3F1 Information Theory Course

Jossy Sayir
js851@cam.ac.uk

November 15, 2011

Contents

Pre-requisites

This course assumes familiarity with the following Part IB courses:

- **Probability** (Paper 7);
- **Signal and Data Analysis** (Paper 6);

Book / Paper List

- **T. M. Cover and J. A. Thomas**, 'Elements of Information Theory', 2nd ed. (Wiley, 2006).

  A very good and comprehensive coverage of all the main aspects of Information Theory. Very clearly written.

- **D. J. C. MacKay**, 'Information Theory, Inference, and Learning Algorithms' (CUP, 2003).

  This book explores the whole topic of Information Theory in a very approachable form. It goes well beyond exam requirements in 3F1, but will be very useful for those interested in a broader understanding of the subject. The book is downloadable **FREE** in pdf (11 MB) from:

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

  This version should not be printed. A printed version is produced by CU Press, and costs about £35.

- **C. E. Shannon**, 'A Mathematical Theory of Communication', in The Bell System Technical Journal, Vol. 27, pp. 379-423, 623-656, July & October, 1948.

  A reprinted version is downloadable **FREE** in pdf (0.4 MB) from:

  http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf

Acknowledgement

These course notes were developed and written by Nick Kingsbury, with only minor edits by Jossy Sayir in the current 2011 edition. The course was developed from an earlier course on Information Theory, given by Dr Tom Drummond until December 2004. We would like to thank him very much for allowing us to base much of this course on his lecture notes and examples. We are also grateful to Dr Albert Guillen i Fabregas, who has provided valuable comments on the communications related aspects of the course.

Jossy Sayir and Nick Kingsbury.

1 Introduction to Information Theory

Information theory is concerned with many things. It was originally developed for designing codes for transmission of digital signals (Shannon, late 1940s).

Other applications include:

- Data compression (e.g. audio using MP3 and images using JPEG)
- Signal analysis
- Machine learning

We will begin by looking at the transmission of **digital data** over the internet and other networks, and its storage in computers and on other media. The aim is to maximise throughput of error-free data and minimise file sizes.

1.1 Source Coding

This is concerned with the representation of messages as digital codes.

The aim is to make messages take up the smallest possible space.

A famous example for plain english text is:

A	● –	J	● – – –	S	● ● ●
B	– ● ● ●	K	– ● –	T	–
C	– ● – ●	L	● – ● ●	U	● ● –
D	– ● ●	M	– –	V	● ● ● –
E	●	N	– ●	W	● – –
F	● ● – ●	O	– – –	X	– ● ● –
G	– – ●	P	● – – ●	Y	– ● – –
H	● ● ● ●	Q	– – ● –	Z	– – ● ●
I	● ●	R	● – ●		

This is Morse Code (from about 1830) and has some interesting properties:

- The codes are not all the same length
- The code uses two symbols ... (or does it?)

How do you decode this?

● ● ● ● ● ● – ● ● ● – ● ● – – – ● – – – – – ● – ● ● – ● ● – ● ●

The duration of the space between symbols in morse is very important as it allows us to distinguish 3 E's from an S, for example.

1.2 Variable length codes

If we want to transmit information as quickly as possible, it is useful to use codes of varying lengths if the source generates symbols with varying probabilities.

We can use shorter codes for common symbols and longer codes for rare ones.

e.g. in Morse code E = •, whereas Q = – – • – .

From now on we shall consider only **binary codes** – i.e. codes using only 0 and 1 to form their codewords.

Constraints:

1. There must be no ambiguity in the code
   (so for binary codes, "0" and "00" cannot both be codewords).
2. It must be possible to recognise a codeword as soon as the last bit arrives
   (so "0" and "01" cannot both be codewords).

A code which satisfies these two conditions is called an **Instantaneous Code**.

Examples

A source emits a sequence of symbols from the set $\{A, B, C, D\}$

Two different codes for transmitting these symbols are proposed:

	Code 1	Code 2
A	00	0
B	01	10
C	10	110
D	11	111

Which of these codes is better?

If all 4 symbols are equally likely, then Code 1 is best:

- Average word length of Code 1 $= \dfrac{2+2+2+2}{4} =$ 2 bits.
- Average word length of Code 2 $= \dfrac{1+2+3+3}{4} =$ 2.25 bits.

BUT if symbol A is more likely than B, and B is more likely than C or D, then the average word length of Code 2 reduces, while that of Code 1 stays the same.

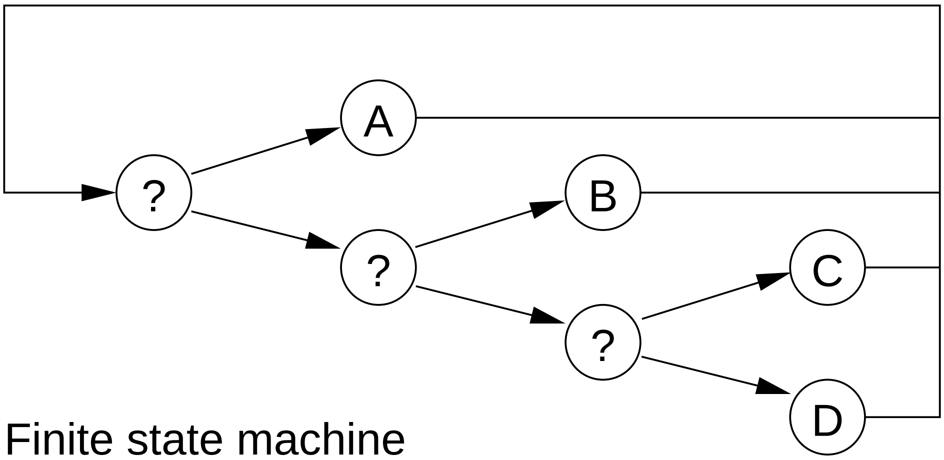
1.3 Instantaneous Codes

An alternative way to phrase the constraints is to state the **Prefix Condition:**

No codeword may be the prefix of any other codeword

(A prefix of a codeword means a consecutive sequence of digits starting at the beginning)

This condition allows us to represent (and decode) the code using a tree:



Finite state machine

A	0
B	10
C	110
D	111

Labelling the upper output of each query node as 0 and the lower output as 1 corresponds to the code words on the left.

This diagram represents a finite state machine for decoding the code.

1.4 Average Code Length

An important property of a code is the average length L of the codewords (averaged according to the probabilities). For N symbols with probabilities p_i and codeword lengths l_i bits, if W is a random variable whose value is the length of the codeword assigned to a source symbol, then L is given by the expectation of the lengths:

$$L = E[W] = \sum_{i=1}^{N} p_i \, l_i \text{ bits}$$

Example

Consider the two codes given in the example above.

If the symbol probabilities and codeword lengths are given by:

	p_i	Code 1 l_i	Code 2 l_i
A	0.6	2	1
B	0.2	2	2
C	0.1	2	3
D	0.1	2	3

Average word length for Code 1 is:

$$0.6 \times 2 + 0.2 \times 2 + 0.1 \times 2 + 0.1 \times 2 = 1.2 + 0.4 + 0.2 + 0.2 = 2 \text{ bits}$$

Average word length for Code 2 is:

$$0.6 \times 1 + 0.2 \times 2 + 0.1 \times 3 + 0.1 \times 3 = 0.6 + 0.4 + 0.3 + 0.3 = 1.6 \text{ bits}$$

1.5 Source Entropy

We need a mathematical foundation for choosing efficient codes. This requires a way of measuring:

1. Information content of a Source
2. Information content of each symbol
3. Efficiency of a code

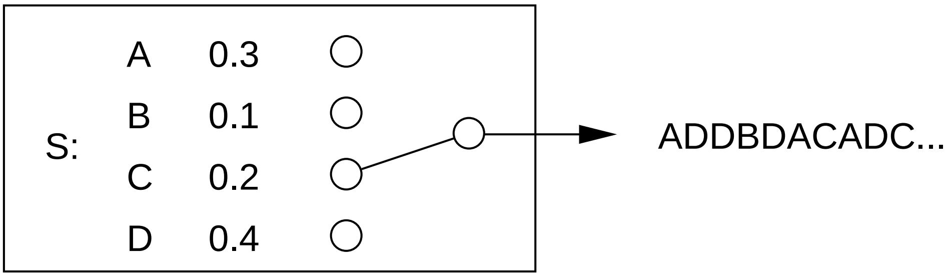
Shannon (1948) had a basic insight:

The information content of a message depends on its a priori probability

- You winning the national lottery – **rare**, so lots of information.
- A typical party political broadcast – **predictable**, so not much information.

The most information is associated with **rare** events.

Consider an order-0 process (also called a **memoryless** process), in which a source S selects each symbol i with probability P_i $(i = 1..N)$.



We want a measure of the average information per symbol $H(S)$.

What properties should $H(S)$ have?

1. It should be continuous in the probabilities p_i.
2. If the p_i are equal then H(S) should increase with N
3. If S_1 and S_2 are two **independent** message sources then a combined source (S_1, S_2) should have

$$H(S_1, S_2) = H(S_1) + H(S_2)$$

(i.e. the information obtained by combining two independent message sources is the sum of the information from each source – e.g. an 8-bit message can be regarded as the sum of two 4-bit messages).

We now show that these properties are satisfied if $H(S)$ is of the form:

$$H(S) = -K \sum_{i=1}^{N} p_i \log(p_i) \qquad \text{where} \quad K > 0$$

Shannon called $H(S)$ the **information entropy** because of its similarity to the concept of entropy in statistical mechanics, where p_i represents the probability of being in a particular thermal microstate i (as in Boltzmann's H-theorem).

Considering the above 3 properties in turn, using $H(S)$ defined above:

1. $p_i \log(p_i)$ is continuous for each p_i in the range $0 \le p_i \le 1$,
   so $H(S)$ will also be continuous in the p_i. (Note: $p \log(p) = 0$, when $p = 0$.)

2. If the p_i are equal, then $p_i = \dfrac{1}{N}$. Hence

   $$H(S) = -K \sum_{i=1}^{N} \tfrac{1}{N} \log(\tfrac{1}{N}) = K \sum_{i=1}^{N} \tfrac{1}{N} \log(N) = K \log(N)$$

   If $H(S)$ is to increase with N, then K must be positive.

3. Let S_1 and S_2 be sources with N_1 and N_2 symbols respectively, and with probabilities $p_1 \ldots p_{N_1}$ and $q_1 \ldots q_{N_2}$. Hence

   $$\sum_{i=1}^{N_1} p_i = 1 \qquad \text{and} \qquad \sum_{j=1}^{N_2} q_j = 1$$

   Entropies of S_1 and S_2 are then

   $$H(S_1) = -K \sum_{i=1}^{N_1} p_i \log(p_i) \quad \text{and} \quad H(S_2) = -K \sum_{j=1}^{N_2} q_j \log(q_j)$$

   and the combined source (S_1, S_2) will have $N_1 N_2$ states with probabilities $p_i q_j$, so

   $$\begin{aligned} H(S_1, S_2) &= -K \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} p_i q_j \log(p_i q_j) \\ &= -K \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} p_i q_j [\log(p_i) + \log(q_j)] \\ &= -K \sum_{i=1}^{N_1} p_i \log(p_i) \sum_{j=1}^{N_2} q_j - K \sum_{j=1}^{N_2} q_j \log(q_j) \sum_{i=1}^{N_1} p_i \\ &= -K \sum_{i=1}^{N_1} p_i \log(p_i) - K \sum_{j=1}^{N_2} q_j \log(q_j) \\ &= H(S_1) + H(S_2) \qquad \text{as required.} \end{aligned}$$

The key to this final result lies in the relationship

$$\log(p_i q_j) = \log(p_i) + \log(q_j)$$

which shows why the entropy expression must contain a $\log(p)$ term and not any other non-linear function.

