathbb{E}[\text{length}] < H(X){+}1$

Huffman code can be this bad too:

For $\mathcal{P}_X = \{1{-}\epsilon, \epsilon\}$, $\;H(x) \to 0\;$ as $\;\epsilon \to 0$

Encoding symbols independently means $\mathbb{E}[\text{length}] = 1$.

Relative encoding length: $\mathbb{E}[\text{length}]/H(X) \to \infty$ (!)

Question: can we fix the problem by encoding blocks?
$H(X)$ is $\log$(effective number of choices)
With many typical symbols the "+1" looks small

Reminder on Relative Entropy and symbol codes:

The Relative Entropy (AKA Kullback–Leibler or KL divergence) gives the expected extra number of bits per symbol needed to encode a source when a complete symbol code uses implicit probabilities $q_i = 2^{-\ell_i}$ instead of the true probabilities p_i.

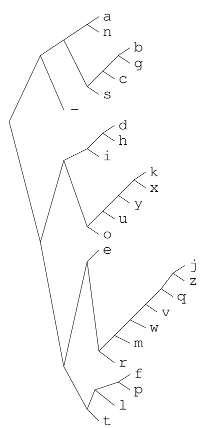
We have been assuming symbols are generated i.i.d. with known probabilities p_i.

Where would we get the probabilities p_i from if, say, we were compressing text? A simple idea is to read in a large text file and record the empirical fraction of times each character is used. Using these probabilities the next slide (from MacKay's book) gives a Huffman code for English text.

The Huffman code uses 4.15 bits/symbol, whereas $H(X) = 4.11$ bits. Encoding blocks might close the narrow gap.

More importantly **English characters are not drawn independently** encoding blocks could be a better model.

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



(Mackay, p100)

Bigram statistics

Previous slide: $\mathcal{A}_X = \{\mathtt{a}-\mathtt{z}, \_\}$, $\;H(X) = 4.11$ bits

Question: I decide to encode bigrams of English text:
$\mathcal{A}_{X'} = \{\mathtt{aa}, \mathtt{ab}, \ldots, \mathtt{az}, \mathtt{a\_}, \ldots, \_\_\}$

What is $H(X')$ for this new ensemble?

A ~ 2 bits
B ~ 4 bits
C ~ 7 bits
D ~ 8 bits
E ~ 16 bits

Z ?

Answering the previous vague question

We didn't completely define the ensemble: what are the probabilities?

We could draw characters independently using p_i's found before. Then a bigram is just two draws from X, often written X^2. $H(X^2) = 2H(X) = 8.22$ bits

We could draw pairs of adjacent characters from English text. When predicting such a pair, how many effective choices do we have? More than when we had $\mathcal{A}_X = \{\mathtt{a}\text{–}\mathtt{z}, \_\}$: we have to pick the first character *and* another character. But the second choice is easier. We expect $H(X) < H(X') < 2H(X)$. Maybe 7 bits? Looking at a large text file the actual answer is about 7.6 bits. This is ≈ 3.8 bits/character — better compression than before.

Shannon (1948) estimated about 2 bits/character for English text. Shannon (1951) estimated about 1 bits/character for English text

Compression performance results from the quality of a probabilistic model and the compressor that uses it.

Human predictions

Ask people to guess letters in a newspaper headline:

k·i·d·s·_·m·a·k·e·_·n·u·t·r·i·t·i·o·u·s·_·s·n·a·c·k·s

11·4·2·1·1·4·2·4·1·1·15·5·1·2·1·1·1·1·2·1·1·16·7·1·1·1·1

Numbers show # guess required by 2010 class

⇒ "effective number of choices" or entropy varies *hugely*

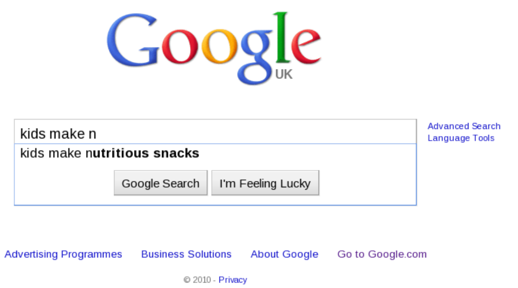
We need to be able to use a different probability distribution for every context

Sometimes many letters in a row can be predicted at minimal cost: need to be able to use < 1 bit/character.

(MacKay Chapter 6 describes how numbers like those above could be used to encode strings.)

Predictions

Cliché Predictions



A more boring prediction game

"I have a binary string with bits that were drawn i.i.d.. Predict away!"

What fraction of people, f, guess next bit is '1'?

Bit:	1	1	1	1	1	1	1	1
f:	$\approx 1/2$	$\approx 1/2$	$\approx 1/2$	$\approx 2/3$	...	...	...	≈ 1

The source was genuinely i.i.d.: each bit was independent of past bits.

We, not knowing the underlying flip probability, learn from experience. Our predictions depend on the past. So should our compression systems.

Arithmetic Coding

For better diagrams and more detail, see MacKay Ch. 6

Consider all possible strings in alphabetical order

(If infinities scare you, all strings up to some maximum length)

Example: $\mathcal{A}_X = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \boxtimes\}$

Where '$\boxtimes$' is a special `End-of-File` marker.

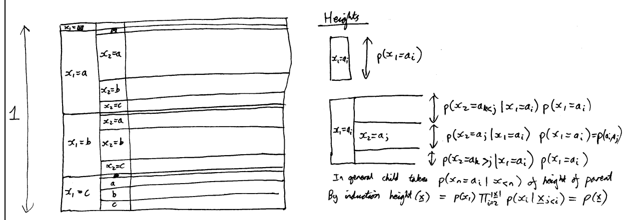
$\boxtimes$
a$\boxtimes$, aa$\boxtimes$, $\cdots$, ab$\boxtimes$, $\cdots$, ac$\boxtimes$, $\cdots$
b$\boxtimes$, ba$\boxtimes$, $\cdots$, bb$\boxtimes$, $\cdots$, bc$\boxtimes$, $\cdots$
c$\boxtimes$, ca$\boxtimes$, $\cdots$, cb$\boxtimes$, $\cdots$, cc$\boxtimes$, $\cdots$, cccccc. . . cc$\boxtimes$

Arithmetic Coding

We give all the strings a binary codeword

Huffman merged leaves — but we have too many to do that

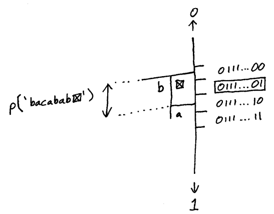
Create a tree of strings 'top-down':



Could keep splitting into *really* short blocks of height P(string)

Arithmetic Coding

Both string tree and binary codewords index intervals $\in [0, 1]$



⇒ Encode bacabab$\boxtimes$ with 0111. . . 01

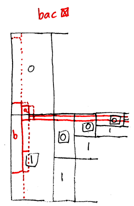
Navigate string tree to find interval on real line.

Use 'binary symbol code budget' diagram[1] to find binary codeword that lies entirely within this interval.

[1] week 3 notes, or MacKay Fig 5.1 p96

Arithmetic Coding

Overlay string tree on binary symbol code tree



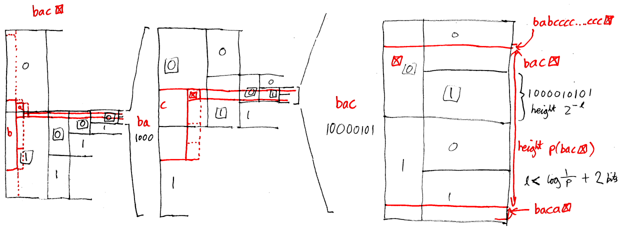
From $P(x_1)$ distribution can't begin to encode 'b' yet

Look at $P(x_2 \,|\, x_1 = \mathtt{b})$ can't start encoding 'ba' either

Look at $P(x_3 \,|\, \mathtt{ba})$. Message for 'bac' begins 1000

Arithmetic Coding

Diagram: zoom in. **Code:** rescale to avoid underflow



From $P(x_4 \mid \mathrm{bac})$. Message 'bac⊠' encoded by 1000010101

Encoding lies only within message: uniquely decodable

1000010110 would also work: slight inefficiency

Tutorial homework: prove encoding length $< \log \frac{1}{P(\mathbf{x})} + 2$ bits
An excess of 2 bits *on the whole file* (millions or more bits?)
Arithmetic coding compresses very close to the information content given by the probabilistic model used by both the sender and receiver.

The conditional probabilities $P(x_i \mid \mathbf{x}_{j<i})$ can change for each symbol. Arbitrary adaptive models can be used (if you have one).

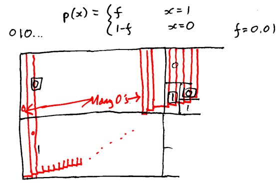
Large blocks of symbols are compressed together: possibly your whole file. The inefficiencies of symbol codes have been removed.

Huffman coding blocks of symbols requires an exponential number of codewords. In arithmetic coding, each character is predicted one at a time, as in a guessing game. The model and arithmetic coder just consider those $|\mathcal{A}_X|$ options at a time. None of the code needs to enumerate huge numbers of potential strings. (De)coding costs should be linear in the message length.

Model probabilities $P(\mathbf{x})$ might need to be rounded to values $Q(\mathbf{x})$ that can be represented consistently by the encoder and decoder. This approximation introduces the usual average overhead: $D_{\mathrm{KL}}(P \,||\, Q)$.

AC and sparse files

Finally we have a practical coding algorithm for sparse files



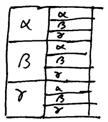
(You could make a better picture!)

The initial code-bit 0, encodes many initial message 0's.

Notice how the first binary code bits will locate the first 1. Something like run-length encoding has dropped out.

Non-binary encoding

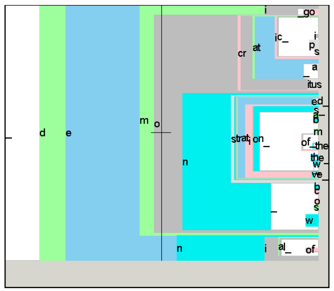
Can overlay string on any other indexing of [0,1] line



Now know how to compress into α, β and γ

Dasher

Dasher is an information-efficient text-entry interface.
Use the same string tree. Gestures specify which one we want.



http://www.inference.phy.cam.ac.uk/dasher/

Information Theory

http://www.inf.ed.ac.uk/teaching/courses/it/

Week 5
Models for stream codes

Iain Murray, 2013

School of Informatics, University of Edinburgh

Card prediction

3 cards with coloured faces:

1. one white and one black face
2. two black faces
3. two white faces

I shuffle cards and turn them over randomly. I select a card and way-up uniformly at random and place it on a table.

