Information Theory

Peter Fenwick
COMPSCI 314
January 29, 2004

***Note** that this document is intended as a summary of some of the basic results of information theory, rather than as a formal text. Most results are therefore just quoted, rather than formally proved.*

1 Introduction

Much computing deals with the representation, transmission or storage of information. A frequent question is "How much space is needed to store this information?", or, equivalently, "How long does it take to transmit this information?" Matters such as this are dealt with by *Information Theory* and *Coding Theory*, introduced by Claude Shannon in 1948 in his papers *A Mathematical Theory of Communication.* [1]

To handle these problems we consider a general information transmission system, as shown in Figure 1; remember that storage (movement in time) is equivalent to transmission (movement in space).

- The **source** generates a stream of *information*, where information may be loosely defined as anything that changes the knowledge or understanding of something. (A formal definition will be given later.) The information is usually a stream of *symbols*, chosen from some *source coding alphabet*. Thus we may have *letters* from the standard alphabet {*a, b, c, d, . . . ,* z}, *bits* from the binary alphabet {0, 1}, or *digits* from the numerals {0, 1, 2, . . . , 9}.

  The source is described by both its alphabet and the probabilities of its emitted symbols, such as

  $$\{P(a), P(b), P(c), P(d), \ldots\}$$

  The information associated with receiving a symbol is related to the *surprise*; an expected symbol conveys little information, while an unexpected one may convey a lot of information.

- The **coder** converts the stream of source symbols into a stream of symbols from a *code alphabet*, which is usually but not necessarily binary. Thus an ASCII coder converts letters etc into binary *codewords*, such as $A \rightarrow 100\,0001$ or $9 \rightarrow 011\,1001$.

  A good coder is matched to both the source alphabet (and its symbol probabilities) and to the channel. Some important cases are –

  - An *error correcting* coder adds redundant information so that the decoder can recover correct data, even if the channel adds noise and corrupts the transmitted data.
  - A *lossless compressor* uses redundancy in the source to reduce the volume of encoded or transmitted data, while still allowing the decoder to recover the original source data exactly, bit by bit. Lossless compressors are often used for program distribution, such as LZW, BZIP and GZIP.
  - A *lossy compressor* reduces that data volume, but only so that a human observer sees the decoder output as acceptably close to the original. These compressors are used for pictures or sound, such as MP3, JPEG and MPEG, generally where the recovered data is interpreted by humans.

- The **channel** carries the encoded information shifted in either space (data transmission) or time (data recording & playback). The channel will usually have a limited bandwidth or bit rate (its *capacity*) and may introduce noise.

- The **decoder** must simply reverse the coder, delivering data acceptably close to the source output, despite any noise or limitations of the channel. ("Acceptably close" is a nice vague term!)

- The **sink** is expected to receive the decoded information and use it in some sensible way.

2 Information and Sources

Consider the information content (I) of a message of n symbols, each describing an event (E) which occurs with a probability of $P(E)$. Then

[1] The basic references on Information Theory are – R.V. Hartley (*Bell Sys. Tech. Journ.*, Vol 7, p 535 1928), and C. Shannon (*Bell Sys. Tech. Journ.*, Vol 27, pp 379 & 623, 1948).

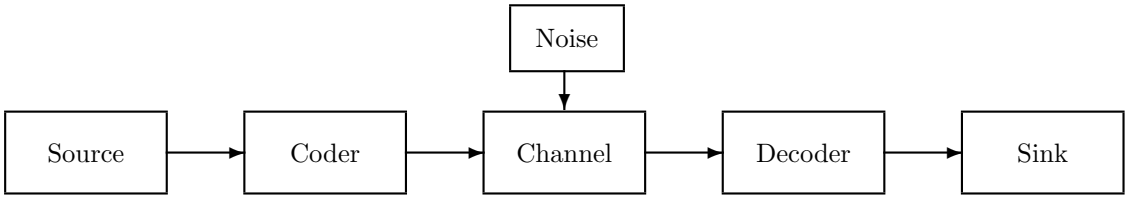


Figure 1: Information transmission system

- I increases as n increases
- as $P(E) \to 0$, $I \to \infty$ (unexpected outcome)
- if $P(E) = 1$ then $I = 0$ (outcome certain)
- if two messages of n_1 and n_2 symbols are concatenated, the resulting information is added, i.e. $I = I_1 + I_2$

Accordingly, we say that being told that an event has occurred with a probability $P(E)$ gives us the information

$$I(E) = \log \frac{1}{P(E)} = -\log P(E)$$

In most cases the logarithms are to the base 2, and the information measure is in "binary *units*" or "bits", which are not to be confused with a "binary *digit*" or bit[2]. A binary *digit* may convey more or less than one binary *unit* of information; on average it will usually convey less, but never more than one binary unit.