In appendix 2 on p. 28 of Shannon's 1948 paper (see book/paper list), he proves that the above logarithmic form for $H(S)$ is the only form which satisfies the above 3 properties.

1.6 Properties of the source entropy

$$H(S) = -K \sum_{i=1}^{N} p_i \log(p_i)$$

The choice of K is arbitrary (as long as it is positive) and equates to choosing a base for the logarithm. The convention is to to use **base 2 logs** and **choose** $K = 1$, so that

$$H(S) = -\sum_{i=1}^{N} p_i \log_2(p_i)$$

The unit of entropy is then **bits** (this name, in wide use nowadays, was suggested to Shannon by Tukey, inventor of the Fast Fourier Transform, as a shorterning of "binary digits").

If natural logs are used instead (base e), the unit of entropy is **nats.** It is easy to see that 1 bit is $\ln 2$ nats, and the entropy in bits can be expressed as

$$H(S) = -\frac{1}{\ln(2)} \sum_{i=1}^{N} p_i \ln(p_i)$$

For a source with $N = 2^n$ equiprobable states, the entropy is $\log_2 N = n$ bits, which is the word-length for a simple uniform-length binary code for this source.

Reminder about logs:

Let $y = 2^x = (e^{\ln(2)})^x = e^{x \ln(2)}$

Take base 2 and base e logs: $x = \log_2(y)$ and $x \ln(2) = \ln(y)$

Therefore $\log_2(y) = \dfrac{\ln(y)}{\ln(2)}$

This works for any bases a and b, not just for 2 and e.

Examples:

1. Two symbols with probability 0.5 (coin flip):

$$H(S) = -\,0.5\log_2(0.5) - 0.5\log_2(0.5) = \log_2(2) = 1\text{ bit}$$

2. Four symbols with probability 0.25:

$$H(S) = -\,0.25\log_2(0.25)\times 4 = \log_2(4) = 2\text{ bits}$$

3. The roll of a 6-sided (fair) die:

$$H(S) = -\,\tfrac{1}{6}\log_2(\tfrac{1}{6})\times 6 = \log_2(6) = 2.585\text{ bits}$$

4. Four symbols with probabilities 0.6, 0.2, 0.1 and 0.1 (the earlier example code):

$$\begin{aligned} H(S) &= -\,0.6\log_2(0.6) - 0.2\log_2(0.2) - \;0.1\log_2(0.1) - 0.1\log_2(0.1) \\ &= 0.4422 + 0.4644 + 0.3322 + 0.3322 = 1.5710\text{ bits.} \end{aligned}$$

5. Two independent sources:

	Symbol	Probability p_i	$-p_i\log_2 p_i$
Source S_1	A	0.7	0.3602
	B	0.3	0.5211
Source S_2	α	0.5	0.5
	β	0.5	0.5

Entropies:

$$H(S_1) = 0.3602 + 0.5211 = 0.8813\text{ bit} \qquad H(S_2) = 0.5 + 0.5 = 1\text{ bit}$$

	Symbol	Probability	$-p_i\log_2 p_i$
Source S $= (S_1, S_2)$	Aα	0.35	0.5301
	Aβ	0.35	0.5301
	Bα	0.15	0.4105
	Bβ	0.15	0.4105

Joint entropy:

$$H(S) = 0.5301 + 0.5301 + 0.4105 + 0.4105 = 1.8813\text{ bits}$$

which shows that:

$$H(S) = H(S_1) + H(S_2)$$

Properties of the source entropy (cont.)

Note that the entropy of a random variable S takes the form of an expectation:

$$H(S) = -\sum_{i=1}^{N} p_i \log_2(p_i) = E[-\log_2(p_S(S))]$$

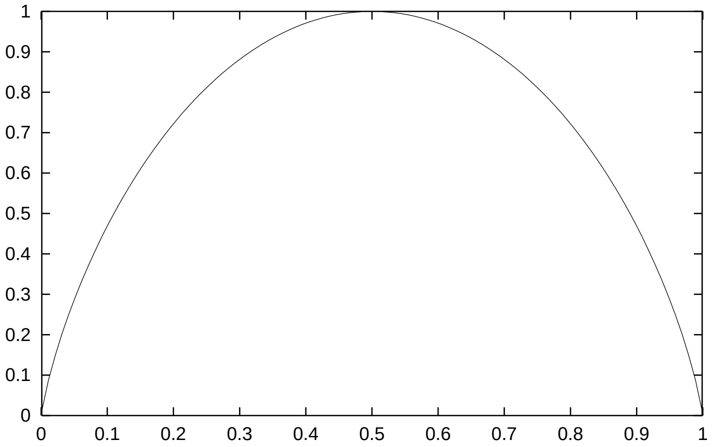
So the information associated with each symbol is $-\log_2(p_i)$

The entropy is maximised when all symbols are equiprobable.

For a source with two symbols with probabilities p and $(1-p)$:

$$H(S) = -\left[\ p \log_2(p) + (1-p) \log_2(1-p)\ \right]$$

as shown below. This function of p is commonly called the **binary entropy function** and denoted $h(p)$.



In general we may prove that for a code with N symbols, the entropy achieves a maximum value of $\log_2 N$ when all the probabilities equal $\frac{1}{N}$.

To do this we calculate

$$H(S) - \log_2 N = -\sum_{i=1}^{N} p_i(\log_2 p_i + \log_2 N) = \sum_{i=1}^{N} p_i \log_2(\frac{1}{Np_i})$$

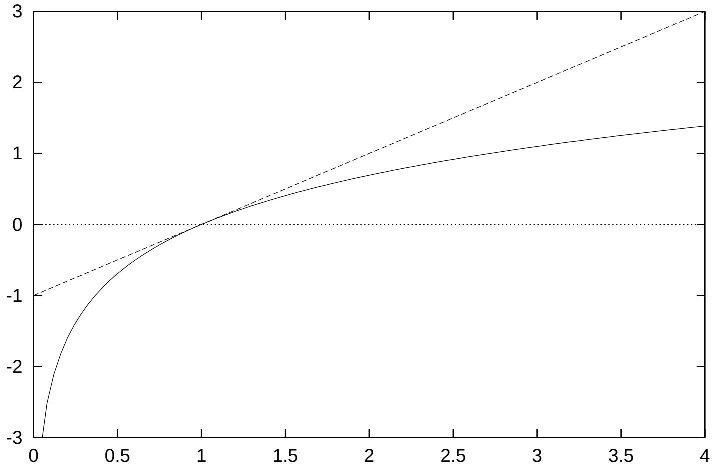
Then we use the relations:

$$\log_2 x = \frac{\ln x}{\ln 2} \qquad \text{and} \qquad \ln x \leq x - 1 \ \ \text{(prove by simple calculus)}$$

to show that $H(S) - \log_2 N \leq 0$, with the equality occurring when $Np_i = 1$ for all p_i.

Completion of this proof is examples sheet question 1. Note, we could, as an alternative, use calculus directly on the p_i terms, but this is not so easy because the p_i must sum to unity at all times so they can't be varied independently.

Plot of $y = \ln(x)$ and $y = x - 1$



1.7 Information rate and redundancy

A source S uses an alphabet of N symbols, has entropy H_S bit/symbol and produces f symbols per second.

The **information rate** of the source is $f\, H_S$ bit/sec.

As discussed above, the maximum entropy an N-ary source can have is

$$H_{\max} = -\log_2(\tfrac{1}{N}) = \log_2 N \quad \text{bit/symbol}$$

The **redundancy** of the source is defined as $(H_{\max} - H_S)$ bit/symbol.

2 Good Variable-Length Codes

2.1 Expected Code Length

Recall from section 1.4 that the average codeword length is $L = \sum_{i=1}^{N} p_i l_i$ bits.

To maximise message throughput on a channel with a given bit rate, we want to minimise L.

We will find out that $L \geq H(S)$.

So we can define the **Rate Efficiency** as $\eta = H(S)/L$.

Examples:

1. Two equiprobable symbols with code 0 and 1:

   $p_0 = p_1 = 0.5$ $\qquad$ $l_0 = l_1 = 1$ bit

   $H(S) = 0.5 \times 1 + 0.5 \times 1 = 1$ bit

   $L = 0.5 \times 1 + 0.5 \times 1 = 1$ bit

   $\eta = 1$

2. Four symbols with probabilities and two codes given in the table below

Symbol	Probability	Code 1	Code 2
A	0.125	00	000
B	0.125	01	001
C	0.25	10	01
D	0.5	11	1

   Probabilities are the same for both codes so $H(S)$ is the same.

   $H(S) = 0.125 \times 3 + 0.125 \times 3 + 0.25 \times 2 + 0.5 \times 1 = 1.75$ bits

   $L_1 = 0.125 \times 2 + 0.125 \times 2 + 0.25 \times 2 + 0.5 \times 2 = 2$ bits

   $\eta_1 = 1.75/2 = 0.875$

   $L_2 = 0.125 \times 3 + 0.125 \times 3 + 0.25 \times 2 + 0.5 \times 1 = 1.75$ bits

   $\eta_2 = 1.75/1.75 = 1.0$

2.2 Kraft Inequality

The Kraft inequality for a valid code with codeword lengths l_i, $i = 1 \dots N$, which satisfies the prefix condition, states that

$$\sum_{i=1}^{N} 2^{-l_i} \leq 1$$

For example consider code 2 from example above:

Symbol	Code	l_i	2^{-l_i}
A	000	3	0.125
B	001	3	0.125
C	01	2	0.25
D	1	1	0.5

Proof

Let m = maximum codeword length.

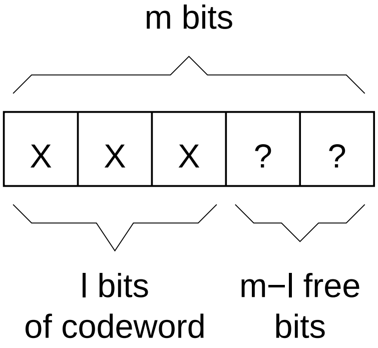
Consider the 2^m possible words of length m

Assign each one to the codeword that is its prefix (if it has one).

There can be at most one codeword which is the prefix of any word of length m since if there were two codewords which were both prefixes then the shorter one would be a prefix for the longer one.

A codeword of length l is the prefix of 2^{m-l} words of length m since there are $m - l$ free bits after the codeword.

2^m words	codeword (prefix)	l_i
000	000	3
001	001	3
010	01	2
011		
100	1	1
101		
110		
111		



So for the example above, $m = 3$. The codeword 01 has length 2 and is the prefix for $2^{3-2} = 2^1 = 2$ words of length 3 (namely 010 and 011).

Adding up the words of length m assigned to each codeword gives:

$$\sum_{i=1}^{N} 2^{m-l_i} \leq 2^m \qquad \text{Hence} \quad \sum_{i=1}^{N} 2^{-l_i} \leq 1$$

The Kraft inequality is a sufficient condition as well as a necessary one. So if you have a set of lengths that satisfy the inequality, then a set of codewords with those lengths exists. The codewords can easily be generated using the same trick as in the proof:

> Find the longest codeword, make a list of all words of that length in ascending (or descending) binary order. Taking the code lengths in ascending order, assign 2^{m-l_i} words to each code and record the common prefix (of length l_i) as the codeword.