Question: You see a black face. What is the probability that the other side of the same card is white?

$$P(x_2 = \mathtt{W} \mid x_1 = \mathtt{B}) = \quad ^1\!/_3, \quad ^1\!/_2, \quad ^2\!/_3, \quad \text{other?}$$

Notes on the card prediction problem:

This card problem is Ex. 8.10a), MacKay, p142.

It is *not* the same as the famous 'Monty Hall' puzzle: Ex. 3.8–9 and http://en.wikipedia.org/wiki/Monty_Hall_problem

The Monty Hall problem is also worth understanding. Although the card problem is (hopefully) less controversial and more straightforward. The process by which a card is selected should be clear: $P(c) = 1/3$ for $c = 1, 2, 3$, and the face you see first is chosen at random: e.g., $P(x_1 = \mathtt{B} | c = 1) = 0.5$.

Many people get this puzzle wrong on first viewing (it's easy to mess up). We'll check understanding again with another prediction problem in a tutorial exercise. If you do get the answer right immediately (are you sure?), this is will be a simple example on which to demonstrate some formalism.

How do we solve it formally?

Use Bayes rule?

$$P(x_2 = \mathtt{W} \mid x_1 = \mathtt{B}) = \frac{\boxed{P(x_1 = \mathtt{B} \mid x_2 = \mathtt{W})}\, P(x_2 = \mathtt{W})}{P(x_1 = \mathtt{B})}$$

The $\boxed{\text{boxed}}$ term is no more obvious than the answer!

Bayes rule is used to 'invert' forward generative processes that we understand.

The first step to solve inference problems is to write down a model of your data.

The card game model

Cards: 1) B|W, 2) B|B, 3) W|W

$$P(c) = \begin{cases} 1/3 & c = 1,2,3 \\ 0 & \text{otherwise.} \end{cases}$$

$$P(x_1 = \mathtt{B} \,|\, c) = \begin{cases} 1/2 & c = 1 \\ 1 & c = 2 \\ 0 & c = 3 \end{cases}$$

Bayes rule can 'invert' this to tell us $P(c \,|\, x_1 = \mathtt{B})$;
infer the generative process for the data we have.

Inferring the card

Cards: 1) B|W, 2) B|B, 3) W|W

$$P(c \,|\, x_1 = \mathtt{B}) = \frac{P(x_1 = \mathtt{B} \,|\, c)\, P(c)}{P(x_1 = \mathtt{B})}$$

$$\propto \begin{cases} 1/2 \cdot 1/3 = 1/6 & c = 1 \\ 1 \cdot 1/3 = 1/3 & c = 2 \\ 0 & c = 3 \end{cases}$$

$$= \begin{cases} 1/3 & c = 1 \\ 2/3 & c = 2 \end{cases}$$

Q *"But aren't there two options given a black face, so it's 50–50?"*
A There are two options, but the likelihood for one of them is $2\times$ bigger

Predicting the next outcome

For this problem we can spot the answer, for more complex problems we want a formal means to proceed.

$P(x_2 \,|\, x_1 = \mathtt{B})$?

Need to introduce c to use expressions we know:

$$P(x_2 \,|\, x_1 = \mathtt{B}) = \sum_{c \in 1,2,3} P(x_2, c \,|\, x_1 = \mathtt{B})$$
$$= \sum_{c \in 1,2,3} P(x_2 \,|\, x_1 = \mathtt{B}, c)\, P(c \,|\, x_1 = \mathtt{B})$$

Predictions we would make if we knew the card, weighted by the posterior probability of that card. $P(x_2 = \mathtt{W} \,|\, x_1 = \mathtt{B}) = 1/3$

Strategy for solving inference and prediction problems:

When interested in something y, we often find we can't immediately write down mathematical expressions for $P(y \,|\, \text{data})$.

So we introduce stuff, z, that helps us define the problem:

$P(y \,|\, \text{data}) = \sum_z P(y, z \,|\, \text{data})$

by using the sum rule. And then split it up:

$P(y \,|\, \text{data}) = \sum_z P(y \,|\, z, \text{data})\, P(z \,|\, \text{data})$

using the product rule. If knowing extra stuff z we can predict y, we are set: weight all such predictions by the posterior probability of the stuff ($P(z \,|\, \text{data})$, found with Bayes rule).

Sometimes the extra stuff summarizes everything we need to know to make a prediction:

$P(y \,|\, z, \text{data}) = P(y \,|\, z)$

although not in the card game above.

Not convinced?

Not everyone believes the answer to the card game question.

Sometimes probabilities are counter-intuitive. I'd encourage you to write simulations of these games if you are at all uncertain. Here is an Octave/Matlab simulator I wrote for the card game question:

```
cards = [1 1;
         0 0;
         1 0];
num_cards = size(cards, 1);

N = 0; % Number of times first face is black
kk = 0; % Out of those, how many times the other side is white

for trial = 1:1e6
    card = ceil(num_cards * rand());
    face = 1 + (rand < 0.5);
    other_face = (face==1) + 1;
    x1 = cards(card, face);
    x2 = cards(card, other_face);

    if x1 == 0
        N = N + 1;
        kk = kk + (x2 == 1);
    end
end

approx_probability = kk / N
```

Sparse files

$\mathbf{x} = 000010000100000001000\ldots000$

We are interested in predicting the $(N+1)$th bit.

Generative model:

$$P(\mathbf{x} \,|\, f) = \prod_i P(x_i \,|\, f) = \prod_i f^{x_i}(1-f)^{1-x_i}$$
$$= f^k(1-f)^{N-k}, \qquad k = \sum_i x_i = \text{“\# 1s”}$$

Can 'invert', find $p(f \,|\, \mathbf{x})$ with Bayes rule

Inferring $f = P(x_i = 1)$

Cannot do inference without using beliefs

A possible expression of uncertainty: $p(f) = 1, \quad f \in [0,1]$

Bayes rule:

$$\begin{aligned} p(f \,|\, \mathbf{x}) &\propto P(\mathbf{x} \,|\, f)\, p(f) \\ &\propto f^k(1-f)^{N-k} \\ &= \text{Beta}(f;\ k+1,\ N-k+1) \end{aligned}$$

Beta distribution:

$\text{Beta}(f;\alpha,\beta) = \frac{1}{B(\alpha,\beta)} f^{\alpha-1}(1-f)^{\beta-1} = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} f^{\alpha-1}(1-f)^{\beta-1}$

Mean: $\alpha/(\alpha+\beta)$

The $\text{Beta}(\alpha,\beta)$ **distribution** is a standard probability distribution over a variable $f \in [0,1]$ with parameters α and β.

The dependence of the probability density on f is summarized by: $\text{Beta}(f;\alpha,\beta) \propto f^{(\alpha-1)}(1-f)^{(\beta-1)}$.
The $1/B(\alpha,\beta) = \Gamma(\alpha+\beta)/(\Gamma(\alpha)\Gamma(\beta))$ term, which is $(\alpha+\beta-1)!/((\alpha-1)!(\beta-1)!)$ for integer α and β, makes the distribution normalized (integrate to one). Here, it's just some constant with respect to the parameter f.

For comparison, perhaps you are more familiar with a Gaussian (or Normal), $\mathcal{N}(\mu,\sigma^2)$, a probability distribution over a variable $x \in [-\infty,\infty]$, with parameters μ and σ^2.

The dependence of the probability density on x is summarized by $\mathcal{N}(x;\mu,\sigma^2) \propto \exp(-0.5(x-\mu)^2/\sigma^2)$.
We divide this expression by $\int \exp(-0.5(x-\mu)^2/\sigma^2)\,\mathrm{d}x = \sqrt{2\pi\sigma^2}$ to make the distribution normalized.

We found that our posterior beliefs, given observations, are proportional to $f^k(1-f)^{N-k}$ and we know $f \in [0,1]$. Given the form of the f dependence, the posterior distribution must be a Beta distribution. We obtain the parameters α and β by comparing the powers of f and $(1-f)$ in the posterior and in the definition of the Beta distribution. Comparing terms and reading off the answer is easier than doing integration to normalize the distribution from scratch, as in equations 3.11 and 3.12 of MacKay, p52.

Again for comparison: if you were trying to infer a real-valued parameter $y \in [-\infty,\infty]$, and wrote down a posterior: $p(y \,|\, D) \propto p(D \,|\, y)\, p(y)$ and found $p(y \,|\, D) \propto \exp(-ay^2+by)$ for some constants a and b, you could state that $p(y \,|\, D) = \exp(-ay^2+by)/Z$, and derive that the constant must be $Z = \int \exp(-ay^2+by)\mathrm{d}y = \ldots$. Alternatively, you could realize that a quadratic form inside an exp is a Gaussian distribution. Now you just have identify its parameters. As $\mathcal{N}(y;\mu,\sigma^2) \propto \exp(-0.5(y-\mu)^2/\sigma^2) \propto \exp(-0.5y^2/\sigma^2 + (\mu/\sigma^2)y)$, we can identify the parameters of the posterior: $\sigma^2 = 1/(2a)$ and $\mu = b\sigma^2 = b/(2a)$.

References on inferring a probability

The 'bent coin' is discussed in MacKay §3.2, p51

See also Ex. 3.15, p59, which has an extensive worked solution.

The MacKay section mentions that this problem is the one studied by Thomas Bayes, published in 1763. This is true, although the problem was described in terms of a game played on a Billiard table.

The Bayes paper has historical interest, but without modern mathematical notation takes some time to read. Several versions can be found around the web. The original version has old-style typesetting. The paper was retypeset, but with the original long arguments, for Biometrica in 1958:
`http://dx.doi.org/10.1093/biomet/45.3-4.296`

Prediction

Prediction rule from marginalization and product rules:

$$P(x_{N+1} \,|\, \mathbf{x}) = \int P(x_{N+1} \,|\, f, \boxed{\mathbf{x}}) \cdot p(f \,|\, \mathbf{x})\, \mathrm{d}f$$

The boxed dependence can be omitted here.

$$P(x_{N+1}\!=\!1 \,|\, \mathbf{x}) = \int f \cdot p(f \,|\, \mathbf{x})\, \mathrm{d}f = \mathbb{E}_{p(f \,|\, \mathbf{x})}[f] = \frac{k+1}{N+2}.$$

Laplace's law of succession

$$P(x_{N+1}\!=\!1 \,|\, \mathbf{x}) = \frac{k+1}{N+2}$$

Maximum Likelihood (ML): $\hat{f} = \mathrm{argmax}_f\, P(\mathbf{x} \,|\, f) = \frac{k}{N}$.
ML estimate is *unbiased*: $\mathbb{E}[\hat{f}] = f$.