2.1 Zero Memory Information Source

This is a convenient model of a simple mechanism which generates information. The successive symbols are chosen from some fixed source alphabet S according to some fixed probabilities.

$$S = \{s_1, s_2, s_3, \ldots, s_n\}$$

Note that S may denote either the source or the alphabet. For the simplest case assume that the symbols are statistically independent – this gives the *zero-memory source*. This source is completely described by its symbol probabilities $P(s_1)$, $P(s_2)$, $P(s_3)$, ..., $P(s_n)$.
For each symbol s_i we receive information

$$I(s_i) = \log \frac{1}{P(s_i)}$$

[2]The two uses of the term bit are both firmly established, despite the confusion that results. The conflict is puzzling, because both terms were devised at about the same time (1946), by workers in the same group at Bell Laboratories.

with probability $P(s_i)$, giving information for that symbol of

$$I(s_i) = P_i \log \frac{1}{P(s_i)}$$

The average information received per symbol is known as the *Entropy* of the source and is denoted by

$$H(S) = \sum_i P(s_i) \log \frac{1}{P(s_i)}$$

For a source of q symbols $H(S) \leq \log q$, with equality if and only if all symbols are equiprobable.

For the important case of a zero-memory binary source $S = \{0, 1\}$, with $P(1) = \omega$ (omega) and $P(0) = 1 - \omega = \overline{\omega}$, the entropy is zero for $\omega = 0$ and $\omega = 1$, and is a maximum of $H = 1.0$ at $\omega = \overline{\omega} = 0.5$. The Entropy Function $H = P \log_2 \frac{1}{P}$ is an important tabulated function in Information Theory and is given later in Table 1.

The information theory entropy H is related to the thermodynamic entropy S; both are functions of the logarithmic probabilities. But the true relationship is difficult and confusing and has led to much spurious and misleading speculation. Beware!

2.2 Extensions of Sources

An *extension* of a source is that new source which results when the emitted symbols are considered in groups. Thus for the binary source {0,1}, the first three extensions have alphabets {0, 1}, {00, 01, 10, 11} and {000, 001, 010, 011, 100, 101, 110, 111}. The nth extension of a source S of q symbols is denoted by S^n. S^n has an alphabet of q^n symbols, where each is a sequence of n symbols from S. It is both obvious and easily proven that $H(S^n) = nH(S)$. In other words the entropy of the nth extension of a source is n times the entropy of the source itself.

2.3 Markov Sources

The symbols of many practical sources are not independent, but have probabilities which depend on

the preceding symbol(s). For an mth order Markov source, the symbol s_i occurs with a probability which is a function of the preceding m symbols, according to the conditional probabilities $P(s_i|s_{j_1}, s_{j_2}, \ldots, s_{j_m})$. Markov sources are of little importance in simple Information Theory, but are crucial to more advanced applications, and data compression.

2.4 Structure of Language

A good example of a Markov source is ordinary text. Consider an alphabet of 27 symbols (26 letters and space).

1. **Zero memory source** with equiprobable symbols.
   Then $H(S) = \log_2(27) = 4.75$ bits/letter.
2. **Zero memory source** with probabilities
   $H(S) = 4.05$ bits/letter
3. **First-order Markov source**, from digraph frequencies $H(S) = 3.32$ bits/letter.
   It is easy to generate text according to first-order Markov statistics. Take a book and choose a letter at random; this becomes the first emitted symbol. Then search through the text for the next occurrence of that letter and choose the following letter as the next symbol. Repeat the process of searching for the next occurrence of the last-emitted symbol and taking the letter which immediately follows it.
4. **Second-order Markov source**,
   $H(S) = 3.1$ bits/letter

In fact the entropy of English is found to be from 0.6 to 1.3 bits per letter. (A good text or lossless compressor can be expected to get about 2.0 bits/letter, including punctuation and upper/lower case.)

Again, a reasonably simple experiment is sufficient to estimate the entropy of English. It requires two people, the first with a book which is not known to the second person. The second person tries to guess what is next – it may be a letter, a word, or even a phrase. Each question and "Yes/No" answer corresponds to one bit of information, and each letter of the text is a source symbol. Guessing may continue until a correct answer, or may stop after say 5–6 attempts for a symbol.

3 Noiseless Coding

A code is a mapping of symbols from a *source alphabet* $S\{s_1, s_2, s_3, \ldots s_n\}$ into a sequence of symbols from the *code alphabet* $X\{x_1, x_2, x_3, \ldots x_v\}$. In general the two alphabets are different. The term "code" may also refer to the set of "codewords", as defined below.

3.1 Classification of Codes

Block code each symbol of S maps into a fixed sequence of codewords of symbols chosen from X.

Non-singular code a block code with all code words distinct.

Uniquely decodeable code the nth extension of the code is non-singular for all finite n

Instantaneous code no complete codeword is a prefix of another codeword from that code.