2.3 Bounds on Codeword Length

We will show that a code can be found such that

$$H(S) \leq L < H(S) + 1$$

This states two things. First that the average codeword length must be at least the entropy of the source. Second that it is possible to find a code for which the average codeword length is less than 1 bit more than the entropy.

Tackling the LHS of the inequality first, consider $H(S) - L$

$$\begin{aligned}
H(S) - L &= \sum_i -p_i \log_2(p_i) - \sum_i p_i l_i \;=\; \sum_i p_i \left(\log_2(\frac{1}{p_i}) - l_i \right) \\
&= \sum_i p_i \;\log_2(\frac{2^{-l_i}}{p_i}) \;=\; \frac{1}{\ln(2)} \sum_i p_i \;\ln(\frac{2^{-l_i}}{p_i}) \\
&\leq \frac{1}{\ln(2)} \sum_i p_i \left(\frac{2^{-l_i}}{p_i} - 1 \right) \qquad \text{since } \ln(x) \leq x - 1 \\
&= \frac{1}{\ln(2)} \left(\sum_i 2^{-l_i} - \sum_i p_i \right) \leq 0 \qquad \text{(Kraft inequality)}
\end{aligned}$$

Note that if $p_i = 2^{-l_i}$ then $H(S) = L$ since we can substitute $\log_2(1)$ into the 2nd line above.

2.4 Shannon Fano Coding

We can show the RHS inequality of the codelength bound by designing a code that will always satisfy it.

First we choose integer codelengths l_i such that

$$l_i = \lceil \log_2(\frac{1}{p_i}) \rceil$$

This means that

$$\log_2(\frac{1}{p_i}) \leq l_i < \log_2(\frac{1}{p_i}) + 1$$

The LHS of this says $2^{-l_i} \leq p_i$, and, since $\sum p_i \leq 1$, we obey the Kraft inequality. So a code exists (we'll actually generate the code in a minute).

Multiplying the RHS by p_i and summing to get L gives

$$L = \sum_i p_i l_i < \sum_i p_i \left(\log_2(\frac{1}{p_i}) + 1 \right) = H(S) + 1$$

So this choice of codelengths will do it, but how do we actually produce a code?

Shannon's original scheme (invented independently by Fano) worked as follows:

1. Sort the probabilities into descending order;
2. Make a cumulative probability table (adding each probability to all those above it);
3. Expand the cumulative probabilities as a binary fraction;
4. Take the first l_i digits in the binary fraction as the code.

For example, consider a source that produces six symbols with probabilities 0.34, 0.19, 0.17, 0.1, 0.1, 0.1.

Symbol	p_i	l_i	$\sum p_{i-1}$	binary	S-F code	Better code
A	0.34	2	0	0.000000	00	00
B	0.19	3	0.34	0.010101	010	01
C	0.17	3	0.53	0.100001	100	100
D	0.1	4	0.7	0.101100	1011	101
E	0.1	4	0.8	0.110011	1100	110
F	0.1	4	0.9	0.111001	1110	111

This Shannon-Fano (S-F) code wastes some of the codespace - by inspection we see that the codes 011, 1010, 1101 and 1111 could be added without violating the prefix condition. Alternatively they can be combined with existing codewords to shorten them and give the better (more efficient) code in the final column. To do this, 011 was combined with 010 to generate the shorter code 01, and then 1010 with 1011 to give 101, etc.

How did Shannon know that the S-F algorithm would always generate a valid code?

Well the codes get longer, so he only had to make sure that each code was not a prefix of any that followed it. But the choice of codelength ensures that we add at least 1 to the least significant bit of the code. For example the probability of C is more than 2^{-3} so we know that the first three 'decimal' (actually binary) places of the cumulative probability for D must be at least $0.100 + 0.001 = 0.101$, so C cannot be a prefix for D (or anything that follows).

In the above example the entropy of the source is 2.4156 bits.

The average length of the S-F code in the table is

$$L_{SF} = 0.34 \times 2 + 0.19 \times 3 + 0.17 \times 3 + 0.1 \times 4 \times 3 = \text{2.96 bits}$$

This still satisfies the bound that it should be no more than 1 bit worse than the entropy. However the 'better' code gets much closer with an average length of

$$L_{\text{better}} = 0.34 \times 2 + 0.19 \times 2 + 0.17 \times 3 + 0.1 \times 3 \times 3 = \text{2.47 bits}$$

We can see quickly that the S-F code is not likely to be very good because, for this code

$$\sum_i 2^{-l_i} = 0.25 + 2 \times 0.125 + 3 \times 0.0625 = 0.6875$$

which is a lot less than the permitted Kraft upper limit of unity. Hence we are only using about 69% of the available code space.

The 'better' code in this case equals the Kraft upper limit and uses 100% of the code space, because we have combined all unused regions of code space with used codewords in order to shorten them without violating the prefix condition.

2.5 Huffman Coding

The Shannon Fano algorithm creates reasonably efficient codes, but we have seen the code is still wasteful. There exists an optimal algorithm (that guarantees to minimise the average codeword length and use 100% of the codespace) called the **Huffman coding algorithm**, as follows.

Given symbols $\{x_1 \ \ldots \ x_N\}$ with probabilities $\{p_1 \ \ldots \ p_N\}$:

1. Make a list of symbols with their probabilities (in descending order helps).
2. Find the two symbols in the list with the smallest probabilities and create a new node by joining them. Label the two new branches with 0 and 1 using some predefined rule (e.g. upper branch $= 0$).
3. Treat the new node as a single symbol with probability equal to the sum of the two joined symbols (which it now replaces in the list).
4. Repeat from step 2 until the root node is created with probability $= 1.0$.
5. Generate the codewords for each symbol, by starting from the root node and following the branches of the tree until each symbol is reached, accumulating the binary labels from step 2 to form each codeword in turn (root node symbol is first, i.e. on the left).

Using the same 6-symbol source as before, figs. 2.1 to 2.3 show the development of a Huffman code tree using this algorithm, after one, three and all iterations respectively. Note the codewords in fig. 2.3, derived in step 5 of the algorithm after the final (5th) iteration of steps 2 to 4. These codewords have the same lengths as the 'better' code in the above table and hence will give the same average length of 2.47 bits.

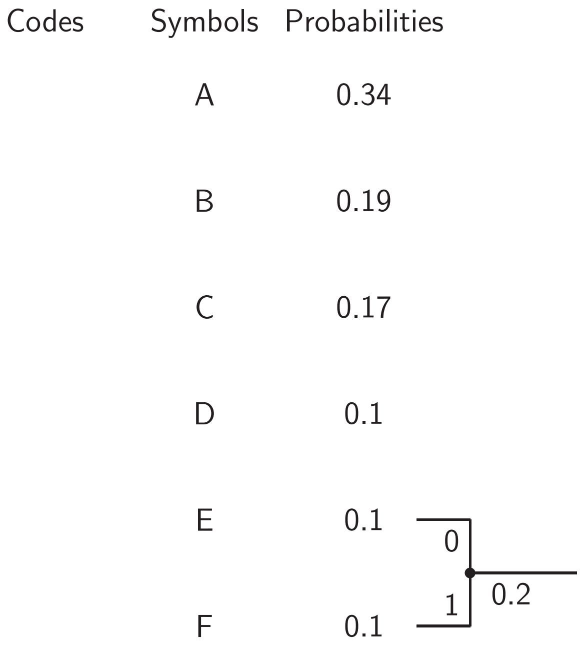


Fig 2.1: Huffman code tree - after node 1.

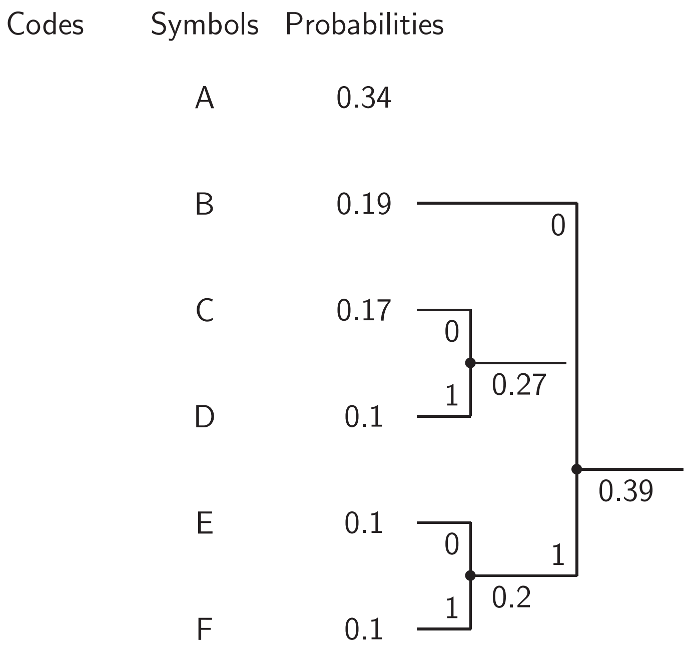


Fig 2.2: Huffman code tree - after nodes 1 to 3.

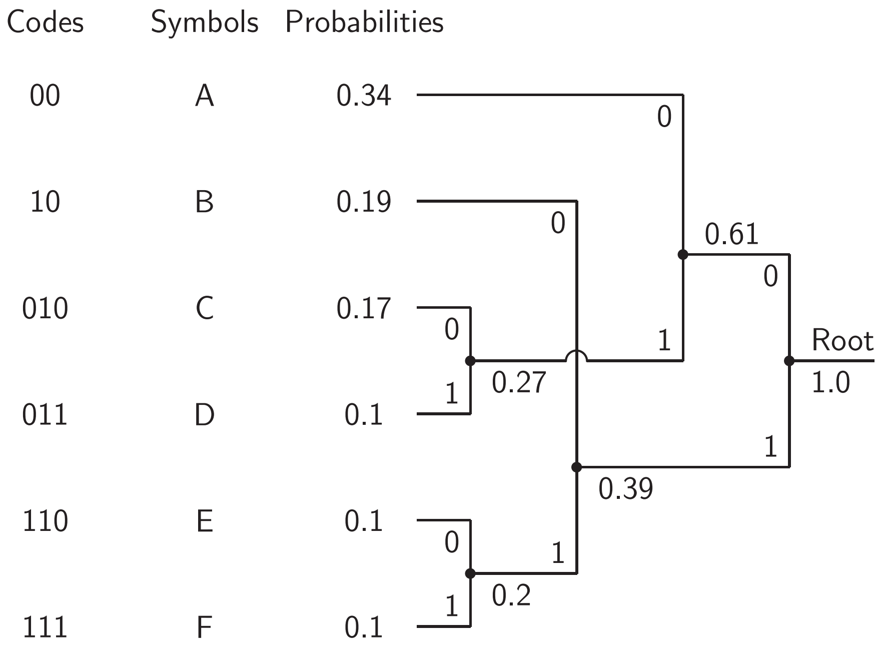


Fig 2.3: Huffman code tree - completed, with codewords.

2.6 Extension Coding

The efficiency of a code is defined as $\eta = H(S)/L.$

Recall that $H(S) \leq L < H(S) + 1$.

This inequality tells us that

$$1 \geq \eta > \frac{H(S)}{H(S)+1}$$

If $H(S) << 1$ then η can be very small.

E.g. for a source with two symbols and probabilities 0.999 and 0.001, $H(S) = 0.011$ but we have to use 1 bit per symbol so η is only 1.1% .

Huffman codes are usually very efficient ($\eta \sim 95$ to 99%) but if one of the p_i is significantly greater than 0.5, the code can become inefficient, especially if the p_i approaches unity.