Laplace's rule is like using the ML estimate, but imagining we saw a 0 and a 1 before starting to read in $\mathbf{x}$.

Laplace's rule biases probabilities towards $1/2$.

ML estimate assigns zero probability to unseen symbols.
Encoding zero-probability symbols needs ∞ bits.

New prior / prediction rule

Could use a Beta prior distribution:

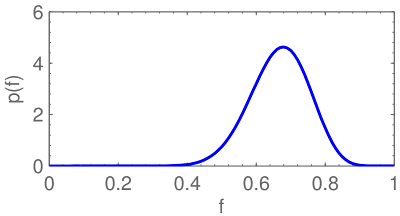
$$\begin{aligned} p(f) &= \mathrm{Beta}(f;\ n_1,\ n_0) \\ p(f \,|\, \mathbf{x}) &\propto f^{k+n_1-1}(1-f)^{N-k+n_0-1} \\ &= \mathrm{Beta}(f;\ k\!+\!n_1,\ N\!-\!k\!+\!n_0) \\ P(x_{N+1}\!=\!1 \,|\, \mathbf{x}) &= \mathbb{E}_{p(f \,|\, \mathbf{x})}[f] = \frac{k+n_1}{N+n_0+n_1} \end{aligned}$$

Think of n_1 and n_0 as previously observed counts

($n_1\!=\!n_0\!=\!1$ gives uniform prior and Laplace's rule)

Large pseudo-counts

Beta(20,10) distribution:



Mean: $2/3$

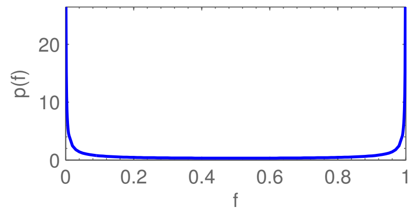
This prior says f close to 0 and 1 are very improbable

We'd need $\gg 30$ observations to change our mind

(to over-rule the prior, or pseudo-observations)

Fractional pseudo-counts

Beta(0.2,0.2) distribution:



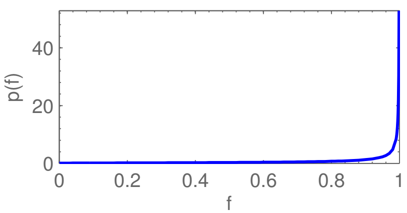
Mean: $1/2$ — notice prior says more than a guess of $f\!=\!1/2$

f is probably close to 0 or 1 but we don't know which yet

One observation will rapidly change the posterior

Fractional pseudo-counts

Beta(1.2,0.2) distribution:



Posterior from previous prior and observing a single 1

Larger alphabets

i.i.d. symbol model:

$$P(\mathbf{x} \,|\, \mathbf{p}) = \prod_i p_i^{k_i}, \qquad \text{where } k_i = \sum_n \mathbb{I}(x_n = a_i)$$

The k_i are counts for each symbol.

Dirichlet prior, generalization of Beta:

$$p(\mathbf{p} \,|\, \boldsymbol{\alpha}) = \mathrm{Dirichlet}(\mathbf{p};\ \boldsymbol{\alpha}) = \frac{\delta(1-\sum_i p_i)}{B(\boldsymbol{\alpha})} \prod_i p_i^{\alpha_i - 1}$$

Dirichlet predictions (Lidstone's law):

$$P(x_{N+1}\!=\!a_i \,|\, \mathbf{x}) = \frac{k_i + \alpha_i}{N + \sum_j \alpha_j}$$

Counts k_i are added to pseudo-counts α_i. All $\alpha_i\!=\!1$ gives Laplace's rule.

More notes on the Dirichlet distribution

The thing to remember is that a Dirichlet is proportional to $\prod_i p_i^{\alpha_i - 1}$

The posterior $p(\mathbf{p} \,|\, \mathbf{x}, \boldsymbol{\alpha}) \propto P(\mathbf{x} \,|\, \mathbf{p})\, p(\mathbf{p} \,|\, \boldsymbol{\alpha})$ will then be Dirichlet with the α_i's increased by the observed counts.

Details (for completeness): $B(\boldsymbol{\alpha})$ is the Beta function $\frac{\prod_i \Gamma(\alpha_i)}{\Gamma(\sum_i \alpha_i)}$.

I left the $0 \le p_i \le 1$ constraints implicit. The $\delta(1 - \sum_i p_i)$ term constrains the distribution to the 'simplex', the region of a hyper-plane where $\sum_i p_i = 1$. But I can't omit this Dirac-delta, because it is infinite when evaluated at a valid probability vector(!).

The density over just the first $(I\!-\!1)$ parameters is finite, obtained by integrating out the last parameter:

$$p(\mathbf{p}_{j<I-1}) = \int p(\mathbf{p} \,|\, \boldsymbol{\alpha})\, \mathrm{d}p_I = \frac{1}{B(\boldsymbol{\alpha})} \left(1 - \textstyle\sum_{i=1}^{I-1} p_i\right)^{\alpha_I - 1} \prod_{i=1}^{I-1} p_i^{\alpha_i - 1}$$

There are no infinities, and the relation to the Beta distribution is now clearer, but the expression isn't as symmetric.

Reflection on Compression

Take any complete compressor.

If "incomplete" imagine an improved "complete" version.

Complete codes: $\sum_{\mathbf{x}} 2^{-\ell(\mathbf{x})} = 1$, $\mathbf{x}$ is whole input file

Interpretation: implicit $Q(\mathbf{x}) = 2^{-\ell(bx)}$

If we believed files were drawn from $P(\mathbf{x}) \neq Q(\mathbf{x})$ we would expect to do $D(P||Q) > 0$ bits better by using $P(\mathbf{x})$.

Compression is the modelling of probabilities of files.

If we think our compressor should 'adapt', we are making a statement about the structure of our beliefs, $P(\mathbf{x})$.

Structure

For any distribution:

$$P(\mathbf{x}) = P(x_1)\prod_{n=2}^{N} P(x_n \,|\, \mathbf{x}_{<n})$$

For i.i.d. symbols: $P(x_n{=}a_i \,|\, \mathbf{p}) = p_i$

$$P(x_n \,|\, \mathbf{x}_{<n}) = \int P(\mathbf{x}_n \,|\, \mathbf{p})\, p(\mathbf{p} \,|\, \mathbf{x}_{<n})\, \mathrm{d}\mathbf{p}$$

$$P(x_n{=}a_i \,|\, \mathbf{x}_{<n}) = \mathbb{E}_{p(\mathbf{p} \,|\, \mathbf{x}_{<n})}[p_i]$$

we saw: easy-to-compute from counts with a Dirichlet prior.

i.i.d. assumption is often terrible: want different structure.

Even then, do we need to specify priors (like the Dirichlet)?

Why not just fit $\mathbf{p}$?

Run over file $\rightarrow$ counts $\mathbf{k}$

Set $p_i = \frac{k_i}{N}$, (Maximum Likelihood, and obvious, estimator)

Save $(\mathbf{p}, \mathbf{x})$, $\mathbf{p}$ in a header, $\mathbf{x}$ encoded using $\mathbf{p}$

Simple? Prior-assumption-free?

Fitting cannot be optimal

When fitting, we never save a file $(\mathbf{p}, \mathbf{x})$ where

$$p_i \neq \frac{k_i(\mathbf{x})}{N}$$

Informally: we are encoding $\mathbf{p}$ twice

More formally: the code is incomplete

However, `gzip` and arithmetic coders are incomplete too, but they are still useful!

In some situations the fitting approach is very close to optimal

Fitting isn't that easy!

Setting $p_i = \frac{k_i}{N}$ is easy. **How do we encode the header?**

Optimal scheme depends on $p(\mathbf{p})$; **need a prior!**

What precision to send parameters?
Trade-off between header and message size.

Interesting models will have many parameters.
Putting them in a header could dominate the message.

Having both ends learn the parameters while {en,de}coding the file avoids needing a header.

For more (non-examinable) detail on these issues see MacKay p352–353

Richer models

Images are not bags of i.i.d. pixels
Text is not a bag of i.i.d. characters/words

(although many "Topic Models" get away with it!)

Less restrictive assumption than:

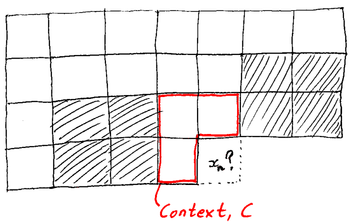
$$P(x_n \,|\, \mathbf{x}_{<n}) = \int P(\mathbf{x}_n \,|\, \mathbf{p})\, p(\mathbf{p} \,|\, \mathbf{x}_{<n})\, \mathrm{d}\mathbf{p}$$

is

$$P(x_n \,|\, \mathbf{x}_{<n}) = \int P(\mathbf{x}_n \,|\, \mathbf{p}_{C(\mathbf{x}_{<n})})\, p(\mathbf{p}_{C(\mathbf{x}_{<n})} \,|\, \mathbf{x}_{<n})\, \mathrm{d}\mathbf{p}_{C(\mathbf{x}_{<n})}$$

Probabilities depend on the local context, C:
— Surrounding pixels, already {en,de}coded
— Past few characters of text

Image contexts

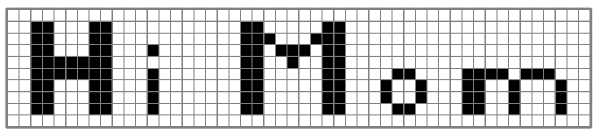


$$P(x_i{=}\texttt{Black} \,|\, C) = \frac{k_{\mathrm{B}|C} + \alpha}{N_C + \alpha|\mathcal{A}|} = \frac{2+\alpha}{7+2\alpha}$$

There are 2^p contexts of size p binary pixels
Many more counts/parameters than i.i.d. model

A good image model?

The context model isn't far off what several real image compression systems do for binary images.



With arithmetic coding we go from 500 to 220 bits

A better image model might do better

If we knew it was text and the font we'd need fewer bits!

Context size

How big to make the context?

`kids_make_nutr` $\boxed{?}$

Context length:

- **0:** i.i.d. bag of characters
- **1:** bigrams, give vowels higher probability
- **>1:** predict using possible words
- **$\gg$1:** use understanding of sentences?