Comma code a code in which all codewords (except perhaps for the longest) end with a special symbol pattern

3.2 Kraft Inequality

A necessary and sufficient for the existence of an instantaneous code with word lengths $\ell_1, \ell_2, \ldots \ell_q$, in an alphabet of r symbols is that

$$\sum_{i=1}^{q} r^{-\ell_i} \leq 1 \quad \textbf{Kraft Inequality}$$

This shows only whether an instantaneous code is possible for the given codeword lengths, but it may help in the construction of a code. For example, to construct a binary code for a decimal source which emits mostly 0s and 1s.

Firstly, code 0 and 1 (the two most probable symbols) as $0 \rightarrow 0$ and $1 \rightarrow 10$. If the other symbols are of length ℓ_i, then by the Kraft inequality

$$\sum_{i=1}^{q} 2^{-\ell_i} \leq 1$$

If all of the other symbols are assumed to be of the same length m, then, by the Kraft inequality,

$$2^{-1} + 2^{-2} + 8 \times 2^{-m} \leq 1$$

whence $m = 5$. The 8 less-probable symbols are all encoded as a '11' prefix ahead of a 3-bit binary number and the complete code is then

0	→	0
1	→	10
2	→	11000
3	→	11001
4	→	11010
5	→	11011
6	→	11100
7	→	11101
8	→	11110
9	→	11111

3.3 Average Code Length

If we encode a set of q symbols with probability P_i into codewords of length ℓ_i, the average length, L, of the code is $L = \sum P_i \ell_i$. It is easily shown that if we encode in an alphabet of r symbols and measure the entropy $H_r(S)$ in base-r units that $H_r(S) \leq L$. Thus the average code length cannot be less than the entropy of the source. We get equality if

$$P_i = r^{-\ell_i}, \text{ or } \ell_i = \log_r \frac{1}{P_i} \text{ for all } i$$

This is an important result, as it relates a relatively abstract Information Theory measure, the entropy, to a much more "practical" value, the average length of the code.

Furthermore, using an alphabet of r symbols, if we select the length of the ith symbol as

$$\log_r \frac{1}{P_i} \leq \ell_i < \log_r \frac{1}{P_i} + 1.$$

Multiply by P_i and sum over all i to get

$$H_r(S) \leq L < H_r(S) + 1.$$

Now apply to the n-th extension of S, considered as a source to obtain

$$H_r(S^n) \leq L_n < H_r(S^n) + 1$$

Thus

$$nH_r(S) \leq L_n < nH_r(S) + 1$$

or

$$H_r(S) \leq \frac{L_n}{n} < H_r(S) + \frac{1}{n}.$$

Now, L_n/n is the average number of code symbols per symbol of S, giving

$$\lim_{n\to\infty} \frac{L_n}{n} = H_r(S) \quad \textbf{Shannon's First Theorem}$$

Shannon's First Theorem shows that we can approach the ideal by coding extensions of the source.

3.4 Compact, or noiseless, coding

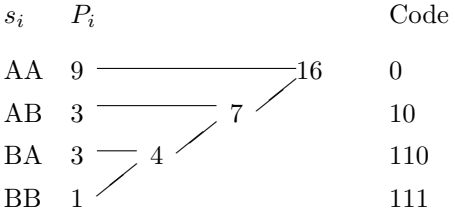
3.4.1 Huffman code

The Huffman code is the most important of the compact codes. To construct a Huffman code, the symbols are first ranked in order of probability. Then, successively combine the two symbols of lowest probability to form a new composite symbol, eventually building a tree where each node has the probability of all those nodes beneath it. The branching ratio of each node is equal to the number of symbols in the code alphabet. The code for each symbol is derived by tracing the path from the root to its leaf, and noting the direction taken at each node.

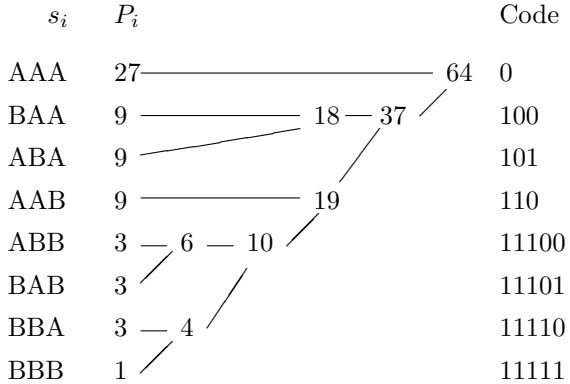
Considering the source with probabilities $\frac{1}{4}$ and $\frac{3}{4}$ (and noting that it is often useful to scale to integer probabilities), the code for the first extension is clearly (remembering that 0s and 1s are interchangeable)

s_i	P_i	code
A	3	0
B	1	1

The codes for the second and third source extensions are then –



Huffman Code for second extension



Huffman Code for third extension

The code efficiency η (eta) may be defined as the ratio $H(S)/L$. For this source the values are –

