Information Theory

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

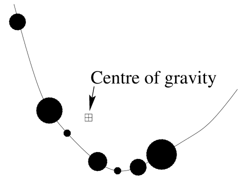
Week 4
Compressing streams

Iain Murray, 2014

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 $(\mathbb{E}[x], \mathbb{E}[f(x)])$, which is above $(\mathbb{E}[x], f(\mathbb{E}[x]))$

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.

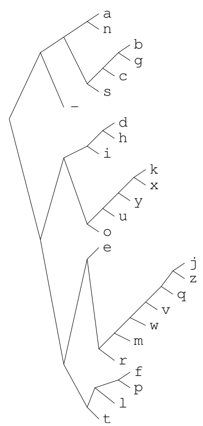
We have been assuming symbols are generated i.i.d. with known probabilities p_i.

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 $\mathcal{A}_X = \{\texttt{a–z}, \_\}$: 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

$\Rightarrow$ "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.

Product rule / Chain rule

$$P(A, B \,|\, H) = P(A \,|\, H)\, P(B \,|\, A, H) = P(B \,|\, H)\, P(A \,|\, B, H)$$

$$= P(A \,|\, H)\, P(B \,|\, H) \quad \text{iff independent}$$

$$P(A,B,C,D,E) = P(A)\, \underbrace{P(B,C,D,E \,|\, A)}_{P(B \,|\, A)\, \underbrace{P(C,D,E \,|\, A,B)}_{P(C \,|\, A,B)\, \underbrace{P(D,E \,|\, A,B,C)}_{P(D \,|\, A,B,C)\, P(E \,|\, A,B,C,D)}}}$$

$$P(\mathbf{x}) = P(x_1) \prod_{d=2}^{D} P(x_d \,|\, \mathbf{x}_{<d})$$

Revision of the product rule:

An identity like $P(A, B) = P(A)\, P(A \,|\, B)$, is true for any variables or collections of variables A and B. You can be explicit about the current situation, by conditioning every term on any other collection of variables: $P(A, B \,|\, H) = P(A \,|\, H)\, P(B \,|\, A, H)$. You can also swap A and B throughout, as these are arbitrary labels.

The second block of equations shows repeated application of the product rule. Each time different groups of variables are chosen to be the 'A', 'B' and 'H' in the identity above. The multivariate distribution is factored into a product of one-dimensional probability distributions (highlighted in blue).

The final line shows the same idea, applied to a D-dimensional vector $\mathbf{x} = [x_1, x_2, \ldots x_D]^\top$. This equation is true for the distribution of *any* vector. In some probabilistic models we choose to drop some of the high-dimensional dependencies $\mathbf{x}_{<d}$ in each term. For example, if $\mathbf{x}$ contains a time series and we believe only recent history affects what will happen next.

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

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$\boxtimes$, $\cdots$

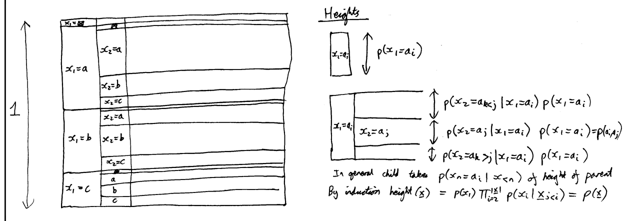
c$\boxtimes$, ca$\boxtimes$, $\cdots$, cb$\boxtimes$, $\cdots$, cc$\boxtimes$, $\cdots$, cccccc... cc$\boxtimes$

Arithmetic Coding

We give all the strings a binary codeword

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

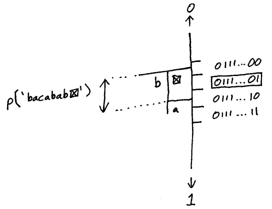
Create a tree of strings 'top-down':



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

Arithmetic Coding

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



$\Rightarrow$ Encode bacabab☒ with 0111...01

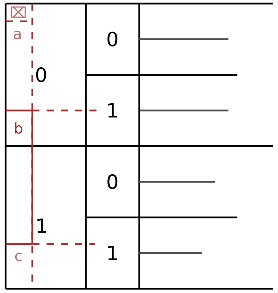
Navigate string tree to find interval on real line.