Ideally we'd use really long contexts, as humans do.

Problem with large contexts

For simple counting methods, statistics are poor:

$$p(x_n = a_i \,|\, \mathbf{x}_{<n}) = \frac{k_{i|C} + \alpha}{N_C + \alpha|\mathcal{A}|}$$

$k_{\cdot|C}$ will be zero for most symbols in long contexts
Predictions become uniform $\Rightarrow$ no compression.

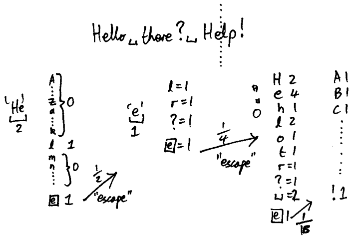
What broke? We believe some contexts are related:

`kids_make_nutr` $\boxed{?}$
`kids_like_nutr` $\boxed{?}$

while the Dirichlet prior says they're unrelated

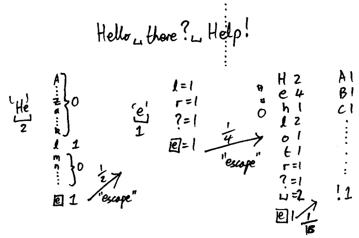
Prediction by Partial Match (PPM)

One way of smoothing predictions from several contexts:



Model: draw using fractions observed at context
Escape to shorter context with some probability (variant-dependent)

Prediction by Partial Match (PPM)



$$P(\texttt{l}\,|\,\texttt{Hello there? He}) = \tfrac{1}{2} + \tfrac{1}{2}\big(\tfrac{1}{4} + \tfrac{1}{4}\big(\tfrac{2}{16} + \tfrac{1}{16}\tfrac{1}{|\mathcal{A}|}\big)\big)$$

$$P(\texttt{!}\,|\,\texttt{Hello there? He}) = \tfrac{1}{2}\tfrac{1}{4}\tfrac{1}{16}\tfrac{1}{|\mathcal{A}|}$$

$$P(\texttt{␣}\,|\,\texttt{Hello there? He}) = \tfrac{1}{2}\big(\tfrac{1}{4}\big(\tfrac{2}{16} + \tfrac{1}{16}\tfrac{1}{|\mathcal{A}|}\big)\big)$$

Prediction by Partial Match comments

First PPM paper: Clearly and Witten (1984). Many variants since. The best PPM variant's text compression is now highly competitive. Although it is clearly possible to come up with better models of text.

The ideas are common to methods with several other names. PPM is a name used a lot in text compression for the combination of this type of model with arithmetic coding.

Other prediction methods and more advanced models

Better methods that smooth counts from different contexts:
`http://research.microsoft.com/pubs/64808/tr-10-98.pdf`

I covered Beta and Dirichlet priors to demonstrate that prediction rules can be derived from models. There isn't time in this course to take this idea further, but state-of-the-art predictions can result from Bayesian inference in more complicated *hierarchical* models:
`http://homepages.inf.ed.ac.uk/sgwater/papers/nips05.pdf`
`http://www.aclweb.org/anthology/P/P06/P06-1124.pdf`

Information Theory

`http://www.inf.ed.ac.uk/teaching/courses/it/`

Week 6
Communication channels and Information

Iain Murray, 2012
School of Informatics, University of Edinburgh

Noisy channel communication

Input File
↓ compress
$\mathbf{x}$ = 00110001
↓ add redundancy

00110001 00110001 00110001	→	Noisy Channel	→	00011001 01110011 10110100

transmitted message
often called $\mathbf{t}$ or $\mathbf{x}$

decode ↓
$\hat{\mathbf{x}}$ = 00110001
decompress ↓

If all errors corrected: **Original file**

Some notes on the noisy channel setup:

Noisy communication was outlined in lecture 1, then abandoned to cover compression, representing messages for a noiseless channel.

Why compress, remove all redundancy, just to add it again?

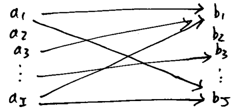
Firstly remember that repetition codes require a *lot* of repetitions to get a negligible probability of error. We are going to have to add better forms of redundancy to get reliable communication at good rates. Our files won't necessarily have the right sort of redundancy.

It is often useful to have modular designs. We can design an encoding/decoding scheme for a noisy channel separately from modelling data. Then use a compression system to get our file appropriately distributed over the required alphabet.

It is possible to design a combined system that takes redundant files and encodes them for a noisy channel. MN codes do this:
`http://www.inference.phy.cam.ac.uk/mackay/mncN.pdf`
These lectures won't discuss this option.

Discrete Memoryless Channel, Q

Discrete: Inputs x and Outputs y have discrete (sometimes binary) alphabets:

$x \in \mathcal{A}_\mathcal{X}$  $y \in \mathcal{A}_\mathcal{Y}$

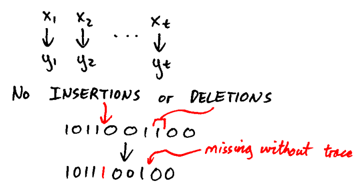
$$Q_{j|i} = P(y\!=\!b_j \,|\, x\!=\!a_i)$$

Memoryless: outputs always drawn using fixed Q matrix

We also assume channel is **synchronized**

Synchronized channels

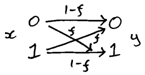
We know that a sequence of inputs was sent and which outputs go with them.



Dealing with insertions and deletions is a tricky topic, an active area of research that we will avoid

Binary Symmetric Channel (BSC)

A natural model channel for binary data:



$$Q = \begin{array}{c} \\ \end{array}\begin{array}{c} \begin{array}{cc} 0 & 1 \end{array} \ x \\ \begin{bmatrix} 1-f & f \\ f & 1-f \end{bmatrix} \begin{array}{l} 0 \\ 1 \end{array} \ y \end{array}$$

Alternative view:

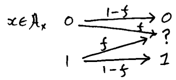
noise drawn from $p(n) = \begin{cases} 1-f & n=0 \\ f & n=1 \end{cases}$

$y \;=\; (x+n) \bmod 2 \;=\; x \;\texttt{XOR}\; n$

```
% Matlab/Octave              /* C (or Python) */
y = mod(x+n, 2);             y = (x+n) % 2;
y = bitxor(x, n);            y = x ^ n;
```

Binary Erasure Channel (BEC)

An example of a non-binary alphabet:



$y \in \mathcal{A}_Y = \{0, 1, ?\}$

$$Q = \begin{array}{c} \begin{array}{cc} 0 & 1 \end{array} \ x \\ \begin{bmatrix} 1-f & 0 \\ f & f \\ 0 & 1-f \end{bmatrix} \begin{array}{l} 0 \\ ? \\ 1 \end{array} \ y \end{array}$$

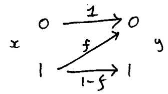
With this channel corruptions are obvious

Feedback: could ask for retransmission
Care required: negotiation could be corrupted too
Feedback sometimes not an option: hard disk storage

The BEC is not the *deletion channel*. Here symbols are replaced with a placeholder, in the deletion channel they are removed entirely and it is no longer clear at what time symbols were transmitted.

Z channel

Cannot always treat symbols symmetrically



$$Q = \begin{array}{c} x \\ \begin{bmatrix} 1 & f \\ 0 & 1-f \end{bmatrix} \begin{array}{l} 0 \\ 1 \end{array} \ y \end{array}$$

"Ink gets rubbed off, but never added"

Channel Probabilities

Channel definition:

$$Q_{j|i} = P(y{=}b_j \,|\, x{=}a_i)$$

Assume there's nothing we can do about Q.
We can choose what to throw at the channel.

Input distribution: $\mathbf{p}_X = \begin{pmatrix} p(x{=}a_1) \\ \vdots \\ p(x{=}a_I) \end{pmatrix}$

Joint distribution: $P(x,y) = P(x)\,P(y\,|\,x)$

Output distribution: $P(y) = \sum_x P(x,y)$

vector notation: $\mathbf{p}_Y = Q\,\mathbf{p}_X$

(the usual relationships for any two variables x and y)

A little more detail on channel probabilities:

More detail on why the output distribution can be found by a matrix multiplication:

$$\begin{aligned} p_{Y,j} = P(y{=}b_j) &= \sum_i P(y{=}b_j, x{=}a_i) \\ &= \sum_i P(y{=}b_j \,|\, x{=}a_i)\, P(x{=}a_i) \\ &= \sum_i Q_{j|i}\, p_{X,i} \\ \mathbf{p}_Y &= Q\,\mathbf{p}_X \end{aligned}$$

Care: some texts (but not MacKay) use the transpose of our Q as the transition matrix, and so use left-multiplication instead.

Channels and Information

Three distributions: $P(x)$, $P(y)$, $P(x,y)$
Three observers: sender, receiver, omniscient outsider

Average surprise of receiver: $H(Y) = \sum_y P(y) \log 1/P(y)$
Partial information about sent file and added noise

Average information of file: $H(X) = \sum_x P(x) \log 1/P(x)$
Sender observes all of this, but no information about noise

Omniscient outsider experiences total joint entropy of file and noise: $H(X,Y) = \sum_{x,y} P(x,y) \log 1/P(x,y)$

Joint Entropy

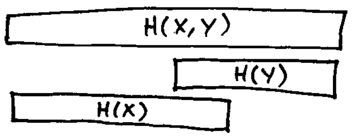
Omniscient outsider gets more information on average than an observer at one end of the channel: $H(X,Y) \geq H(X)$

Outsider can't have more information than both ends combined:

$$H(X,Y) \leq H(X) + H(Y)$$

with equality only if X and Y are independent

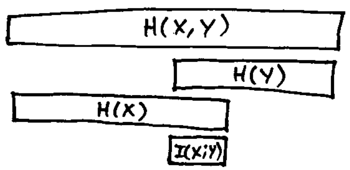
(independence useless for communication!)



Mutual Information (1)

How much too big is $H(X) + H(Y) \neq H(X,Y)$?

Overlap: $I(X;Y) = H(X) + H(Y) - H(X,Y)$
is called the **mutual information**



It's the average information content "shared" by the dependent X and Y ensembles. (more insight to come)

Inference in the channel

The receiver doesn't know x, but on receiving y can update the prior $P(x)$ to a posterior:

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

e.g. for BSC with $P(x{=}1) = 0.5$, $P(x\,|\,y) = \begin{cases} 1-f & x=0 \\ f & x=1 \end{cases}$
other channels may have less obvious posteriors

Another distribution we can compute the entropy of!