$H(S^1) = 0.8113 \quad L_1 = 1.0000 \quad \eta = 0.8113$
$H(S^2) = 1.6226 \quad L_2 = 1.6875 \quad \eta = 0.9615$
$H(S^3) = 2.4338 \quad L_3 = 2.4688 \quad \eta = 0.9859$
$H(S^4) = 3.2451 \quad L_4 = 3.2734 \quad \eta = 0.9913$
$H(S^5) = 4.0564 \quad L_5 = 4.0889 \quad \eta = 0.9921$

3.4.2 Shannon-Fano code

Another, older, code is the Shannon-Fano code. Once again, rank the symbols in decreasing probability. But now divide the symbols into two groups of approximately equal probability: allocate the top group a digit 1 and the lower group a digit 0. Continue further dividing each group until all groups are of size 1 and then collect the code digits. In general the "cut" will be within a symbol; the preferred group for that symbol may be obvious, but in case of doubt include the symbol in the group of smaller probability to obtain the best balance in the cut[3]. The Shannon-Fano code is generally inferior to the Huffman code, but may be more amenable to theoretical analysis.

As an example the Shannon-Fano code for the 3rd extension above is

s_i	P_i	Code
AAA	27	00
BAA	9	01
ABA	9	100
AAB	9	101
ABB	3	1100
BAB	3	1101
BBA	3	1110
BBB	1	1111

Shannon-Fano Code for third extension

The average length of this code is $L = 2.5938$, and its efficiency is $\eta = 0.9383$; the Shannon-Fano code is therefore less efficient than the Huffman code. In this case the Shannon-Fano codes for the first two extensions of this source are identical to the Huffman codes, but the code for the third extension (as above) is inferior to that for the second extension.

(At first sight it *looks* as though the Shannon-Fano code should be more efficient, because its longest codewords are all shorter than those of the Huffman code. But the *most frequent* codeword is shorter for the Huffman code, and this is enough to tip the balance.)

Especially with Huffman codes, the efficiency usually tends to 1 quite quickly as extensions are coded[4].

[3]This rule is *not* well defined, which is major problem with the Shannon-Fano code.

4 Information Channels

The *channel*, the path from the source to the receiver, will usually perturb the signal, i.e. is *noisy*. For simplicity we usually deal with *binary* channels, which accept only the binary digits {0, 1}. We will consider two types.

- A *Binary Erasure Channel* accepts binary symbols {0,1} and delivers the symbols {0, 1, X}, where any error becomes the "X", or don't-know symbol. An example is a channel which is subject to fading, or a channel where each symbol has a parity check.
- A *Binary Symmetric Channel* (BSC) delivers only 0s and 1s, but with a finite, and symmetrical, probability of corruption or error. The immediate concern here is not with how to transmit the information, but with the theoretical limits on the possible data rates.

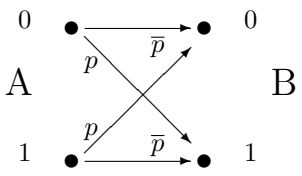
A channel is defined by three things, its input alphabet A $\{a_1, a_2, \ldots, a_r\}$, its output alphabet B $\{b_1, b_2, \ldots, b_s\}$ and lastly the conditional probabilities $P(b_j|a_i)$ of receiving the symbol b_j when a_i is sent.

$$P(b_j) = \sum_i P(b_j|a_i)P(a_i)$$

A *Binary Symmetric Channel* with error probability p is defined by the *channel matrix* below, where $\overline{p} = (1-p)$ and $p = P(0|1) = P(1|0)$ and $\overline{p} = P(0|0) = P(1|1)$.

$$\begin{pmatrix} \overline{p} & p \\ p & \overline{p} \end{pmatrix}$$

An alternative definition of a Binary Symmetric Channel uses the transition diagram –



Transition diagram for BSC

[4]Because the codeword length must be a whole number of bits, we sometimes find that efficiency *decreases* slightly for some higher extensions, in apparent violation of the rule.

4.1 Channel Capacity

Our main interest in channels is, firstly, that they exist and are described as shown above and, secondly, that it is possible to calculate a theoretical limit to the information transmission capacity of a noisy channel. This limit is known as the *Channel Capacity*.