A solution to this problem is to group sequences of symbols together in strings of length K. This is called **extending the source to order K**. If we have an alphabet of N symbols in the original source, we will then have an alphabet of N^K symbols in the extended source.

If the source is memoryless then the probability of a given combined symbol in the extended source is just the product of the probabilities of each symbol from the original source that occurs in the combination.

The entropy of the extended source is then simply

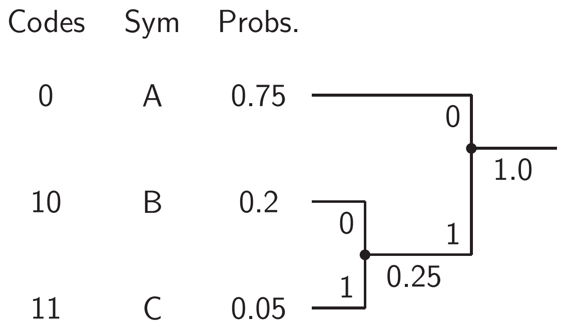
$$K \times H(S)$$

So η is bounded below by

$$\eta > \frac{KH(S)}{KH(S)+1}$$

which tends to 1 for large K. In practice K only needs to be large enough to bring the probability of the most common extended symbol down to approximately 0.5 .

Example



Codes	Sym	Probs.
0	A	0.75
10	B	0.2
11	C	0.05

Fig 2.4: Huffman code for 3-symbol source with one very likely symbol.

Codes	Sym	Probs.
0	AA	0.5625
10	AB	0.15
110	BA	0.15
11100	BB	0.04
11101	AC	0.0375
11110	CA	0.0375
111110	BC	0.01
1111110	CB	0.01
1111111	CC	0.0025

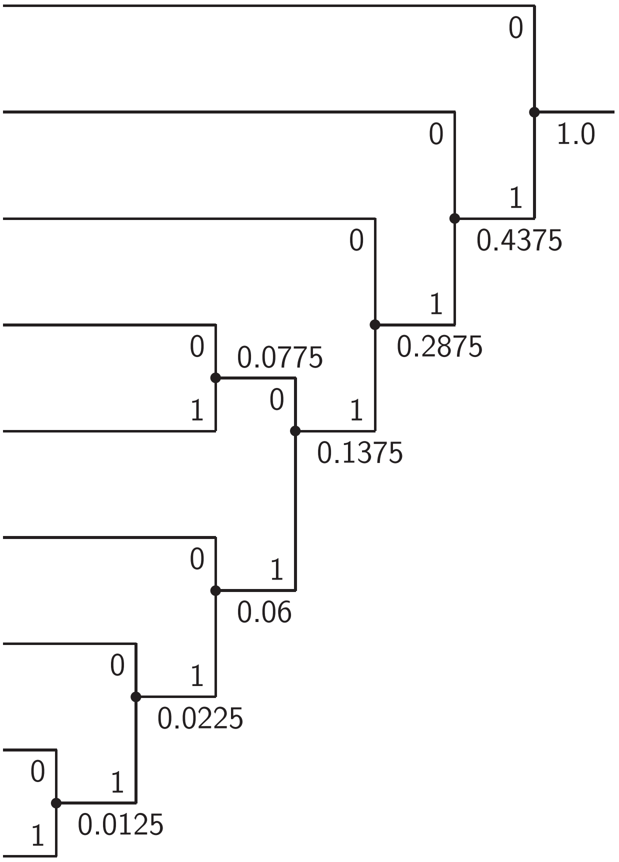


Fig 2.5: Huffman code for order-2 extended source with $3^2 = 9$ symbols.

The 3-symbol source in fig. 2.4 can be analysed as follows:

$$\begin{aligned} H(S) &= -[\,0.75\log_2(0.75) + 0.2\log_2(0.2) + 0.05\log_2(0.05)\,] \\ &= 0.9918 \text{ bits} \\ L_S &= 0.75 \times 1 + 0.2 \times 2 + 0.05 \times 2 \;=\; 1.25 \text{ bits} \\ \eta_S &= \frac{0.9918}{1.25} \;=\; 79.34\% \end{aligned}$$

If we extend to order 2, as in fig. 2.5, the results become:

$$\begin{aligned} H(S,S) &= 2\,H(S) \;=\; 1.9836 \text{ bits} \\ L_{S,S} &= 0.5625 \times 1 + 0.15(2+3) + 0.04 \times 5 + 0.0375(5+5) \\ &\quad + 0.01(6+7) + 0.0025 \times 7 \;=\; 2.035 \text{ bits} \\ \eta_{S,S} &= \frac{1.9836}{2.035} \;=\; 97.47\% \end{aligned}$$

Note the dramatic increase in efficiency with just order-2 extension!

2.7 Arithmetic Coding

Arithmetic Coding is an algorithm for extending the order to the entire length of the message using a form of Shannon-Fano coding. The inefficiencies of S-F coding become negligible when the order is large. The technique is often attributed to M.I.T. professor Peter Elias because he described it to Norman Abrahamson who mentioned it in a note in his 1963 textbook. In a conversation I had with Peter Elias on April 6, 1998, he vehemently denied having come up with the technique and told me that many people were discussing this technique at M.I.T. at the time and it may have been suggested by Bob Gallager, another M.I.T. professor. I then spoke to Bob Gallager who categorically denied being the inventor of the technique, and claimed that it may have been part of Shannon's lecture at M.I.T., for which no written record is available. The name "arithmetic" is derived from Shannon's description of S-F coding in his 1948 paper, where he refers to it as "the arithmetic process described above". The technique as described below would require an infinite precision calculator. Therefore, an essential aspect of the technique in terms of practical applicability is its realisation in finite precision. This was invented by R. Pasco in his 1976 Stanford University PhD dissertation and later also published by Rissanen and Langdon from IBM research.

The algorithm maintains two values which are the lower and upper ends of a range starting at 0 and 1. This range is then partitioned into sub-ranges according to the probability of each symbol. The appropriate sub-range is then selected to become the new range according to the symbol emitted by the source. Fig. 2.6 shows this for a 4-symbol source and the message $BCABD$.

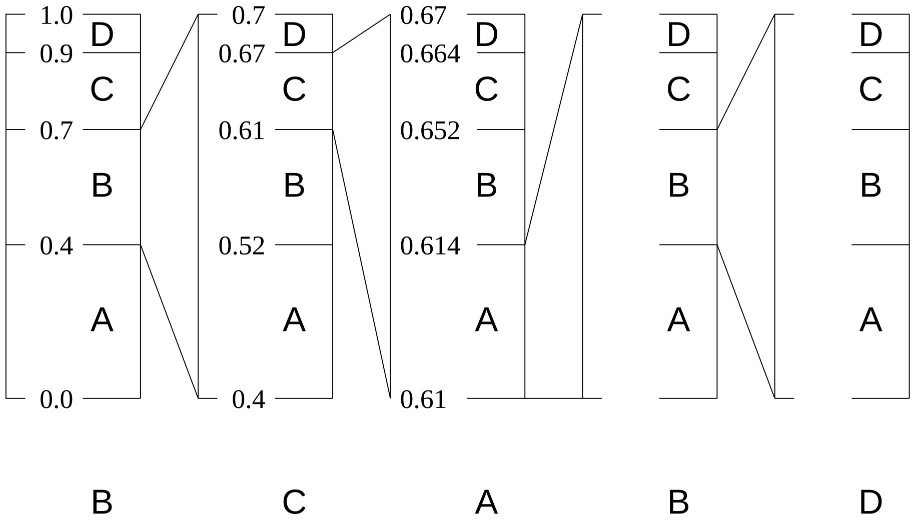


Fig. 2.6: Arithmetic coding of a 4-symbol source with probabilities $\{0.4, 0.3, 0.2, 0.1\}$.

When the message is complete, the fraction is converted to binary and the appropriate number of bits are transmitted over the channel. The precision required (and hence the number of bits) is determined by the size of the final sub-range, which is just the product of the probabilities of all the symbols in the message.

The fraction produced in the above example is

$$\underbrace{0.4+0.3}_{B}(\underbrace{0.7+0.2}_{C}(\underbrace{0+0.4}_{A}(\underbrace{0.4+0.3}_{B}(\underbrace{0.9+0.1}_{D}\ r)))) = 0.6261+0.0007r$$

where $0 \leq r < 1$ is a remainder variable which defines the final sub-range.

To prevent any ambiguity in the coded message, the final sub-range must be larger than 2^{-n} where n is the length of the fraction in bits. Now $2^{-11} < 0.0007$ and hence 11 bits of precision are needed in this case.

It is interesting to compare this arithmetic code with Huffman coding for this message. The word lengths for an order-1 Huffman code for this source are $\{1, 2, 3, 3\}$ for the symbols $\{A, B, C, D\}$ respectively. Hence our message $BCABD$ would require

$$2 + 3 + 1 + 2 + 3 = 11 \text{ bits}$$

which is the same as for the arithmetic code.

We would need a much longer message to be able to see any significant difference between the performance of the codes as they are both very efficient.

The main advantage of arithmetic codes is that they can easily be designed to adapt to changing statistics of the symbols as the message progresses, and it is not necessary to know the symbol probabilities before the coding starts. It is very difficult to do this with Huffman codes, which must be designed in advance.

For these reasons, arithmetic codes are now used extensively in JPEG-2000 image coders and in MPEG-4 (high-definition) video coders. Note that there are special scaling techniques that can be used in practice, to prevent arithmetic precision problems when the message is long and the sub-ranges get very small indeed.

A lot more detail on arithmetic codes is given in chapter 6 of David MacKay's online book.

3 Higher Order Sources

3.1 Higher order sources

Until now we have dealt with simple sources where the probabilities for successive symbols are independent.

This is not true for many sources (e.g. English text).

If a “Q” appears in the signal, there is a high probability that the next symbol will be a “U”. So the “Q” is giving us information about the next symbol.

More generally, symbol n provides information about symbol $n + 1$.

We should be able to exploit this to make a more efficient code.

Examples of English text

The following samples of text were generated by a random number generator, which picked letters and spaces using the same probability distributions as occur in natural english. The only difference is that in the 1st, 2nd and 3rd-order cases, the probabilities of the current letter depended on what letter or letters came before the one being picked.

0 order Markov model: (letters are independent)

OCRO HLI RGWR NMIELWIS EU LL NBNESEBYA TH EEI ALHENHTTPA OOBTTVA NAH BRL

1st order Markov model: (each pdf is conditional on the previous letter)

ON IE ANTSOUTINYS ARE T INCTORE ST BE S DEAMY ACHIN D ILONASIVE TUCOOWE AT TEASONARE FUSO TIZIN ANDY TOBE SEACE CTISBE

2nd order Markov model: (each pdf is conditional on the two previous letters)

IN NO IST LAT WHEY CRATICT FROURE BERS GROCID PONDENOME OF DEMONSTRURES OF THE REPTAGIN IS REGOACTIONA OF CRE

3rd order Markov model: (each pdf is conditional on the three previous letters)

THE GENERATED JOB PROVIDUAL BETTER TRAND THE DISPLAYED CODE ABOVERY UPONDULTS WELL THE CODERST IN THESTICAL IT DO HOCK BOTHE MERG INSTATES CONS ERATION NEVER ANY OF PUBLE AND TO THEORY EVENTIAL CALLEGAND TO ELAST BENERATED IN WITH PIES AS IS WITH THE

Note the dramatic increase in similarity to english in the higher order models.

To analyse these sort of effects, we need some information theory mathematics about non-independent random variables.