Conditional Entropy (1)

We can condition every part of an expression on the setting of an arbitrary variable:

$$H(X\,|\,y) = \sum_x P(x\,|\,y) \log 1/P(x\,|\,y)$$

Average information available from seeing x, given that we already know y.

On average this is written:

$$H(X\,|\,Y) = \sum_y P(y)\, H(X\,|\,y) = \sum_{x,y} P(x,y) \log 1/P(x\,|\,y)$$

Conditional Entropy (2)

Similarly

$$H(Y\,|\,X) = \sum_{x,y} P(x,y) \log 1/P(y\,|\,x)$$

is the average uncertainty about the output that the sender has, given that she knows what she sent over the channel.

Intuitively this should be less than the average surprise that the receiver will experience, $H(Y)$.

Conditional Entropy (3)

The *chain rule* for entropy:

$$H(X,Y) = H(X) + H(Y\,|\,X) = H(Y) + H(X\,|\,Y)$$

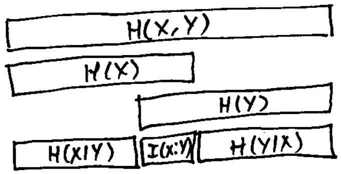
"The average coding cost of a pair is the same regardless of whether you treat them as a joint event, or code one and then the other."

Proof:

$$H(X,Y) = \sum_x \sum_y p(x)\, p(y\,|\,x) \left[\log \frac{1}{p(x)} + \log \frac{1}{p(y\,|\,x)}\right]$$

$$= \sum_x p(x) \log \frac{1}{p(x)} \cancel{\sum_y p(y\,|\,x)}^{\,1} + \sum_x \sum_y p(x,y) \log \frac{1}{p(y\,|\,x)}$$

Mutual Information (2)



The receiver thinks: $I(X;Y) = H(X) - H(X\,|\,Y)$

The mutual information is, on average, the information content of the input minus the part that is still uncertain after seeing the output. That is, the average information that we can get about the input over the channel.

$I(X;Y) = H(Y) - H(Y\,|\,X)$ is often easier to calculate

The Capacity

Where are we going?

$I(X;Y)$ depends on the channel and input distribution $\mathbf{p}_X$

The Capacity: $C(Q) = \max_{\mathbf{p}_X} I(X;Y)$

C gives the maximum average amount of information we can get in one use of the channel.

We will see that reliable communication is possible at C bits per channel use.

Lots of new definitions

When dealing with extended ensembles, independent identical copies of an ensemble, entropies were easy: $H(X^K) = K\,H(X)$.

Dealing with channels forces us to extend our notions of information to collections of dependent variables. For every joint, conditional and marginal probability we have a different entropy and we'll want to understand their relationships.

Unfortunately this meant seeing a lot of definitions at once.
They are summarized on pp138–139 of MacKay. And also in the following tables.

The probabilities associated with a channel

Very little of this is special to channels, it's mostly results for any pair of dependent random variables.

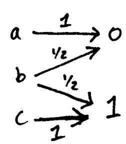
Distribution	Where from?	Interpretation / Name
$P(x)$	We choose	Input distribution
$P(y\,|\,x)$	Q, channel definition	Channel noise model Sender's beliefs about output
$P(x,y)$	$p(y\,|\,x)\,p(x)$	Omniscient outside observer's joint distribution
$P(y)$	$\sum_x p(x,y) = Q\,\mathbf{p}_X$	(Marginal) output distribution
$P(x\,|\,y)$	$p(y\,|\,x)\,p(x)/p(y)$	Receiver's beliefs about input. "Inference"

Corresponding information measures

$H(X)$	$\sum_x p(x) \log 1/p(x)$	Ave. info. content of source Sender's ave. surprise on seeing x
$H(Y)$	$\sum_y p(y) \log 1/p(y)$	Ave. info. content of output Partial info. about x *and* noise Ave. surprise of receiver
$H(X,Y)$	$\sum_{x,y} p(x,y) \log 1/p(x,y)$	Ave. info. content of (x,y) or "source and noise". Ave. surprise of outsider
$H(X\,|\,y)$	$\sum_x p(x\,|\,y) \log 1/p(x\,|\,y)$	Uncertainty after seeing output
$H(X\,|\,Y)$	$\sum_{x,y} p(x,y) \log 1/p(x\,|\,y)$	Average, $\mathbb{E}_{p(y)}[H(X\,|\,y)]$
$H(Y\,|\,X)$	$\sum_{x,y} p(x,y) \log 1/p(y\,|\,x)$	Sender's ave. uncertainty about y
$I(X;Y)$	$H(X){+}H(Y){-}H(X,Y)$	'Overlap' in ave. info. contents
	$H(X) - H(X\,|\,Y)$	Ave. uncertainty reduction by y Ave info. about x over channel.
	$H(Y) - H(Y\,|\,X)$	Often easier to calculate

And review the diagram relating all these quantities!

Ternary confusion channel



$$\begin{array}{cccc} & a & x\;b & c & \\ Q = & \begin{bmatrix} 1 & 1/2 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} & & & \begin{matrix} 0 \\ 1 \end{matrix}\; y \end{array}$$

Assume $\mathbf{p}_X = [1/3, 1/3, 1/3]$. What is $I(X;Y)$?

$H(X) - H(X\,|\,Y) = H(Y) - H(Y\,|\,X) = 1 - 1/3 = 2/3$

Optimal input distribution: $\mathbf{p}_X = [1/2, 0, 1/2]$

For which $I(X;Y) = 1$, the *capacity* of the channel.

Information Theory

http://www.inf.ed.ac.uk/teaching/courses/it/

Week 7
Noisy channel coding

Iain Murray, 2012

School of Informatics, University of Edinburgh

Mutual Information revisited

Verify for yourself: $I(X;Y) = D_{\mathrm{KL}}(p(x,y)\,||\,p(x)\,p(y))$

Mutual information is non-negative:

$H(X) - H(X\,|\,Y) = I(X;Y) \geq 0,$ Proof: Gibbs' inequality

Conditioning cannot increase uncertainty on average

Concavity of Entropy

Consider $H(X) \geq H(X\,|\,C)$ for the special case:

$$p(c) = \begin{cases} \lambda & c=1 \\ 1-\lambda & c=2 \end{cases}$$

$$p(x\,|\,c{=}1) = p_1(x), \quad p(x\,|\,c{=}2) = p_2(x)$$

$$p(x) = \lambda p_1(x) + (1-\lambda)p_2(x)$$

which implies the entropy is concave:

$$H(\lambda \mathbf{p}_1 + (1-\lambda)\mathbf{p}_2) \;\geq\; \lambda H(\mathbf{p}_1) \,+\, (1-\lambda)H(\mathbf{p}_2)$$

Concavity of I(X;Y)

$$\begin{aligned} I(X;Y) &= H(Y) - H(Y\,|\,X) \\ &= H(Q\mathbf{p}_X) - \mathbf{p}_X^\top H(Y\,|\,x) \end{aligned}$$

First term concave in $\mathbf{p}_X$ (concave function of linear transform)

Second term linear in $\mathbf{p}_X$

Mutual Information is concave in input distribution

It turns out that $I(X;Y)$ is convex in the channel paramters Q. Reference: Cover and Thomas §2.7.

Noisy typewriter

See the fictitious noisy typewriter model, MacKay p148

For Uniform input distribution: $\mathbf{p}_X = [^1/_{27}, ^1/_{27}, \ldots ^1/_{27}]^\top$

$H(X) = \log(27)$

$$p(x\,|\,y{=}\mathtt{B}) = \begin{cases} ^1/_3 & x = \mathtt{A} \\ ^1/_3 & x = \mathtt{B} \\ ^1/_3 & x = \mathtt{C} \\ 0 & \text{otherwise.} \end{cases} \quad \Rightarrow H(X\,|\,y{=}\mathtt{B}) = \log 3$$

$$H(X\,|\,Y) = \mathbb{E}_{p(y)}[H(X\,|\,y)] = \log 3$$

$$I(X;Y) = H(X) - H(X\,|\,Y) = \log {}^{27}/_3 = \log_2 9 \text{ bits}$$

Noisy Typewriter Capacity:

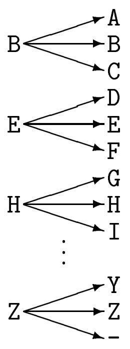
In fact, the capacity: $C = \max_{\mathbf{p}_X} I(X;Y) = \log_2 9$ bits

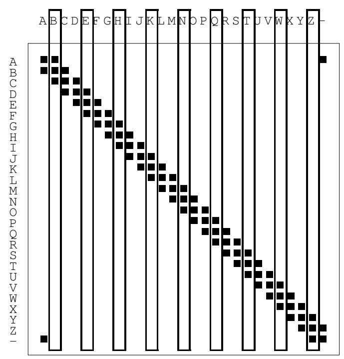
Proof: any asymmetric input distribution can be shifted by any number of characters to get new distributions with the same mutual information (by symmetry). Because $I(X;Y)$ is concave, any convex combination of these distributions will have performance as good or better. The uniform distribution is the average of all the shifted distributions, so can be no worse than any asymmetric distribution.

Under the uniform input distribution, the receiver infers 9 bits of information about the input. Shannon's theory will tell us that this is the fastest rate that we can communicate information without error.

For this channel there is a simple way of achieving error-less communication at this rate: only use 9 of the inputs as on the next slide. Confirm that the mutual information for this input distribution is also $\log_2 9$ bits.

Non-confusable inputs





MacKay, p153

The challenge

Most channels aren't as easy-to-use as the typewriter.

How to communicate without error with messier channels?

Idea: use N^{th} extension of channel:

Treat N uses as one use of channel with

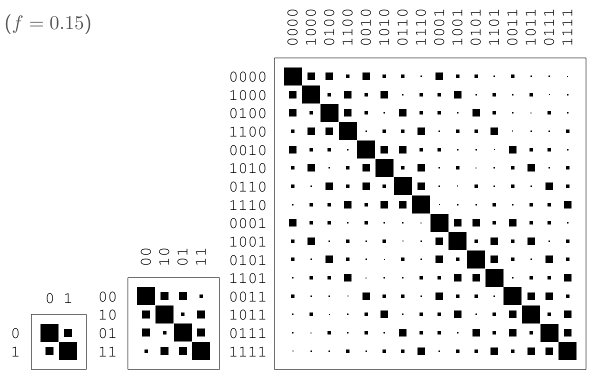
Input $\in \mathcal{A}_X^N$

Output $\in \mathcal{A}_Y^N$

For *large* N a subset of inputs can be non-confusable with high-probability.