A very important result, which will not be proved, is the "Fundamental Theorem of Information Theory" –

> *It is possible to transmit information through a noisy channel with an arbitrarily small probability of error if, and only if, the rate of information is less than channel capacity.*

A very simple derivation of the channel capacity is now given. Assume that we have an information source A sending over a noisy channel and that the system includes a "compensator" mechanism which indicates when the channel output is in error: the error indication may be regarded as another information source E. The channel can deliver output at a specified rate and corresponds to an information source B. (Note that it is the *output* bit rate B that is defined by the physical channel, rather than the input rate.) Some of the information from B is used for error indication (source E), and the remainder is available for information from A.

$$\begin{aligned} H(B) &= H(A) + H(E) \\ P(b_j) &= \sum_i P(a_i)P(b_j|a_i) \quad \text{can get } H(B) \\ P(e_1) &= \sum_{i \neq j} P(a_i)P(b_j|a_i) \quad \text{output in error} \\ P(e_0) &= \sum_{i=j} P(a_i)P(b_j|a_i) \quad \text{output correct} \end{aligned}$$

From these formulæ we can calculate $H(B)$ and $H(E)$, and therefore $H(A)$, the maximum amount of information which the channel can accept. This technique gives the information which can be transmitted for given source statistics; the maximum information is defined as the *channel capacity* and occurs, for a BSC, when the source symbols are equiprobable. For a non-symmetric channel, transmission at the channel capacity occurs at some other source symbol probabilities.

For example, a binary source $P(0) = 0.75$, $P(1) = 0.25$ transmits over a channel with error probability 0.1 at a maximum rate of 100 bit/s (binary *digits*, not binary *units*).

Firstly calculate the probabilities of the *received* symbols, $P(B)$ to give the received information rate.

$$\begin{aligned} P(B=0) &= 0.75 \times 0.9 + 0.25 \times 0.1 = 0.7 \\ P(B=1) &= 0.75 \times 0.1 + 0.25 \times 0.9 = 0.3 \end{aligned}$$

Thus $H(B) = 0.8813$, giving 88.13 bit/s *information* from the channel.

Now calculate the information rate needed to correct the errors. $P(E=0) = 0.9$ and $P(E=1) = 0.1$ from the channel definition, whence $H(E) = 0.4690$. The total information delivered is 88.13 bit/s, but of this $0.4690 \times 100 = 46.9$ bit/s is taken by error correction. This leaves 41.23 bit/s as the available Channel Capacity.

For equiprobable source symbols $H(B) = 1.0$ and $H(E) = 0.4690$; the channel capacity is only 53% that of a quiet channel. With an error rate of 0.01, the capacity is 92% (8% loss), and an error rate of 0.001 gives a capacity of 98.86% (1.14% loss).

5 Continuous Information Systems

In many information systems the signals can assume a continuous spectrum of values rather than just a few (e.g. 2 in a binary system). These systems are characterised by the bandwidth W, the signal power P, and the noise power N.

The *Sampling Theorem* states that a signal occupying a bandwidth W can be completely specified by $2W$ independent samples per unit time. Thus to digitise a standard telephone channel with a bandwidth of 3100Hz, we need at least 6200 samples per second. In fact the normal telephone sample rate is 8000 samples per second, which is somewhat above the "Nyquist" limit.

As a very crude determination of the capacity of a noisy information channel, note that the noise limits the accuracy with which we can measure the signal power. If the signal power is P and the noise power is N, we can distinguish $1 + \frac{P}{N}$ signal levels. If these are encoded, a single "state" of the signal can convey $\log(1 + \frac{P}{N})$ bits of information. As we can establish W states per second the total information capacity, C, is

$$C = W \log(1 + \frac{P}{N}) \text{ bits/second.}$$

(*This answer is correct, but for the wrong reason!*

While we can establish $2W$ *states per second, the nature of noise actually means that we can distinguish fewer than* $(1 + \frac{P}{N})$ *states. The two errors cancel.*)

This result is probably the one most important consequence of Shannon's Information theory as it shows that it is possible to "trade off" noise against bandwidth, or vice versa.)

The rigorous proof involves noting that a signal of bandwidth W lasting for a time T can be characterised by $2WT$ numbers ("samples"). A particular signal can be considered as an ordered set of $2WT$ numbers which defines a point in a Cartesian space of $2WT$ dimensions. It can be shown that the distance from the origin to any point is related to the signal energy E by

$$\begin{aligned} d^2 &= 2WE \\ &= 2WTP \qquad P \text{ is average signal power} \end{aligned}$$

Noise forces each signal point to be surrounded by an area of uncertainty which forms an n-sphere of radius $\sqrt{2WTN}$. The received signals have an average power of $(P + N)$ and lie on an n-sphere of radius $\sqrt{2WT(P + N)}$. A signal point is uniquely decodeable if and only if it is enclosed by only 1 noise n-sphere centred on a valid signal point. (Compare with Hamming distances in the discrete case.)

The volume of an n-sphere of radius r in n dimensions is proportional to r^n, and has almost all of its volume near the surface[5]. The signal n-spheres each have a volume proportional to $\sqrt{(P + N)^{2WT}}$ and the noise n-spheres $\sqrt{N^{2WT}}$. The number of identifiable messages is the number of noise n-spheres which will fit into the signal n-sphere, or $M = \sqrt{(\frac{P+N}{N})^{2WT}}$; the channel capacity is $C = \log_2 \frac{M}{T}$ bits/second.

$$\text{Thus} \qquad C \leq W \log_2(1 + \tfrac{P}{N})$$

$$\text{and in fact} \quad C = W \log_2(1 + \tfrac{P}{N})$$

It can be shown that the function which transmits the maximum entropy/unit time is one which approximates a noise source for any input message, i.e. the frequency spectrum is flat and independent of the information and there is little correlation between adjacent sample representations.