3.2 Non-independent variables

If two random variables X and Y are independent then for any given (x, y):

$$P(x, y) = P(x)P(y)$$

If the variables are not independent then $P(x, y)$ can be arbitrary, so we must look at the joint probability distribution $P(X, Y)$ of the combined variables. Here is an example:

$P(X, Y)$:

X Y	a	b	c	$\sum_X$
α	0.3	0.02	0.01	0.33
β	0.02	0.3	0.02	0.34
γ	0.01	0.02	0.3	0.33
$\sum_Y$	0.33	0.34	0.33	1.0

For X we can compute the probability that it takes the value a, b or c by summing the probabilities in each column of the table. We can then compute the entropy of X:

$$H(X) = H([0.33\ \ 0.34\ \ 0.33]^T) = 1.5848 \text{ bits}; \qquad \text{and similarly for } Y.$$

(For convenience I used the Matlab function entropy(p) given below.)

We can also compute the entropy of X given $Y = \alpha, \beta$ or γ:

$$\begin{aligned}
H(X|Y=\alpha) &= -\textstyle\sum_i p(x_i|\alpha)\log_2 p(x_i|\alpha) \\
&= H([0.3\ \ 0.02\ \ 0.01]^T/0.33) \ = \ 0.5230 \text{ bits} \\
H(X|Y=\beta) &= -\textstyle\sum_i p(x_i|\beta)\log_2 p(x_i|\beta) \\
&= H([0.02\ \ 0.3\ \ 0.02]^T/0.34) \ = \ 0.6402 \text{ bits} \\
H(X|Y=\gamma) &= -\textstyle\sum_i p(x_i|\gamma)\log_2 p(x_i|\gamma) \\
&= H([0.01\ \ 0.02\ \ 0.3]^T/0.33) \ = \ 0.5230 \text{ bits}
\end{aligned}$$

Clearly these are each a lot lower than $H(X)$, and this is because there is high correlation between X and Y, as seen from the relatively large diagonal terms in the above 3×3 joint probability matrix. So if we know Y, we can make a much better guess at X because it is very likely to be the same as Y in this example.

We shall now look at this more formally.

```
function h = entropy(p)
% Calculate the entropies of processes defined by the columns of probability
% matrix p.  The max function overcomes problems when any elements of p = 0.
%
h = -sum(p .* log(max(p,1e-10))) / log(2);
return;
```

3.3 Mutual Information

The average entropy of X given Y can be computed by adding up these three entropies multiplied by the probability that $Y = \alpha,\ \beta$ or γ :

$$\begin{aligned} H(X|Y) &= H(X|Y=\alpha)\, p_Y(\alpha) + H(X|Y=\beta)\, p_Y(\beta) + H(X|Y=\gamma)\, p_Y(\gamma) \\ &= 0.5230 \ .\ 0.33 \ +\ 0.6402 \ .\ 0.34 \ +\ 0.5230 \ .\ 0.33 \\ &= 0.5628 \text{ bits} \end{aligned}$$

This entropy is much less than $H(X)$ which means that on average, the random variable X carries less information if you already know Y than if you don't.

In other words, knowing Y provides some information about X. This **mutual information** can be measured as the drop in entropy:

$$I(X;Y) = H(X) - H(X|Y)$$

In this case it is $1.5848 - 0.5628 =$ 1.0220 bits.

Based on the above arguments, we may obtain the following general expressions for conditional entropy and mutual information:

$$\begin{aligned} H(X|Y) &= -\sum_j p(y_j) \sum_i p(x_i|y_j) \log_2 p(x_i|y_j) \\ &= -\sum_j \sum_i p(x_i, y_j) \log_2 p(x_i|y_j) \qquad \text{since } p(x,y) = p(x|y)\ p(y) \\ I(X;Y) &= -\sum_i p(x_i) \log_2 p(x_i) + \sum_j \sum_i p(x_i, y_j) \log_2 p(x_i|y_j) \\ &= \sum_j \sum_i p(x_i, y_j) [\log_2 p(x_i|y_j) - \log_2 p(x_i)] \qquad \text{since } \sum_j p(x, y_j) = p(x) \\ &= \sum_j \sum_i p(x_i, y_j) \log_2 \left(\frac{p(x_i|y_j)}{p(x_i)} \right) \end{aligned}$$

$I(Y;X)$ can also be obtained in the same way and, using Bayes rule, it turns out that:

$$I(Y;X) = I(X;Y) \qquad \text{(see example sheet)}$$

Hence X provides the same amount of information about Y as Y does about X; so this is called the **mutual** information between X and Y.

3.4 Algebra of Mutual Information

We can also compute the entropy of the joint probability distribution

$$H(X,Y) = -\sum_i \sum_j p(x_i, y_j) \log_2 p(x_i, y_j) = -\sum_i \sum_j p_{i,j} \log_2(p_{i,j})$$

where the $p_{i,j}$ are the elements of matrix $P(X,Y)$.

In our example, $H(X,Y) = 2.1477$ bits.

We find that

$$\begin{aligned} H(X,Y) &= H(X) + H(Y|X) = H(Y) + H(X|Y) \\ &= H(X) + H(Y) - I(X;Y) \end{aligned}$$

In other words if I'm going to tell you the values of X and Y, I can tell you X first and Y second or vice versa. Either way, the total amount of information that I impart to you is the same, and it is $I(X;Y)$ less than if the two values were told independently.

Note that this all applies even when $P(X,Y)$ is not a symmetric matrix and X and Y have different probabilities. The only requirement on $P(X,Y)$ is that the sum of all its elements should be unity, so that it represents a valid set of probabilities.

3.5 Example: the Binary Symmetric Communications Channel

An important use of the Mutual Information concept is in the theory of digital communications. In section 2.2.1 of the 3F1 Random Processes course we considered the detection of binary signals in noise and showed how the presence of additive noise leads to a probability of error that can be calculated from the signal-to-noise ratio of the channel.

Let the probability of error on a channel U be p_e, and let the input and output of U be random processes X and Y. U is called **symmetric if p_e is the same for both states of its input X**. The mutual information of U is then

$$\begin{aligned} I(Y;X) &= H(Y) - H(Y|X) = H(Y) - \sum_x p(x)\ H(Y|X=x) \\ &= H(Y) - [p_X(0)h(p_e) + p_X(1)h(p_e)] \\ &= H(Y) - h(p_e) \end{aligned}$$

where $h(.)$ denotes the binary entropy function introduced in the previous section. Note that the probabilities of the output Y are given by

$$p(y) = (1-p_e)\ p(X=y) + p_e\ p(X \neq y) \quad \text{for } y = 0, 1$$

so if we know $p(X)$, we can calculate $p(Y)$ and hence $H(Y)$.

The mutual information of U tells us how much information is passed through U from X to Y.

The **Capacity** C_U of the channel U is defined (by Shannon) as the maximum of the mutual information over all possible choices for $p(X)$. In the case of the binary symmetric channel this occurs when $p(X) = [0.5 \;\; 0.5]$, so that $p(Y) = [0.5 \;\; 0.5]$ also and $H(Y) = 1$. Hence

$$C_U = \max_{p(X)} \; I(Y;X) = 1 - h(p_e)$$

The capacity of a channel tells us the maximum number of bits of information per transmitted symbol that we can expect to be able to pass through the channel error-free, assuming that a very good error-correcting code is used. Hence the reciprocal of this tells us the minimum amount of redundancy (code rate) that is likely to be needed in an error-correcting code for the channel.

For example if $p_e = 0.1$, then $h(p_e) = 0.469$ (see graph on page 12 of these notes), and so $C_U = 0.531$ bits per transmitted bit. Hence for every 1000 bits transmitted over the noisy channel, only 531 bits of information can be reliably passed (assuming a good error-correcting code is available to convert the raw bit error rate of 1-in-10 down to essentially zero error probability). An (ideal) error correction code with a rate of at least $531 : 1000 = 1 : 1.884$ would therefore be required to transmit reliably over this channel. Typically a code rate of approximately $1 : 2$ would be needed in practice, although some of the latest turbo or low-density parity check (LDPC) coding methods could get closer to theory than this.

These ideas can readily be extended to channels for non-binary sources and with non-symmetric error probabilities. In fact, the example probability matrix $P(X,Y)$ in section 3.2 (page 28) could apply to a 3-level communication channel with unequal error probabilities and a slightly unequal input distribution $p(X)$.

3.6 Markov sources

An interesting and useful class of dependent random sources are those in which the probability distribution for each new symbol (or state) in a stream depends only on a **finite number of previous symbols** (states), and not on the whole stream of previous symbols. These are know as Markov sources.

More formally we can define them as follows:

An **N^{th}-order Markov Source** is a stationary random process $S_0, S_1, S_2, \ldots$, such that the probability distribution of S_n conditional on $S_0, \ldots S_{n-1}$ is a function of $S_{n-N}, \ldots S_{n-1}$ only. (i.e. each S_n is **conditional only on the most recent N samples**.)

When N is small, this considerably simplifies the analysis of dependent sources, and it is often quite a good assumption to make in real systems.

Conditional probability tables

For a first-order Markov process, we usually represent the **transition probabilities** rather than the joint distribution in a table. The following table (matrix) shows the conditional probability distribution for S_n given S_{n-1}.

$P(S_n|S_{n-1})$:

S_{n-1}	A	B	C
S_n			
A	0.5	0.6	0.4
B	0.1	0.3	0.5
C	0.4	0.1	0.1
$\sum_{S_n}$	1.0	1.0	1.0

Note that each column of $P(S_n|S_{n-1})$ must sum to unity, since it is a **conditional** probability matrix (rather than the whole matrix summing to unity for a **joint** probability matrix). The rows do **not** need to sum to unity.

To work with this table, we usually need to know the equilibrium probability distribution, which is the probability that an individual symbol plucked from the stream takes the values A, B or C after the effect of any initial state S_0 has died away. These are the probabilities you would use if you modelled the source with a zero-order Markov model.

Now we may write

$$\begin{aligned}
P(S_n = A) \;&=\; P(S_n = A|S_{n-1} = A)\ P(S_{n-1} = A) \\
&\quad + P(S_n = A|S_{n-1} = B)\ P(S_{n-1} = B) \\
&\quad + P(S_n = A|S_{n-1} = C)\ P(S_{n-1} = C) \\
P(S_n = B) \;&=\; P(S_n = B|S_{n-1} = A)\ P(S_{n-1} = A) \\
&\quad + P(S_n = B|S_{n-1} = B)\ P(S_{n-1} = B) \\
&\quad + P(S_n = B|S_{n-1} = C)\ P(S_{n-1} = C) \\
P(S_n = C) \;&=\; P(S_n = C|S_{n-1} = A)\ P(S_{n-1} = A) \\
&\quad + P(S_n = C|S_{n-1} = B)\ P(S_{n-1} = B) \\
&\quad + P(S_n = C|S_{n-1} = C)\ P(S_{n-1} = C)
\end{aligned}$$

If we write $P(S_n)$ as a column vector:

$$P(S_n) = [\ P(S_n = A)\ \ P(S_n = B)\ \ P(S_n = C)\ ]^T$$

and similarly for $P(S_{n-1})$, then the above set of equations can be written much more simply as a matrix multiplication:

$$P(S_n) = P(S_n|S_{n-1})\ P(S_{n-1})$$

In the equilibrium state S_e, we require that

$$P(S_e) = P(S_n) = P(S_{n-1}) \qquad \text{and hence} \quad P(S_e) = P(S_n|S_{n-1})\ P(S_e)$$

Solving for $P(S_e)$ is a simple eigenvector problem, with $P(S_e)$ being the eigenvector of $P(S_n|S_{n-1})$ which has an eigenvalue of unity. We can easily verify that for the above matrix, the solution is

$$P(S_e) = [\ 0.5 \ \ 0.25 \ \ 0.25\ ]^T$$

In the case of probabilities, there is no uncertainty in the scaling of the eigenvector as this must sum to unity to be a valid probability distribution.