Extensions of the BSC

$(f = 0.15)$

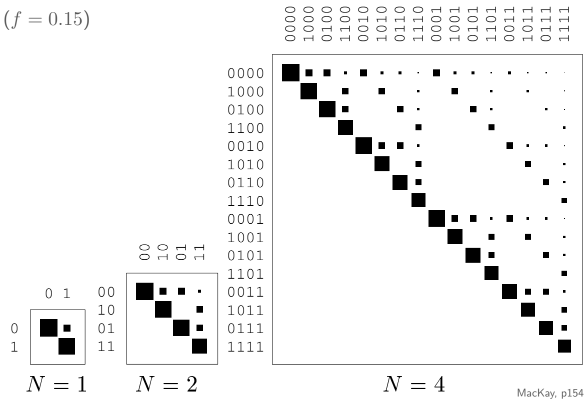


$N = 1$ $N = 2$ $N = 4$

MacKay, p153

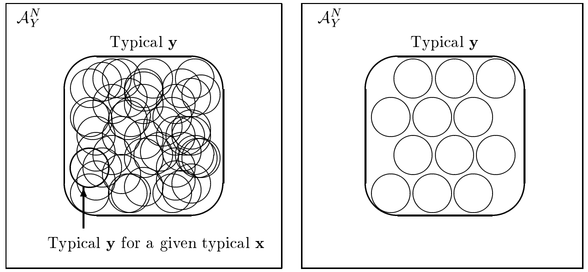
Extensions of the Z channel

($f = 0.15$)



MacKay, p154

Non-confusable typical sets



MacKay, p155

Do the 4th extensions look like the noisy typewriter?

I think they look like a mess! For the BSC the least confusable inputs are `0000` and `1111` – a simple repetition code. For the Z-channel one might use more inputs if one has a moderate tolerance to error. (Might guess this: the Z-channel has higher capacity.)

To get really non-confusable inputs need to extend to larger N. Large blocks are hard to visualize. The cartoon on the previous slide is part of how the noisy channel theorem is proved.

We know from source-coding that only some large blocks under a given distribution are "typical". For a given input, only certain outputs are typical (e.g., all the blocks that are within a few bit-flips from the input). If we select only a tiny subset of inputs, *codewords*, whose typical output sets only weakly overlap. Using these nearly non-confusable inputs will be like using the noisy typewriter.

That will be the idea. But as with compression, dealing with large blocks can be impractical. So first we're going to look at some simple, practical error correcting codes.

ISBNs — checksum example

On the back of Bishop's Pattern Recognition book: (early printings)
`ISBN: 0-387-31073-8`
Group-Publisher-Title-Check

The check digit: $x_{10} = x_1 + 2\,x_2 + 3\,x_3 + \cdots + 9\,x_9 \bmod 11$

Matlab/Octave: `mod((1:9)*[0 3 8 7 3 1 0 7 3]', 11)`

Numpy: `dot([0,3,8,7,3,1,0,7,3], r_[1:10]) % 11`

Questions:
— Why is the check digit there?
— $\sum_{i=1}^{9} x_i \bmod 10$ would detect any single-digit error.
— Why is each digit pre-multiplied by i?
— Why do mod 11, which means we sometimes need `X`?

Some people often type in ISBNs. It's good to tell them of mistakes without needing a database lookup to an archive of all books.

Not only are all single-digit errors detected, but also transposition of two adjacent digits.

The back of the MacKay textbook cannot be checked using the given formula. In recent years books started to get 13-digit ISBN's. These have a different check-sum, performed modulo-10, which doesn't provide the same level of protection.

Check digits are such a good idea, they're found on *many* long numbers that people have to type in, or are unreliable to read:
— Product codes (UPC, EAN, ...)
— Government issued IDs for Tax, Health, etc., the world over.
— Standard magnetic swipe cards.
— Airline tickets.
— Postal barcodes.

[7,4] Hamming Codes

Sends $K = 4$ source bits
With $N = 7$ uses of the channel

Can detect *and correct* any single-bit error in block.

My explanation in the lecture and on the board followed that in the MacKay book, p8, quite closely.

You should understand how this block code works.

To think about: how can we make a code (other than a repetition code) that can correct more than one error?

[N,K] Block codes

[7,4] Hamming code was an example of a block code

We use $S = 2^K$ codewords (hopefully hard-to-confuse)

Rate: # bits sent per channel use:

$$R = \frac{\log_2 S}{N} = \frac{K}{N}$$

Example, repetition code R_3:
$N = 3$, $S = 2$ codewords: 000 and 111. $R = 1/3$.

Example, $[7,4]$ Hamming code: $R = 4/7$.

Some texts (not MacKay) use $(\log_{|\mathcal{A}_X|} S)/N$, the relative rate compared to a uniform distribution on the non-extended channel. I don't use this definition.

Noisy channel coding theorem

Consider a channel with capacity $C = \max_{\mathbf{p}_X} I(X;Y)$

[E.g.'s, Tutorial 5: BSC, $C = 1 - H_2(f)$; BEC $C = 1 - f$]

No feed back channel

For any desired error probability $\epsilon > 0$, e.g. 10^{-15}, 10^{-30}...

For any rate $R < C$

1) There is a block code (N might be big) with error $< \epsilon$ and rate $K/N \geq R$.

2) If we transmit at a rate $R > C$ then there is a non-zero error probability that we cannot go beneath.

The minimum error probability for $R > C$ is found by "rate distortion theory", mentioned in the final lecture, but not part of this course. More detail §10.4, pp167–168, of MacKay. Much more in Cover and Thomas.

Capacity as an upper limit

It is easy to see that errorless transmission above capacity is impossible for the BSC and the BEC. It would imply we can compress any file to less than its information content.

BSC: Take a message with information content $K + NH_2(f)$ bits. Take the first K bits and create a block of length N using an error correction code for the BSC. Encode the remaining bits into N binary symbols with probability of a one being f. Add together the two blocks modulo 2. If the error correcting code can identify the 'message' and 'noise' bits, we have compressed $K + NH_2(f)$ bits into N binary symbols. Therefore, $N > K + NH_2(f) \;\Rightarrow\; K/N < 1 - H_2(f)$. That is, $R < C$ for errorless communication.

BEC: we typically receive $N(1-f)$ bits, the others having been erased. If the block of N bits contained a message of K bits, and is recoverable, then $K < N(1-f)$, or we have compressed the message to less than K bits. Therefore $K/N < (1-f)$, or $R < C$.

Linear [N,K] codes

Hamming code example of linear code: $\mathbf{t} = G^{\top}\mathbf{s} \bmod 2$

Transmitted vector takes on one of 2^K codewords

Codewords satisfy $M = N - K$ constraints: $H\mathbf{t} = 0 \bmod 2$

Dimensions:

$\mathbf{t}$	$N \times 1$
$G^{\top}$	$N \times K$
$\mathbf{s}$	$K \times 1$
H	$M \times N$

For the BEC, choosing constraints H at random makes communication approach capacity for large N!

Required constraints

There are $E \approx Nf$ erasures in a block

Need E *independent* constraints to fill in erasures

H matrix provides $M = N - K$ constraints.
But they won't all be independent.

Example: two Hamming code parity checks are:
$t_1 + t_2 + t_3 + t_5 = 0$ and $t_2 + t_3 + t_4 + t_6 = 0$
We could specify 'another' constraint:
$t_1 + t_4 + t_5 + t_6 = 0$
But this is the sum (mod 2) of the first two, and provides no extra checking.

H constraints

Q. Why would we choose H with redundant rows?

A. We don't know ahead of time which bits will be erased. Only at decoding time do we set up the M equations in the E unknowns.

For H filled with $\{0, 1\}$ uniformly at random, we expect to get E independent constraints with only $M = E+2$ rows.

Recall $E \approx Nf$. For large N, if $f < M/N$ there will be enough constraints with high probability.

Errorless communication possible if
$f < (N-K)/N = 1 - R$ or if $R < 1-f$, i.e., $R < C$.

A large random linear code achieves capacity.

Details on finding independent constraints:

Imagine that while checking parity conditions, a row of H at a time, you have seen n independent constraints so far.

$P(\text{Next row of } H \text{ useful}) = 1 - 2^n/2^E = 1 - 2^{n-E}$

There are 2^E possible equations in the unknowns, but 2^n of those are combinations of the n constraints we've already seen.

Expect number of wasted rows before we see E constraints:

$$\sum_{n=0}^{E-1}\left(\frac{1}{1-2^{n-E}} - 1\right) = \sum_{n=0}^{E-1}\frac{1}{2^{E-n}-1} = 1 + 1/3 + 1/7 + \ldots$$

$$< 1 + 1/2 + 1/4 + \cdots < 2$$

(The sum is actually about 1.6)

Packet erasure channel

Split a video file into $K = 10,000$ packets and transmit

Some might be lost (dropped by switch, fail checksum, . . .)

Assume receiver knows the identity of received packets:
— Transmission and reception could be synchronized
— Or large packets could have unique ID in header

If packets are 1 bit, this is the BEC.

Digital fountain methods provide cheap, easy-to-implement codes for erasure channels. They are *rateless*: no need to specify M, just keep getting packets. When slightly more than K have been received, the file can be decoded.

Digital fountain (LT) code

Packets are sprayed out continuously
Receiver grabs any $K' > K$ of them (e.g., $K' \approx 1.05K$)
Receiver knows packet IDs n, and encoding rule

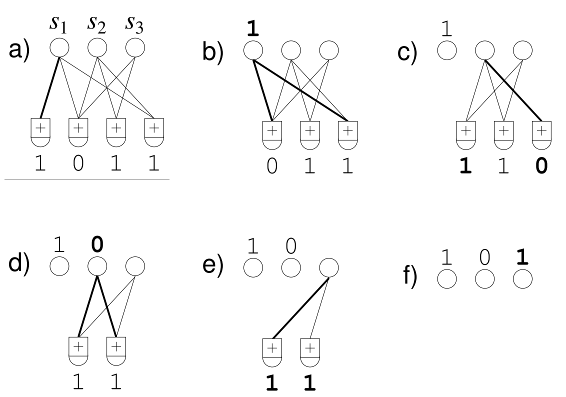
Encoding packet n:

Sample d_n pseudo-randomly from a degree distribution $\mu(d)$
Pick d_n pseudo-random source packets
Bitwise add them mod 2 and transmit result.

Decoding:

1. Find a check packet with $d_n = 1$
Use that to set corresponding source packet
Subtract known packet from all checks
Degrees of some check packets reduce by 1. GoTo 1.