In general it is found that if the noise has some identifiable characteristic, we can transmit the maximum information by using a form of modulation which mimics the noise, but in some known way.

[5]Consider dV/dr, even for a 3-sphere. If the skin of an orange has a thickness of 20% of the radius, the radius of the flesh is 80% and its volume is $0.8^3 = 0.512$). The flesh (80% of radius) then contains only 50% of the volume, and there seems to be nothing left after the orange has been peeled.

The practical result of this theory in practical telecommunications is that a telephone channel with bandwidth of 3100Hz can transmit reliable data at considerably more than 3100 bits/second, provided that we have a suitably good signal/noise ratio, and that we use an efficient modulation technique.

6 Text Compression

NOTE: only LZW as in the textbook is examinable for COMPSCI 314 S1 C in 2004. None of the other material here will be examined.

Huffman and Shano-Fano Codes are clearly suitable for compressing text. However they must encode with high-order extensions for good performance and need *a priori* symbol probabilities for the extended codes. Such information is not easy to obtain. While some workers have used adaptive Huffman coding, there are now much better text compressors. Most compressors learn from the incoming data, and automatically adapt with no prior assumptions about its structure and statistics.

6.1 LZ-77 Ziv-Lempel Compressors

These use a "window" on the last few thousand bytes of text and search that window for strings matching the incoming text. Each match is replaced by a pointer to the previous occurrence of that string, ie a [displacement, length] pair. Characters which do not match are encoded as literals. The compressed output is thus a mixture of string pointers and literals.

Writing literals as themselves, and enclosing phrase pointers in braces as {*displ*, *length*}, the text

`the␣car␣on␣the␣left␣hit␣the␣car␣i␣left`

could be encoded as –

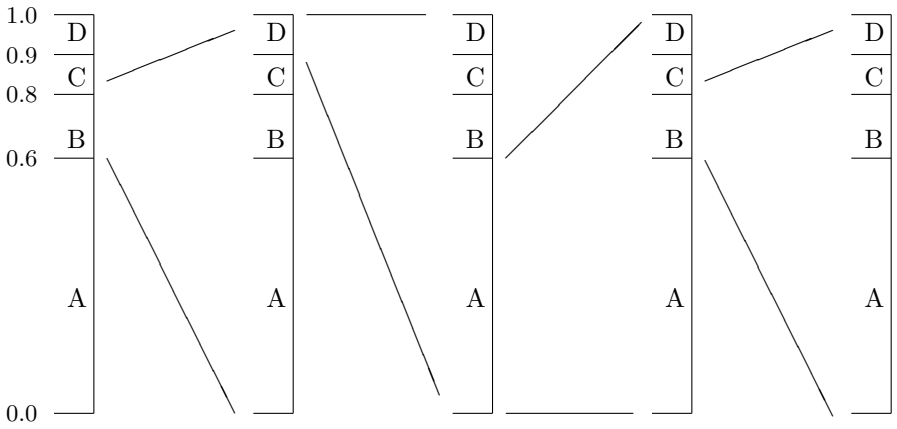
`the␣car␣on␣{11,4}left␣hit␣{24,8}i{19,5}`

Variations of LZ-77 compression lie mainly in the coding methods. One example of a (non-standard) LZ-77 encoding has four entities in the code –

0	single literal byte, copied without change	
1	8-bit pointer, say [6, 2]	(6 disp, 2 len)
2	16-bit pointer, say [11,5]	(11 disp, 5 len)
3	24-bit pointer, say [15, 9]	(15 disp, 9 len)

Each collection of 4 entities is preceded by a single "flag" byte which has 4 2-bit flags to indicate the nature of the following entities.

Most production data file compressors (such as `GZIP` and `WinZip`, but *not* `BZIP` and `BZIP2`) use versions of the LZ-77 algorithm.



Symbol	B	D	A	B	...
High Limit	0.80	0.80	0.792	0.7896	
Low Limit	0.60	0.78	0.780	0.7872	

Figure 2: Example of Arithmetic Coding

6.2 LZ-78 and LZW compressors

Another important family of compression algorithms (LZ-78) build a dictionary of known phrases and emit dictionary indices rather than string pointers. The LZW variant of the LZ-78 algorithm is fully described in the text and lecture notes.

LZ-78 compressors generally give rather poorer compression than their LZ-77 counterparts, but are rather more amenable to theoretical analysis. The Unix `compress` utility is the main example of an LZ78 or LZW compressor.