We can now compute the conditional entropy and mutual information of this 1st-order Markov source.

$H(S_n|S_{n-1})$ can be obtained by computing the entropy of each column of $P(S_n|S_{n-1})$, multiplying by the probability of that symbol from $P(S_e)$, and summing.

$$\begin{aligned} H(S_n|S_{n-1}=A) &= -0.5\log_2(0.5) - 0.1\log_2(0.1) - 0.4\log_2(0.4) = 1.3610 \\ H(S_n|S_{n-1}=B) &= -0.6\log_2(0.6) - 0.3\log_2(0.3) - 0.1\log_2(0.1) = 1.2955 \\ H(S_n|S_{n-1}=C) &= -0.4\log_2(0.4) - 0.5\log_2(0.5) - 0.1\log_2(0.1) = 1.3610 \end{aligned}$$

and so

$$H(S_n|S_{n-1}) = [\ 1.3610 \ \ 1.2955 \ \ 1.3610\ ]\ P(S_e) = 1.3446 \text{ bits}$$

Comparing this with the entropy of the symbols separately (zero-order model):

$$H(S_n) = H(S_{n-1}) = H(S_e) = -0.5\log_2(0.5) - 2\ .\ 0.25\log_2(0.25) = 1.5 \text{ bits}$$

Therefore the mutual information between adjacent symbols is

$$I(S_n; S_{n-1}) = H(S_n) - H(S_n|S_{n-1}) = 1.5 - 1.3446 = 0.1554 \text{ bits}$$

3.7 Coding a first order source

A simple order-1 Huffman coding scheme for the above source (assuming a zero-order Markov model) would use the probabilities $P(S_e)$, which happen to be integer powers of two, so a perfect Huffman code is possible with codewords $\{A = 0, B = 10, C = 11\}$. The mean length of this code equals the zero-order entropy and is 1.5 bits.

We can use the mutual information between symbols to make our coding scheme more efficient.

Huffman Coding

For Huffman coding, we can extend the source to some order > 1, say 3. We can then build a probability table for all sequences of 3 symbols. The first symbol has no prior information, so the probability comes from the equilibrium distribution, $P(S_e)$. The probabilities for the other two symbols then each depend on the symbol before.

The total entropy of sequences of 3 symbols is then:

$$\begin{aligned} H(S_n, S_{n+1}, S_{n+2}) &= H(S_n) + H(S_{n+1}|S_n) + H(S_{n+2}|S_{n+1}) \\ &= H(S_n) + 2H(S_{n+1}|S_n) \\ &= 3H(S_n) - 2I(S_{n+1}; S_n) \\ &= 3\ .\ 1.5 - 2\ .\ 0.1554 \\ &= 4.1892 \text{ bits} \;=\; 1.3964 \text{ bits/symbol} \end{aligned}$$

This source has $3^3 = 27$ message states and will be very efficiently represented by a Huffman code, so the real bit rate per symbol will be very close (within about 1%) of the entropy calculated above.

Alternatively we could use 3 separate Huffman codes to implement each column of the conditional probability matrix, choosing which one to use according to the previous symbol each time. Although, for a long sequence, this could in theory get down to just the conditional entropy, 1.3446 bits/symbol, in practice this method can only achieve 1.475 bits/symbol because the individual 3-state Huffman codes are too small to work efficiently, having word lengths of $\{1, 2, 2\}$ bits as their only option.

Arithmetic Coding

Arithmetic Coding is a way to obtain a coding rate very close to the conditional entropy, by coding a relatively long sequence of symbols at once.

To implement this, we just compute the code fractions for the initial symbol using the zero order distribution and then compute all the subsequent code fractions using the marginal distribution from each column of the conditional probability matrix, according to the previous symbol each time (as for the 3 Huffman codes above). As long as we do this over many symbols (≥ 100 say), then the coding efficiency will be very close to 100%, both because the higher entropy of the initial symbol will become a negligible fraction of the whole and because the arithmetic code guarantees to code within 1 bit of the total entropy of the source.

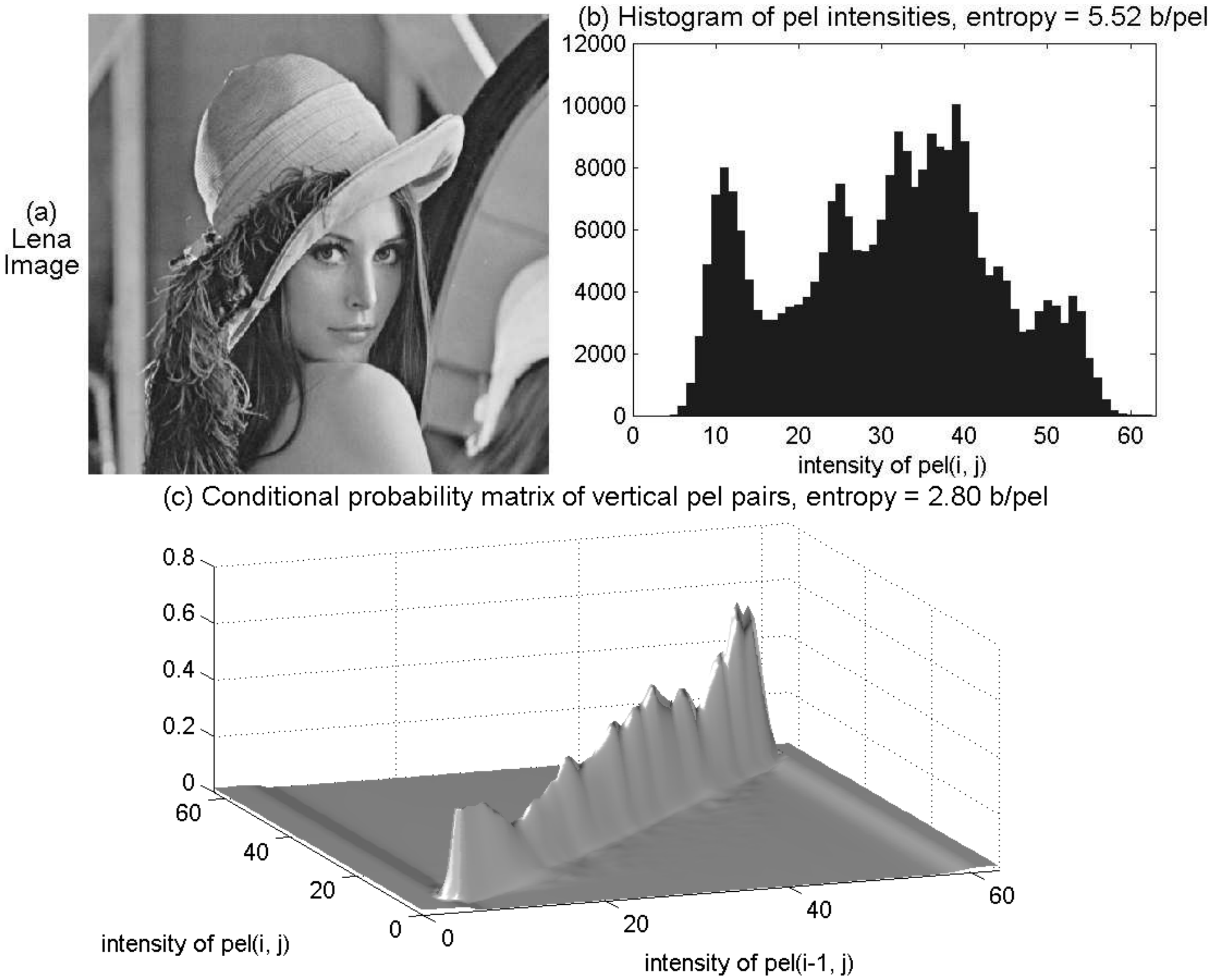


Fig. 3.1: Lena image, with its zero-order histogram of pixel intensities, and its first-order conditional probability matrix of pairs of vertically adjacent pixels.

3.8 Example: Lossless Image Coding

To illustrate these ideas on a more practical example, consider the problem of coding the pixels of a 512 × 512 image in the most efficient format which does not lose any information after the initial quantisation process. We will use the well-known 'Lena' image and will assume the pixels have been quantised to 64 levels, as shown in fig. 3.1a. This amount of quantisation is barely perceptible under most viewing conditions.

Hence the simplest code would be a uniform 64-state code, requiring 6 bits per pixel. The histogram for this quantised image is shown in fig. 3.1b, and the zero-order entropy of this image source $S_{i,j}$ (i.e. assuming uncorrelated pixels) is given by

$$H_0(S_{i,j}) = -\sum_{k=0}^{63} p_k \log_2(p_k) \qquad \text{where} \quad p_k = \frac{n_k}{\sum_{k=0}^{63} n_k}$$

and the n_k are the histogram counts for each of the 64 quantiser states.

For this image source

$$H_0(S_{i,j}) = 5.5246 \text{ bit/pel}$$

which is reasonably close to the 6 bit/pel of a uniform code (because p_k, the pmf of the pixel intensities in fig. 3.1b, is reasonably uniform across most of the states, $k = 0 \dots 63$). Throughout this discussion, we will use the abbreviation "pel" to stand for "pixel".

First-order Markov coder

Now we shall try to get a more efficient coder by assuming a 1st-order Markov model for this source.

To do this we look at the joint pmf of pairs of vertically adjacent pixels (we could have chosen horizontally adjacent pairs but vertical pairs work slightly better for this image). Fig. 3.1c shows the conditional probability matrix for the intensity of pel(i,j) conditioned on the intensity of pel$(i-1,j)$ (the pel above it) for all 256K pels in this image. We see the very strong concentration of the probabilities along or near the leading diagonal of the matrix, which illustrates how each pixel is very likely to have a value close to that of its neighbour above.

Based on a 1st-order Markov model that assumes that the pmf of each pixel depends only on the state of the pixel above it, the 1st-order conditional entropy of this image source is then

$$H_1(S_{i,j} \mid S_{i-1,j}) = \sum_{k=0}^{63} H_1(S_{i,j} \mid S_{i-1,j} = k)\ P(S_{i-1,j} = k) = 2.7930 \text{ bit/pel}$$

This is about half the zero-order entropy, and represents a significant saving in bit rate to code the image.

At the top of each column, we simply use a uniform 6-bit code to code the first pixel, which only increases the overall entropy to 2.7993 bit/pel.

Pixel-difference coder

In the case of lossless image coding, there is a much simpler option than the Markov coder. This arises because the conditional probabilities down each column of the matrix in fig. 3.1c are very similar to each other in shape and are all centred on the value of the previous pixel. This means that the pmf of the **difference** between pairs of vertically adjacent pixels will be virtually independent of the pixel values, and so can be efficiently coded with a zero-order model.

Fig. 3.2a shows the image of the pixel differences down the columns of the image (i.e. between pairs of vertically adjacent pixels), and fig. 3.2b shows their zero-order histogram. Using the formula, given at the start of this section, the zero-order entropy of the pixel differences is now found to be 2.8013 bit/pel, which is remarkably close to the value of 2.7930 bit/pel calculated above.

This coder is much simpler to implement as it requires only a single Huffman code, rather than the 64 codes needed for the first-order Markov coder.