LT code decoding



Soliton degree distribution

Ideal wave of decoding always has one $d = 1$ node to remove

"Ideal soliton" does this in expectation:

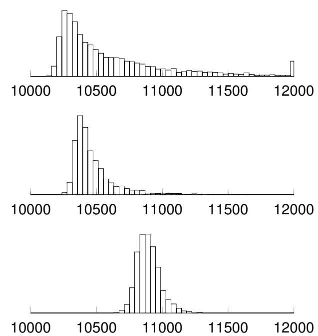
$$\rho(d) = \begin{cases} 1/K & d = 1 \\ 1/d(d-1) & d = 2, 3, \ldots, K \end{cases}$$

(Ex. 50.2 explains how to show this.)

A robustified version, $\mu(d)$, ensures decoding doesn't stop and all packets get connected. Still get $R \to C$ for large K.

A Soliton wave was first observed in 19 C Scotland on the Union Canal.

Number of packets to catch



$K = 10,000$ source packets

Numbers of transmitted packets required for decoding on random trials for three different packet distributions.

MacKay, p593

Reed–Solomon codes (sketch mention)

Widely used: e.g. CDs, DVDs, Digital TV

k message symbols $\rightarrow$ coefficients of degree $k-1$ polynomial

Evaluate polynomial at $> k$ points and send

Some points can be erased:
Can recover polynomial with any k points.

To make workable, polynomials are defined on Galois fields.

Reed–Solomon codes can be used to correct bit-flips as well as erasures: like identifying outliers when doing regression.

Information Theory

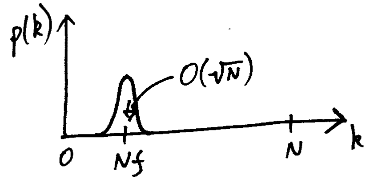
http://www.inf.ed.ac.uk/teaching/courses/it/

Week 8
Noisy channel coding theorem and LDPC codes

Iain Murray, 2012
School of Informatics, University of Edinburgh

Typical sets revisited

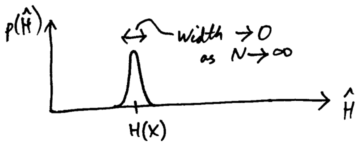
Week 2: looked at $k = \sum_i x_i,\;\; x_i \sim \mathrm{Bernoulli}(f)$



Saw number of 1's is almost always in narrow range around expected number. Indexing this 'typical set' was the cost of compression.

Typical sets: general alphabets

More generally look at $\hat{H} = \frac{1}{N}\sum_i \log \frac{1}{P(x_i)},\;\; x_i \sim P$



Define typical set: $\mathbf{x} \in T_{N,\beta}$ if $\left|\frac{1}{N}\log\frac{1}{P(\mathbf{x})} - H(X)\right| < \beta$

For any β, $P(\mathbf{x} \in T_{N,\beta}) > 1-\delta$, for any δ if N big enough

See MacKay, Ch. 4

Source Coding Theorem

(MacKay, p82–3 for details)

Min probability in $T_{N,\beta}$ is $2^{-N(H(X)+\beta)}$

Therefore typical set has size $\leq 2^{N(H(X)+\beta)}$

For large N can set β small

Index almost all strings with $\log_2 2^{NH(X)} = NH(X)$ bits

We now extend ideas of typical sets to joint ensembles of inputs and outputs of noisy channels. . .

Jointly typical sequences

For $n = 1 \ldots N$: $x_n \sim \mathbf{p}_X$

Send $\mathbf{x}$ over extended channel: $y_n \sim Q_{\cdot|x_n}$

Jointly typical:
$(\mathbf{x},\mathbf{y}) \in J_{N,\beta}$ if $\left|\frac{1}{N}\log\frac{1}{P(\mathbf{x},\mathbf{y})} - H(X,Y)\right| < \beta$

There are $\leq 2^{N(H(X,Y)+\beta)}$ jointly typical sequences

Chance of being jointly typical

$(\mathbf{x},\mathbf{y})$ from channel are jointly typical with prob $1-\delta$

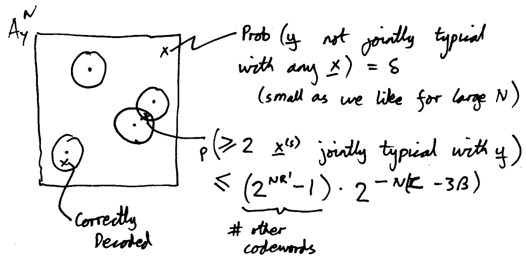
$(\mathbf{x}',\mathbf{y}')$ generated independently are rarely jointly typical

$$\begin{aligned} P(\mathbf{x}',\mathbf{y}' \in J_{N,\beta}) &= \sum_{(\mathbf{x},\mathbf{y})\in J_{N,\beta}} P(\mathbf{x})\,P(\mathbf{y}) \\ &\leq |J_{N,\beta}|\, 2^{-N(H(X)-\beta)}\, 2^{-N(H(Y)-\beta)} \\ &\leq 2^{N(H(X,Y)-H(X)-H(Y)+3\beta)} \\ &\leq 2^{-N(I(X;Y)-3\beta)} \\ &\leq 2^{-N(C-3\beta)}, \quad \text{for optimal } \mathbf{p}_X \end{aligned}$$

Random typical set code

Randomly choose $S = 2^{NR'}$ codewords $\{\mathbf{x}^{(s)}\}$

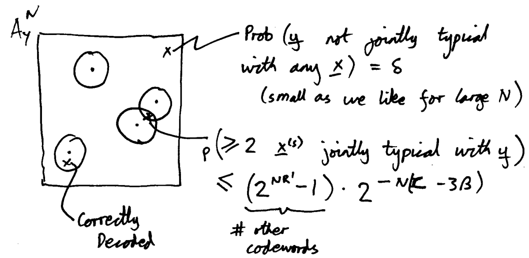
Decode $\mathbf{y} \rightarrow \hat{s}$ if $(\mathbf{y},\mathbf{x}^{(\hat{s})}) \in J_{N,\beta}$
and no other $(\mathbf{y},\mathbf{x}^{(s')}) \in J_{N,\beta}$



Error rate averaged over codes

Set rate $R' < C-3\beta$. For large N prob. confusion $< \delta$

Total error probability on average $< 2\delta$



Error for a particular code

We randomly drew all the codewords for each symbol sent.

Block error rate averaged over all codes:

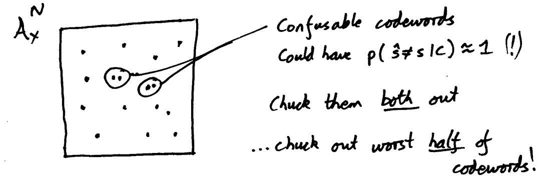
$$\langle p_B \rangle \equiv \sum_{\mathcal{C}} P(\hat{s} \neq s \,|\, \mathcal{C})\, P(\mathcal{C}) < 2\delta$$

Some codes will have error rates more/less than this

There exists *a* code with block error:

$$p_B(\mathcal{C}) \equiv P(\hat{s} \neq s \,|\, \mathcal{C}) < 2\delta$$

Worst case codewords



Maximal block error: $p_{BM}(\mathcal{C}) \equiv \max_s P(\hat{s} \neq s \,|\, s, \mathcal{C})$ could be close to 1.

$p_{BM} < 4\delta$ for expurgated code.

Now have $2^{NR'-1}$ codewords, rate $= R' - 1/N$.

Noisy channel coding theorem

For N large enough, can shrink β's and δ's close to zero.

For large N a code exists with rate close to C with error close to zero. (As close as you like for large enough N.)

In the 'week 7' notes we showed that it is impossible to transmit at rates greater than the capacity, without non-negligible probability of error for particular channels. This is also true in general.

Code distance

Distance, $d \equiv \min_{s,s'} \left|\mathbf{x}^{(s)} - \mathbf{x}^{(s')}\right|$

E.g., $d\!=\!3$ for the $[7,4]$ Hamming code

Can *always* correct $\lfloor (d-1)/2 \rfloor$ errors

Distance of random codes?

$\left|\mathbf{x}^{(s)} - \mathbf{x}^{(s')}\right| \approx \frac{N}{2}$ for large N

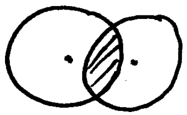
Not *guaranteed* to correct errors in $\geq \frac{N}{4}$ bits

With BSC get $\approx Nf$ errors, and proof works for $f > \frac{1}{4}$

Distance isn't everything

Distance can sometimes be a useful measure of a code

However, good codes have codewords that aren't separated by twice the number of errors we want to correct

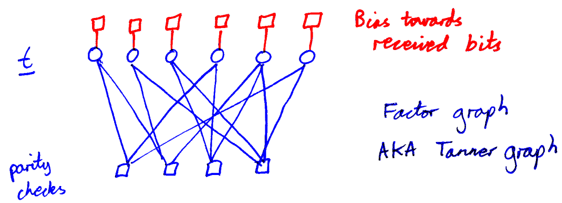


In high-dimensions the overlapping volume is tiny.

Shannon-limit approaching codes for the BSC correct *almost all* patterns with Nf errors, even though they can't strictly correct *all* such patterns.

Low Density Parity Check codes

LDPC codes originally discovered by Gallagher (1961)
Sparse graph codes like LDPC not used until 1990s.



Prior over codewords $P(\mathbf{t}) \propto \mathbb{I}(H\mathbf{t}\!=\!\mathbf{0})$
Posterior over codewords $P(\mathbf{t}\,|\,\mathbf{r}) \propto P(\mathbf{t})\, Q(\mathbf{r}\,|\,\mathbf{t})$

Why Low Density Parity Check (LDPC) codes?

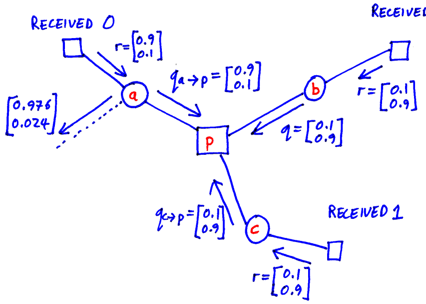
For some channels, the noisy channel coding theorem can be reproved for randomly generated linear codes. However, not all ways of generating *low-density* codes, with each variable only involved in a few parity checks and vice-versa, are very good.

For some sequences of low-density codes, the Shannon limit is approached for large block-lengths.