6.3 Statistical Compressors

More powerful compression techniques analyze the statistics of the input text and determine inter-symbol dependencies. These are generally the most effective compressors, but often use large amounts of memory and processing. Most statistical compressors build tables or *models* of symbols that have been seen in previous *contexts*. For example, given the context '␣th' (where ␣ represents a space), it is most unlikely that the next symbol will be anything other than $\{a, e, i, o, u, r, y\}$. But we always need a special *escape* symbol (say $\emptyset$) to be emitted if an unexpected symbol is found; the symbols in this context become $\{a, e, i, o, u, r, y, \emptyset\}$, which can at worst be all represented in 3 bits.

Usually the frequent symbols are given shorter codes, according to their statistics (as in Huffman codes). But in general, a statistical compressor will use the context of a symbol to reduce the number of possible symbols, predict their frequencies, and allow shorter codes to be used. The context *order* corresponds to the order of a Markov information source, as considered earlier.

A good example is arithmetic compression, which involves transforming the input text into a single number of very high precision. It can be shown that arithmetic compressors are optimal and encode arbitrarily close to the source entropy. Arithmetic compressors are often the final stage of a more complex compressor, especially ones that analyze the contexts of symbols and predict probable symbols within each context.

Assume an alphabet $\{A, B, C, D\}$ with relative symbol probabilities $\{0.6, 0.2, 0.1, 0.1\}$, as in Fig. 2. We encode the symbols as numbers in the ranges A $(0 - 0.6]$; B $(0.6 - 0.8]$; C $(0.8 - 0.9]$; D $(0.9 - 1.0]$, where the upper limit is in all cases excluded from the range.

If the first symbol is a B, we know that the code value Z must be such that $0.6 \leq Z < 0.8$ and encode the second symbol within this range. If the second symbol is D, the range becomes $0.78 \leq Z < 0.8$; for the third symbol A, the range becomes $0.78 \leq Z < 0.792$, and so on.

As the encoding proceeds, high order digits become constant and independent of any later symbols. These digits are then emitted as the compressed data, and the range is magnified as far as possible while

keeping $0 \leq Z < 1$. The ranges may be predefined and fixed (Order-(-1) compressor) or adjusted by symbol frequencies as the compression proceeds (Order-0). More advanced compressors allow for conditional probabilities based on the the preceding character (Order-1), or even more (Order-N). These *PPM* (Prediction by Partial Matching) compressors are among the best known.

Also as coding proceeds, the symbol frequencies are updated according to the processed data so that the symbol statistics and data encoding reflect the data.

Decoding involves checking the current code value against the range for each byte, emitting the correct byte, subtracting its lower limit and adjusting the range of the number.

The best compressors achieve about 2.0 bits/byte, averaged over a reference corpus of files.

6.4 Burrows Wheeler compression

Both LZ-7x and PPM compressors process the data sequentially, symbol by symbol. The Burrows Wheeler compressors by contrast process the data in blocks of perhaps 1 million bytes (which is often the whole file).

The BW algorithm descriptions refer to Figure 3.

6.4.1 The compression

Take the input string and write all of its cyclic rotations to form a square matrix. Sort the matrix rows into increasing lexicographic order. Emit the symbols in the last column, and also the index of the row with the original input.

The sorting collects together similar contexts; because similar contexts probably have similar symbols, there tend to be only a few emitted symbols in any neighbourhood of the transmitted data. This *locality* is captured by a recency or Move-To-Front recoding. (How many *different* symbols have occurred since this one was last seen?)

In this example the emitted string is `pssmipissii` which after MTF recoding becomes $ps0mi313010$, with many small values and, often, runs of zeros. These recoded values are finally compressed by a Huffman or arithmetic coder. (Most text files have about 60% zeros after MTF recoding. The MTF value n typically occurs with probability $P(n) \propto n^{-2}$.)

	sym-bol	context	Index	sym-bol	con-text	link
	p	imississip	1	p	i...	5
	s	ippimissis	2	s	i...	7
	s	issippimis	3	s	i...	10
	m	ississippi	4	m	i...	11
→	i	mississipp	5	i	m...	4
	p	pimississi	6	p	p...	1
	i	ppimississ	7	i	p...	6
	s	sippimissi	8	s	s...	2
	s	sissippimi	9	s	s...	3
	i	ssippimiss	10	i	s...	8
	i	ssissippim	11	i	s...	9

Figure 3: The forward and reverse transformations

6.4.2 The decompression

The first steps of decompression reverse firstly the statistical coding and then the MTF recoding. This recovers the permuted symbols which are the last column of the matrix, as above.

To recover the data note first that the matrix columns are permutations of one another. The first column (which has the *trailing* contexts of the emitted symbols) is recovered by sorting the symbols. We therefore have {*symbol*, *context*} pairs for each of the permuted symbols, which match first to first, second to second, and so on.