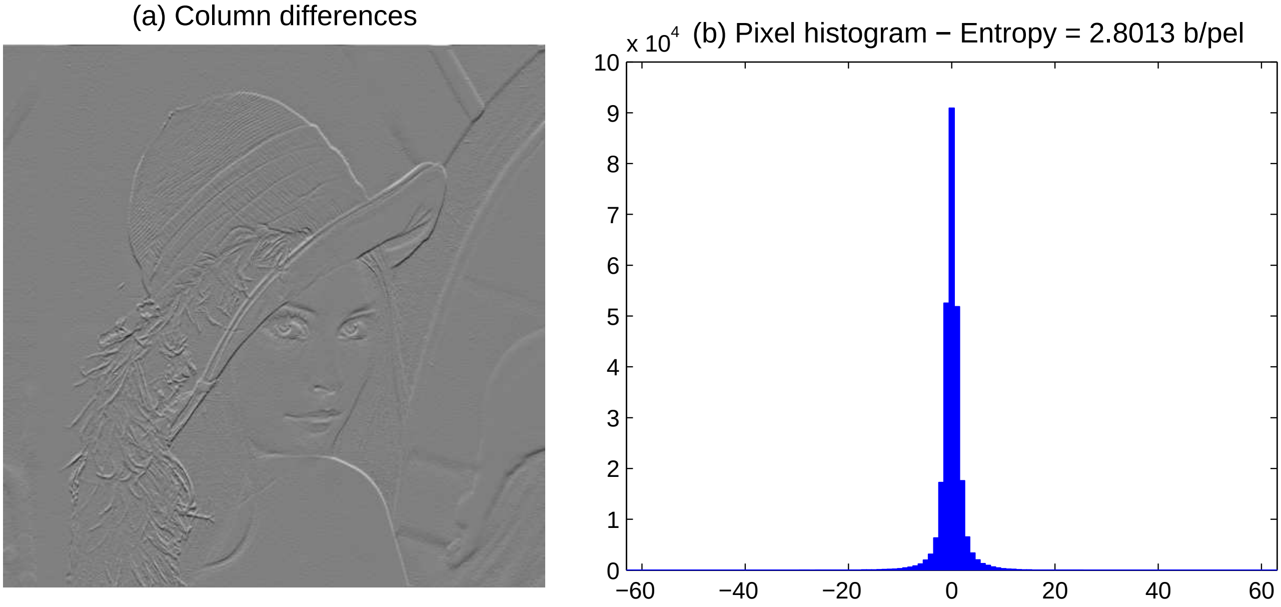


Fig. 3.2: Differences of pairs of vertically adjacent pixels, and the zero-order histogram of these pixel-differences.

But let us now try to do **even better** than this!

First we can try pixel differences along the rows of the image instead of down the columns. This is slightly worse with an entropy of 3.1694 bit/pel (probably because there are more near-vertical edges in this image than near-horizontal ones).

Next we can try to take row differences of the column differences (i.e. second-order differences in both directions), but this still gives a worse entropy than pure column differences, at 2.9681 bit/pel. This is because second-order differences start to become rather noisy.

We can make progress however, by observing that the more active (less grey) pixels in fig. 3.2a mainly tend to be clustered along edges and textures in the original image and there are large areas of low activity in between these regions.

As a result, in fig. 3.2a, **it is more likely that a pixel will be large if one of its neighbours is large** and much less likely if all its neighbours are small.

To show the advantages of this in a coding situation, we develop a first-order Markov coder in which the previous state is not a pixel value, but instead is a local context parameter C, based on the maximum magnitude of several neighbouring pixel-difference values, which is found to be a good measure of local image activity.

In the pixel-difference image (fig. 3.2a), we may only use neighbouring pixels that have been coded before the current pixel, since these are the only values available to the decoder at the time. Suppose the current pixel is c and we are using a raster scan order along successive rows of the image, then the 8 immediate neighbours of c may be labelled as follows

$$\begin{array}{ccc} b & b & b \\ b & c & a \\ a & a & a \end{array}$$

where b indicates pixels coded **before** c, and a indicates pixels coded **after** c.

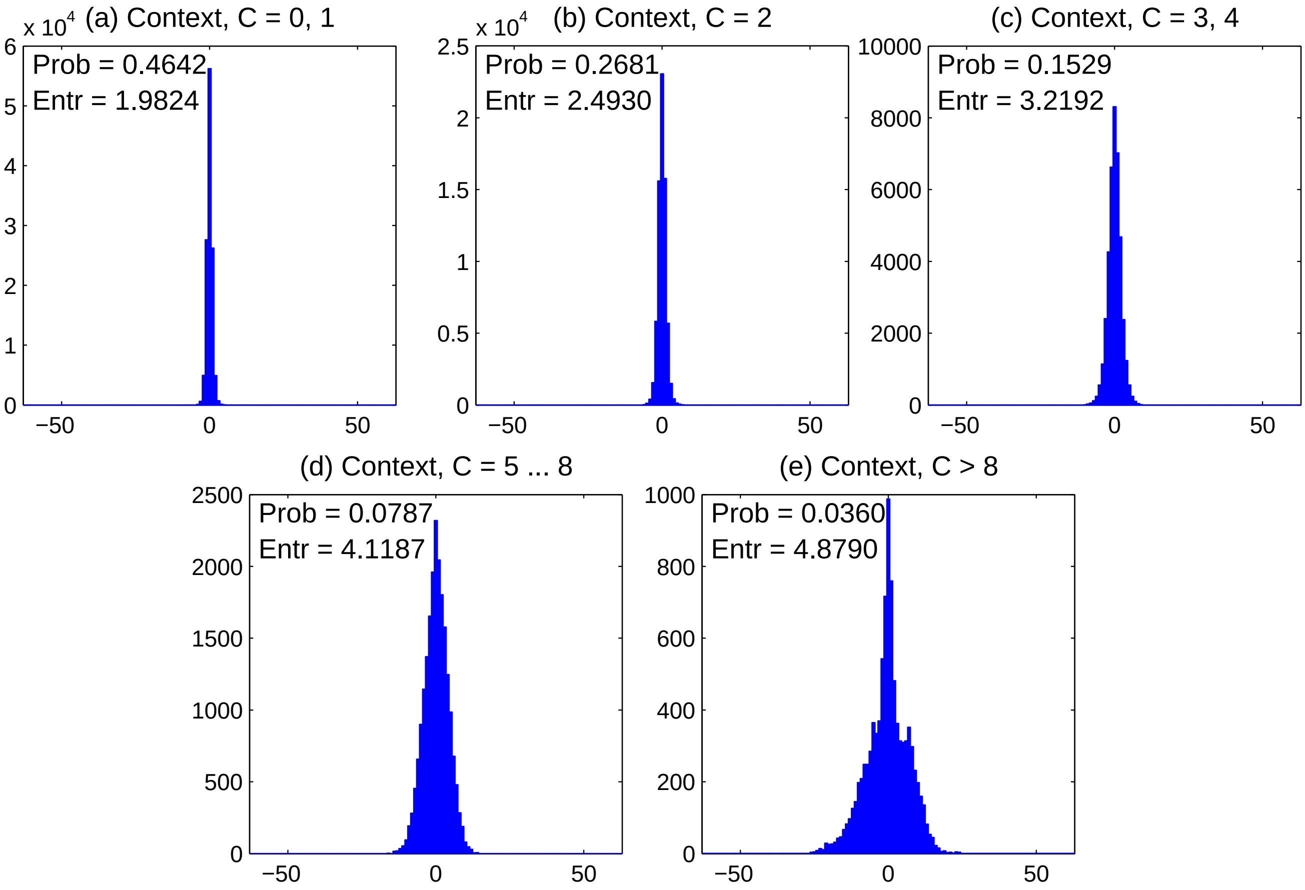


Fig. 3.3: Histograms of pixel differences, conditional on the context parameter C, based on the maximum absolute value of previously coded pixels in the immediate neighbourhood.

We define the context C for coding pixel c to be the maximum absolute value of the 4 pixels b.

Fig. 3.3 shows the histograms of pixels c, based on their contexts C. We see that when C is small (very little surrounding activity) the histogram is tightly concentrated on zero and produces a low entropy of 1.9824, while at larger values of C the entropy rises to 4.879 bit/pel. However these larger values of C are quite rare (as shown by the probability values in fig. 3.3), so the mean entropy now reduces to

$$\begin{aligned} H_1(S_{i,j}|C) &= 1.9824\ .\ 0.4642 + 2.493\ .\ 0.2681 + 3.2192\ .\ 0.1529 \\ &\quad + 4.1187\ .\ 0.0787 + 4.879\ .\ 0.036 \;=\; 2.5809 \text{ bit/pel} \end{aligned}$$

This context-based coder of pixel differences represents about an 8% reduction on our previous best entropy of 2.7993 bit/pel (from the first-order Markov coder of pixel values), and it is simpler because we only use 5 contexts (instead of 63) and hence only require 5 separate codes, one for each histogram in fig. 3.3.

It is difficult to get the entropy much below this value using lossless coding, but lossy techniques such as JPEG (and the newer JPEG 2000 and JPEG XR standards) can get the coded bit rate down to less than 1 bit/pel (this is covered in module 4F8).

4 Sources with Continuous Variables

4.1 Entropy of continuous sources

We can also define entropy for continuous variables with probability **density** functions.

Review of density functions

For a continuous variable X with density function $p_X(x)$ (abbreviated to $p(x)$), the probability that X has a value between a and b is:

$$P(a \leq x < b) = \int_a^b p(x)\, dx$$

The whole probability distribution must integrate to 1

$$\int_{-\infty}^{\infty} p(x)\, dx = 1$$

since X must be somewhere between $-\infty$ and ∞.

Entropy of a continuous variable

The entropy of a continuous variable X is defined to be

$$h(X) = -\int_{-\infty}^{\infty} p(x) \log_2 p(x)\, dx$$

This takes a similar form to the entropy of a discrete variable, but in some sense behaves a bit differently because the logarithms are now of probability densities rather than discrete probabilities. It is also often known as the **Differential Entropy** for reasons which will become clear below. Note the use of h, rather than H, for entropies of continuous variables.

Example entropies of continuous sources

1. Uniform distribution over (0,1)

$$p(x) = \begin{cases} 0 & x < 0 \\ 1 & 0 < x < 1 \\ 0 & x > 1 \end{cases}$$

$$h(X) = -\int_0^1 1 \times \log_2(1)\, dx = 0$$

2. Uniform distribution over (0,a)

$$p(x) = \begin{cases} 0 & x < 0 \\ \frac{1}{a} & 0 < x < a \\ 0 & x > a \end{cases}$$

$$h(X) = -\int_0^a \tfrac{1}{a} \times \log_2(\tfrac{1}{a})\, dx = \log_2(a)$$

Note: Scaling up the width of any continuous distribution by a increases its entropy by $\log_2(a)$.

3. Unit-variance normal distribution (Gaussian)

$$p(x) = \tfrac{1}{\sqrt{2\pi}}\, e^{-x^2/2}$$

Since this is an exponential function we use the ln form of the entropy expression:

$$\begin{aligned} \ln(2)\ .\ h(X) &= -\int p(x) \ln p(x)\, dx \;=\; \int \tfrac{1}{\sqrt{2\pi}}\, e^{-x^2/2} \left(\tfrac{x^2}{2} + \ln\sqrt{2\pi} \right) dx \\ &= \tfrac{1}{2} \int x^2\, \tfrac{1}{\sqrt{2\pi}}\, e^{-x^2/2}\, dx + \ln\sqrt{2\pi} \int \tfrac{1}{\sqrt{2\pi}}\, e^{-x^2/2}\, dx \\ (\text{var.} = 1) \quad &= \tfrac{1}{2}\ .\ 1 + \ln\sqrt{2\pi}\ .\ 1 \;=\; \ln\sqrt{2\pi e} \;=\; 1.4189 \\ \therefore\ h(X) &= \log_2\sqrt{2\pi e} \;=\; 1.4189/\ln(2) \;=\; 2.0471 \end{aligned}$$

For non-unit variance, we just add $\log_2(\sigma)$ to $h(X)$.