For both uniformly random linear codes, or random LDPC codes, the results are for optimal decoding: $\hat{\mathbf{t}} = \mathrm{argmax}_{\mathbf{t}}\, P(\mathbf{t}\,|\,\mathbf{r})$. This is a hard combinatorial optimization problem in general. The reason to use low-density codes is that we have good approximate solvers: use the sum-product algorithm (AKA "loopy belief propagation") — decode if the thresholded beliefs give a setting of $\mathbf{t}$ that satisfies all parity checks.

Sum-Product algorithm

Example with three received bits and one parity check



p336 MacKay, p399 Bishop "Pattern recognition and machine learning"

Sum-Product algorithm notes:

Beliefs are combined by element-wise multiplying
Two types of messages: variable $\to$ factor and factor $\to$ variable
Messages combine beliefs from all neighbours except recipient

Variable $\to$ factor:

$$q_{n\to m}(x_n) = \prod_{m' \in \mathcal{M}(n)\backslash m} r_{m'\to n}(x_n)$$

Factor $\to$ variable:

$$r_{m\to n}(x_n) = \sum_{\mathbf{x}_m \backslash n} \left(f_m(\mathbf{x}_m) \prod_{n' \in \mathcal{N}(m)\backslash n} q_{n'\to m}(x_{n'}) \right)$$

Example $r_{p\to a}$ in diagram, with sum over $(b,c) \in \{(0,0),(0,1),(1,0),(1,1)\}$
$r_{p\to a}(0) = 1\!\times\!0.1\!\times\!0.1 + 0 + 0 + 1\!\times\!0.9\!\times\!0.9 = 0.82$
$r_{p\to a}(1) = 0 + 1\!\times\!0.1\!\times\!0.9 + 1\!\times\!0.9\!\times\!0.1 + 0 = 0.18$

More Sum-Product algorithm notes:

Messages can be renormalized, e.g. to sum to 1, at any time.

I did this for the outgoing message from a to an imaginary factor downstream. This message gives the relative beliefs about about the settings of a given the graph we can see:

$$b_n(x_n) = \prod_{m' \in \mathcal{M}(n)} r_{m' \to n}(x_n)$$

The settings with maximum belief are taken and, if they satisfy the parity checks, used as the decoded codeword.

The beliefs are the correct posterior marginals if the factor graph is a tree. Empirically the decoding algorithm works well on low-density graphs that aren't trees. Loopy belief propagation is also sometimes used in computer vision and machine learning, however, it will not give accurate or useful answers on all inference/optimization problems!

We haven't covered efficient implementation which uses Fourier transform tricks to compute the sum quickly.

Information Theory

http://www.inf.ed.ac.uk/teaching/courses/it/

Week 9
Hashes and lossy memories

Iain Murray, 2012
School of Informatics, University of Edinburgh

Course overview

Source coding / compression:
— Losslessly representing information compactly
— Good probabilistic models → better compression

Noisy channel coding / error correcting codes:
— Add redundancy to transmit without error
— Large pseudo-random blocks approach theory limits
— Decoding requires large-scale inference (cf Machine learning)

Other topics in information theory
— Cryptography: not covered here
— Over capacity: using fewer bits than info. content
— Rate distortion theory
— Hashing

Rate distortion theory (taster)

Q. How do we store N bits of information with $N/3$ binary symbols (or N uses of a channel with $C = 1/3$)?

A. We can't without a non-negligible probability of error. But what if we were forced to try?

Idea 1:
— Drop $2N/3$ bits on the floor
— Transmit $N/3$ reliably
— Let the receiver guess the remaining bits

Expected number of errors: $2N/3 \cdot 1/2 = N/3$

Can we do better?

Reversing a block code

Swap roles of encoder and decoder for $[N, K]$ block code

E.g., Repetition code R_3

Put message through decoder first, transmit, then encode

110111010001000 → 11000 → 111111000000000

111 and 000 sent without error. Other six blocks lead to one error. Error rate $= 6/8 \cdot 1/3 = 1/4$, which is $< 1/3$

Slightly more on MacKay p167–8, much more in Cover and Thomas.

Rate distortion theory plays little role in practical lossy compression systems for (e.g.) images. It's a challenge to find practical coding schemes that respect perceptual measures of distortion.

Hashing

Hashes reduce large amounts of data into small values

(obviously the info. content of a source is not preserved in general)

Computers, humans and other animals can do amazing things, very quickly, based on tiny amounts of information.

Understanding how to use hashes can make progress in cognitive science and practical information systems.

Some of this is long-established computer science

A surprising amount is fertile research ground

Hashing motivational examples:

Many animals can do amazing things. While: `http://www.google.com/technology/pigeonrank.html` was a hoax. The paper on the next slide and others like it are not.

It isn't just pigeons. *Amazingly* humans can do this stuff too. Paul Speller demonstrated that humans can remember to distinguish similar pictures of pigeons over many minutes(!). `http://www.webarchive.org.uk/wayback/archive/20100223122414/http://www.oneandother.co.uk/participants/PaulSpeller`

How can we build systems that rapidly recall arbitrary labels attached to large numbers of rich but noisy media sources? YouTube has recently done this on a *very* large scale for copyright enforcement.

Some web browsers rapidly prove that a website isn't on a malware black-list without needing to access an external server, or needing an explicit list of all black-listed sites. (False positives can be checked with a request to an external server.)

Journal of Experimental Psychology:
Animal Behavior Processes
1984, Vol. 10, No. 2, 256–271

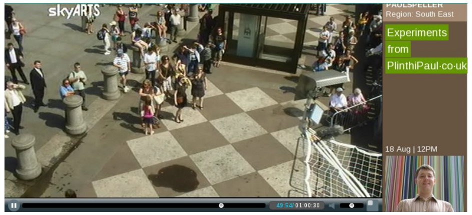
Pigeon Visual Memory Capacity

William Vaughan, Jr., and Sharon L. Greene
Harvard University

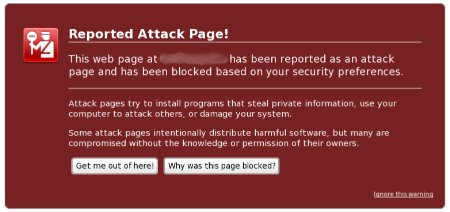
This article reports on four experiments on pigeon visual memory capacity. In the first experiment, pigeons learned to discriminate between 80 pairs of random shapes. Memory for 40 of those pairs was only slightly poorer following 490 days without exposure. In the second experiment, 80 pairs of photographic slides were learned; 629 days without exposure did not significantly disrupt memory. In the third experiment, 160 pairs of slides were learned; 731 days without exposure did not significantly disrupt memory. In the fourth experiment, pigeons learned to respond appropriately to 40 pairs of slides in the normal orientation and to respond in the opposite way when the slides were left-right reversed. After an interval of 751 days, there was a transient disruption in discrimination. These experiments demonstrate that pigeons have a heretofore unsuspected capacity with regard to both breadth and stability of memory for abstract stimuli and pictures.

Remembering images

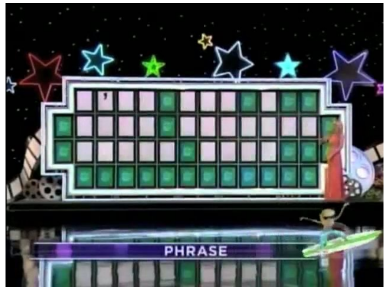
Remembering images



'Safe browsing'

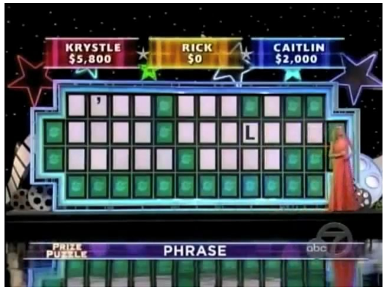


Information retrieval



Wheel of Fortune, Nov 2010

Information retrieval

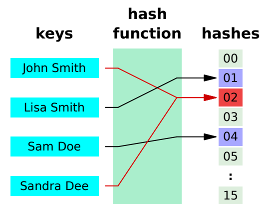


Information retrieval

Hash functions

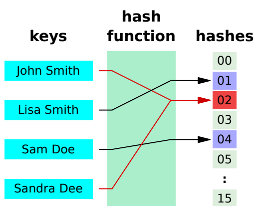
A common view:

file $\to$ b bit string (maybe like random bits)



Many uses: e.g., integrity checking, security, communication with feedback (`rsync`), indexing for information retrieval

Hash Tables



Hash indexes table of pointers to data

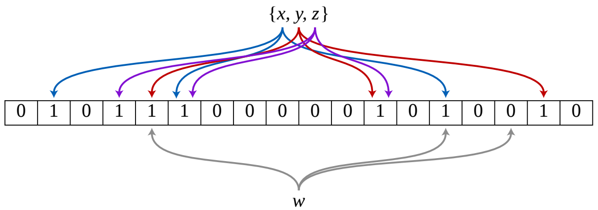
When hash table is empty at index, can *immediately* return '`Not found`'

Need to resolve conflicts. Ways include:

— List of data at each location. Check each item in list.
— Put pointer to data in next available location. Deletions need 'tombstones', rehash when table is full
— 'Cuckoo hashing': use > 1 hash and recursively move pointers out of the way to alternative locations.

Bloom Filters

Hash files multiple times (e.g., 3)
Set (or leave) bits equal to 1 at hash locations



Immediately know haven't seen w: ≥ 1 bits are zero

Notes on Bloom filters

Probability of false negative is zero

Probability of false positive depends on number of memory bits, M, and number of hash functions, K.

For fixed large M the optimal K (ignoring computation cost) turns out to be the one that sets $\approx 1/2$ of the bits to be on. This makes sense: the memory is less informative if sparse.

Other things we've learned are useful too. One way to get a low false positive rate is to make K small but M huge. This would have a huge memory cost…except we could compress the sparse bit-vector. This can potentially perform better than a standard Bloom filter (but the details will be more complicated).

Google Chrome uses (or at least used to use) a Bloom filter with $K = 4$ for its safe web-browsing feature.

Hashing in Machine Learning

A couple of example research papers

Semantic Hashing (Salakhutdinov & Hinton, 2009)
— Hash bits are "latent variables" underlying data
— 'Semantically' close files $\rightarrow$ close hashes
— Very fast retrieval of 'related' objects

Feature Hashing for Large Scale Multitask Learning,
(Weinberger et al., 2009)
— 'Hash' large feature vectors without (much) loss in (spam) classification performance.
— Exploit multiple hash functions to give millions of users personalized spam filters at only about twice the cost (time and storage) of a single global filter(!).