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

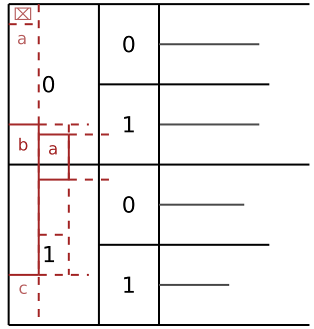
Arithmetic Coding

Overlay string tree on binary symbol code tree.
Goal: encode bac☒



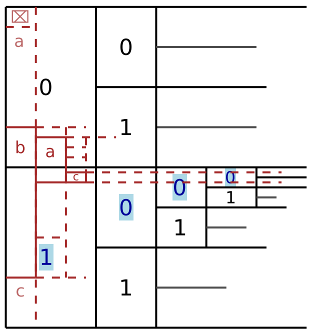
From $P(x_1)$ distribution can't begin to encode 'b' yet

Arithmetic Coding



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

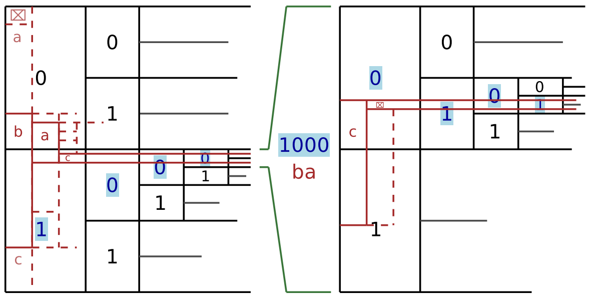
Arithmetic Coding



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

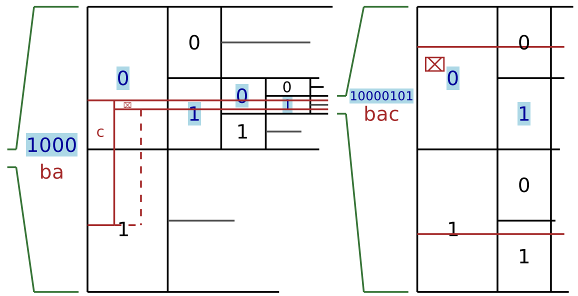
Arithmetic Coding

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



From $P(x_4 \,|\, \text{bac})$. Encoding of 'bac☒' starts 10000101...

Arithmetic Coding

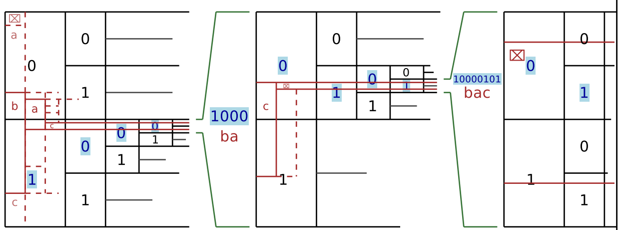


1000010101 uniquely decodes to 'bac☒'

10000101**10** would also work: slight inefficiency

Arithmetic Coding

Zooming out...

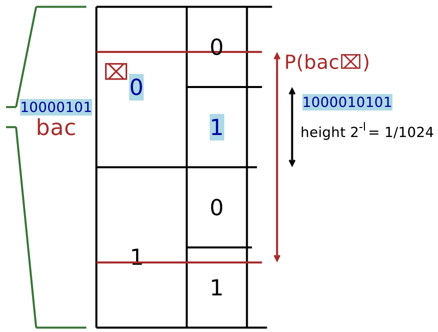


1000010101 uniquely decodes to 'bac☒'

10000101**10** would also work: slight inefficiency

Arithmetic Coding

Zooming in...



$2^{-l} > 4\,P(\mathbf{x} = \text{bac☒}) \quad \Rightarrow \quad l < -\log P(\mathbf{x} = \text{bac☒}) + 2$ bits

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

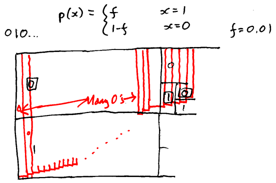
Large blocks of symbols are compressed together: possibly your whole file. The inefficiencies of symbol codes have been removed.

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: $D_{\text{KL}}(P \,||\, Q)$.

AC and sparse files

Finally we have a practical coding algorithm for sparse files



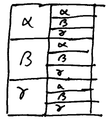
(You could make a better picture!)

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

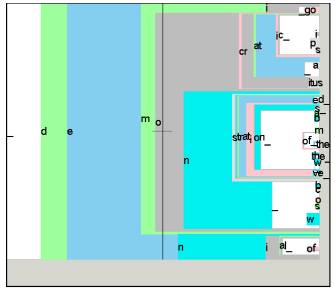
Can overlay string on any other indexing of [0,1] line



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/