It would take an infinite number of bits to specify the value of a continuous variable to arbitrary precision so in some sense continuous variables should have infinite entropy.

This is for exactly the same reason that it would in general take an infinite number of bits to store the value of a real number, say π, to absolute precision.

Despite this, knowing that a variable has a value in the range $(0, 1)$ suggests that we know more than if it were in the range $(0, 2)$ (i.e. that it has less uncertainty). Our entropy calculations above reflect this intuition. So the entropy measure is useful for comparing continuous distributions.

In order to understand how the entropy of a continuous variable relates to the entropy of a discrete variable, we have to consider **quantising** the continuous variable.

4.2 Entropy change due to quantisation

If we take a continuous random variable X, we can create a discrete random variable Y by quantising X. A **uniform** quantisation of X can be performed by cutting the range of X up into a discrete number of equally sized sections.

For example we could quantise X using a **quantisation step size** of 1 and define a variable Y in terms of X as follows:

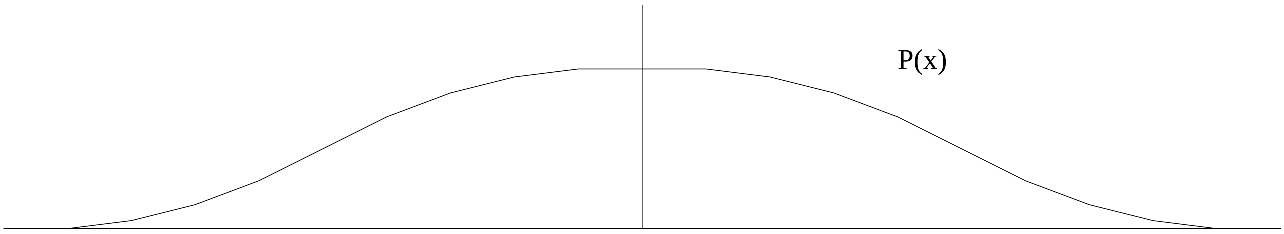
$$y = \begin{cases} \dots & \dots \\ -2 & (-2.5 \leq x < -1.5) \\ -1 & (-1.5 \leq x < -0.5) \\ 0 & (-0.5 \leq x < 0.5) \\ 1 & (0.5 \leq x < 1.5) \\ 2 & (1.5 \leq x < 2.5) \\ \dots & \dots \end{cases}$$

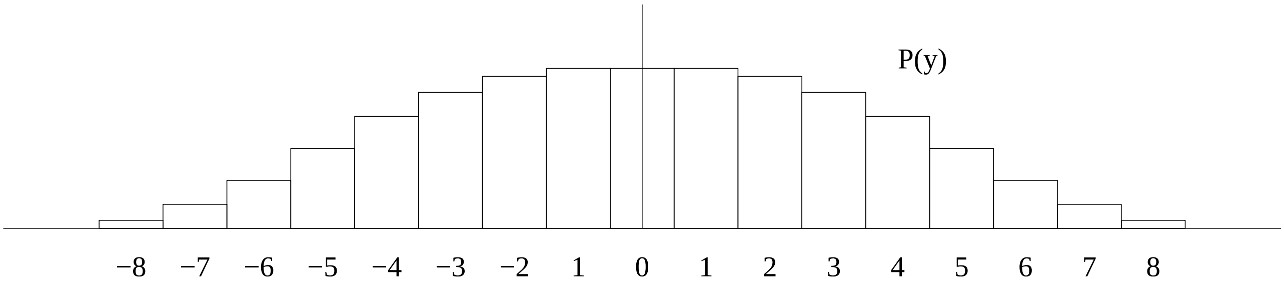
Or more simply: $y = \text{round}(x) = \lfloor x + 0.5 \rfloor$ (where $\lfloor . \rfloor$ is the integer 'floor' function.)

We can then calculate the probability **distribution** of Y:

$$P(y = n) = \int_{n-0.5}^{n+0.5} p(x)dx$$

This is equivalent to computing the histogram of a distribution





You would expect that the entropy of the discrete distribution (Y) should be related to the entropy of the continuous density function (X). The easiest way to see this is to consider quantising the distribution using a very small quantisation step (or bin width), δ. This means that the possible values for y are integer multiples of δ.

We can compute the entropy of the discrete distribution as:

$$H(Y) \;=\; -\sum_{i=-\infty}^{\infty} P(y = i\delta)\log_2(P(y = i\delta))$$

Now

$$P(y = i\delta) \;=\; \int_{(i-0.5)\delta}^{(i+0.5)\delta} p(x)\;dx \;\simeq\; \delta\,p(i\delta)$$

since we can approximate $p(x)$ by the value in the centre of the integral.

So:

$$\begin{aligned} H(Y) \;&=\; -\sum_{i=-\infty}^{\infty} \delta\,p(i\delta)\log_2(\delta\,p(i\delta)) \\ &=\; -\log_2(\delta) - \sum_{i=-\infty}^{\infty} \delta\,p(i\delta)\log_2(p(i\delta)) \end{aligned}$$

since

$$\sum_{i=-\infty}^{\infty} \delta\,p(i\delta) \to \int_{-\infty}^{\infty} p(x)\,dx = 1 \qquad \text{as} \quad \delta \to 0$$

Similarly the second term in $H(Y)$ tends to the following integral:

$$\sum_{i=-\infty}^{\infty} \delta\,p(i\delta)\log_2 p(i\delta) \to \int_{-\infty}^{\infty} p(x)\log_2 p(x)\,dx = -h(X) \qquad \text{as} \quad \delta \to 0$$

This means that

$$H(Y) \to h(X) - \log_2(\delta) \qquad \text{as} \quad \delta \to 0$$

Now $\log_2(\delta)$ is the entropy of a continuous uniform distribution of width δ which is the quantisation step (or bin) width. (See our earlier example with $a = \delta$.)

So $H(Y) + \log_2(\delta) \to h(X)$. This is equivalent to saying that the entropy of X is the entropy of Y plus the entropy of a uniform distribution over the quantisation bin.

If $p_X(x)$ were uniform over a unit range, $h(X)$ would be zero and so $H(Y)$ would be $-\log_2(\delta)$. Hence for an arbitrary pdf $p_X(x)$, $h(X)$ is the **difference** between the entropy of X, quantised with step δ, and the entropy of a uniform unit pdf, also quantised with δ (as $\delta \to 0$). This explains the name **differential entropy** for $h(X)$.

4.3 Mutual information of continuous variables

Another way of thinking about differential entropy is to consider the mutual information of X and Y.

In the case of quantisation, Y is completely determined by X, and so

$$I(Y;X) = H(Y)$$

On the other hand:

$$I(X;Y) = h(X) - h(X|Y)$$

$h(X)$ is the (differential) entropy of the continuous random variable X and $h(X|Y)$ is the entropy of X given that you know which quantisation bin it falls into. Since the distribution is assumed to be uniform over the quantisation bin for small δ, this is the quantity $\log_2(\delta)$ that we saw on the previous page.

We can consider the mutual information of two continuous variables in the same way. Because mutual information is the difference of two entropies, it can be used with differential entropies of continuous variables just as easily as with entropies of discrete variables, since the $\log_2(\delta)$ terms of the former will cancel out. This property is used in the following example.

4.4 Example: the Gaussian Communications Channel

We return to the topic of communication channels, but this time consider a classic one from Shannon's original work – **the Gaussian channel**.

The Gaussian channel operates in discrete time (like the binary channel), but it accepts continuous (pdf) input variables drawn from random process X and produces continuous output variables Y, such that the samples of Y are related to those of X by

$$y_i = x_i + n_i$$

where the n_i are samples of a Gaussian noise process N whose pdf is a normal distribution with variance σ_N^2. N is assumed to be uncorrelated with X.

For this example we shall assume that the pdf of X is also a normal distribution (which turns out to be optimal, see next page) and that it has variance σ_X^2, which typically represents the upper limit on the mean energy per symbol for proper operation of the channel.

Since X and N are uncorrelated and normal, the pdf of Y will be a normal distribution with variance given by

$$\sigma_Y^2 = \sigma_X^2 + \sigma_N^2$$

Using arguments similar to those we used for the case of the binary symmetric channel, we may calculate the information capacity of the Gaussian channel, in bits per transmitted symbol, to be the mutual information between X and Y. However this time we use differential entropies, because X and Y both use continuous variables.

$$I(Y;X) = h(Y) - h(Y|X) = h(Y) - h(X+N|X) = h(Y) - h(N)$$

Using the result for the entropy of normal distributions from section 4.1, we get

$$\begin{aligned} h(N) &= \log_2 \sqrt{2\pi e} + \log_2(\sigma_N) \\ h(Y) &= \log_2 \sqrt{2\pi e} + \log_2(\sigma_Y) \end{aligned}$$

Hence the mutual information of the channel is

$$\begin{aligned} I(Y;X) &= h(Y) - h(N) \; = \; \log_2(\sigma_Y) - \log_2(\sigma_N) \\ &= \log_2 \frac{\sigma_Y}{\sigma_N} \; = \; \tfrac{1}{2} \log_2 \frac{\sigma_X^2 + \sigma_N^2}{\sigma_N^2} \\ &= \tfrac{1}{2} \log_2 \left(1 + \frac{\sigma_X^2}{\sigma_N^2} \right) \end{aligned}$$

As defined in section 3.5, the capacity of the channel is the maximum of $I(Y;X)$ over all choices of input distribution $p(X)$. It can be shown that for mean-power-limited continuous signals, the normal distribution (our assumption above) is optimal and gives the maximum $I(Y;X)$ for any given ratio of signal to noise power. Hence the capacity of the Gaussian channel is

$$\begin{aligned} C_G &= \tfrac{1}{2} \log_2 \left(1 + \frac{\sigma_X^2}{\sigma_N^2} \right) \quad \text{bits per transmitted symbol} \\ &= B \log_2 \left(1 + \frac{\sigma_X^2}{\sigma_N^2} \right) \quad \text{bits per second} \end{aligned}$$

where B is the channel bandwidth in Hz, because, in theory, 2B symbols per second may be transmitted through a B-Hz channel using ideal sinc-shaped transmit pulses.

The mean signal power P_X will then be $2B\sigma_X^2$, and the mean noise power P_N in a B-Hz bandwidth will be $2B\sigma_N^2$. This gives the standard result (in the E&I data book, page 27):

$$C_G = B \log_2 \left(1 + \frac{P_X}{P_N} \right) \quad \text{bits per second}$$

Shannon, in his 1948 paper, was able to show that error-free communications could be achieved in theory up to just below this capacity limit by using transmit waveforms that were essentially many long segments of bandlimited Gaussian noise, and by using a very-high-complexity receiver which correlated all possible transmit waveforms with the received waveform in order to determine the closest matching transmitted signal.

However it has only been possible to approach this level of performance in practice (to within 0.04 dB with very long codes!) during the last 10 years or so, following the development of turbo-codes and low-density parity check (LDPC) codes, sophisticated iterative decoding algorithms, and high-performance signal processing chips to implement them. These techniques are discussed in detail in module 4F5 ‘Advanced Wireless Communications’, which continues on from this year’s 3F1 and 3F4 modules.