In this example link the first `i` context to the first `i` data and so on for all the `i`'s, and similarly for the `m`, `p`'s and `s`'s in order, building the last column of Figure 3. Traversing this list from any starting position gives a corresponding rotation of the input text or, equivalently, the following context of that symbol. The emitted row index (5 in this case) just selects that context that happens to be the original unrotated data.

(*If all of this seems like magic, well it isn't really. But the Burrows Wheeler algorithm is very subtle and elegant. And, quite incidentally, David Wheeler worked on the very first computers and wrote his first computer programs in about 1948! He published this algorithm in 1994.*)

Burrows Wheeler compressors are fast and give excellent compression. They are not quite as good as the best PPM compressors, but are very close. The BZIP2 compressor is widely used for program distribution.

P	$P \log_2 \frac{1}{P}$									
0.00	0.0000	0.0100	0.0179	0.0251	0.0319	0.0382	0.0443	0.0501	0.0557	0.0612
0.01	0.0664	0.0716	0.0766	0.0815	0.0862	0.0909	0.0955	0.0999	0.1043	0.1086
0.02	0.1129	0.1170	0.1211	0.1252	0.1291	0.1331	0.1369	0.1407	0.1444	0.1481
0.03	0.1518	0.1554	0.1589	0.1624	0.1659	0.1693	0.1727	0.1760	0.1793	0.1825
0.04	0.1858	0.1889	0.1921	0.1952	0.1983	0.2013	0.2043	0.2073	0.2103	0.2132
0.05	0.2161	0.2190	0.2218	0.2246	0.2274	0.2301	0.2329	0.2356	0.2383	0.2409
0.06	0.2435	0.2461	0.2487	0.2513	0.2538	0.2563	0.2588	0.2613	0.2637	0.2662
0.07	0.2686	0.2709	0.2733	0.2756	0.2780	0.2803	0.2826	0.2848	0.2871	0.2893
0.08	0.2915	0.2937	0.2959	0.2980	0.3002	0.3023	0.3044	0.3065	0.3086	0.3106
0.09	0.3127	0.3147	0.3167	0.3187	0.3207	0.3226	0.3246	0.3265	0.3284	0.3303
0.10	0.3322	0.3341	0.3359	0.3378	0.3396	0.3414	0.3432	0.3450	0.3468	0.3485
0.11	0.3503	0.3520	0.3537	0.3555	0.3572	0.3588	0.3605	0.3622	0.3638	0.3654
0.12	0.3671	0.3687	0.3703	0.3719	0.3734	0.3750	0.3766	0.3781	0.3796	0.3811
0.13	0.3826	0.3841	0.3856	0.3871	0.3886	0.3900	0.3915	0.3929	0.3943	0.3957
0.14	0.3971	0.3985	0.3999	0.4012	0.4026	0.4040	0.4053	0.4066	0.4079	0.4092
0.15	0.4105	0.4118	0.4131	0.4144	0.4156	0.4169	0.4181	0.4194	0.4206	0.4218
0.16	0.4230	0.4242	0.4254	0.4266	0.4278	0.4289	0.4301	0.4312	0.4323	0.4335
0.17	0.4346	0.4357	0.4368	0.4379	0.4390	0.4401	0.4411	0.4422	0.4432	0.4443
0.18	0.4453	0.4463	0.4474	0.4484	0.4494	0.4504	0.4514	0.4523	0.4533	0.4543
0.19	0.4552	0.4562	0.4571	0.4581	0.4590	0.4599	0.4608	0.4617	0.4626	0.4635
0.20	0.4644	0.4653	0.4661	0.4670	0.4678	0.4687	0.4695	0.4704	0.4712	0.4720
0.21	0.4728	0.4736	0.4744	0.4752	0.4760	0.4768	0.4776	0.4783	0.4791	0.4798
0.22	0.4806	0.4813	0.4820	0.4828	0.4835	0.4842	0.4849	0.4856	0.4863	0.4870
0.23	0.4877	0.4883	0.4890	0.4897	0.4903	0.4910	0.4916	0.4923	0.4929	0.4935
0.24	0.4941	0.4947	0.4954	0.4960	0.4966	0.4971	0.4977	0.4983	0.4989	0.4994
0.25	0.5000	0.5006	0.5011	0.5016	0.5022	0.5027	0.5032	0.5038	0.5043	0.5048
0.26	0.5053	0.5058	0.5063	0.5068	0.5072	0.5077	0.5082	0.5087	0.5091	0.5096
0.27	0.5100	0.5105	0.5109	0.5113	0.5118	0.5122	0.5126	0.5130	0.5134	0.5138
0.28	0.5142	0.5146	0.5150	0.5154	0.5158	0.5161	0.5165	0.5169	0.5172	0.5176
0.29	0.5179	0.5182	0.5186	0.5189	0.5192	0.5196	0.5199	0.5202	0.5205	0.5208
0.30	0.5211	0.5214	0.5217	0.5220	0.5222	0.5225	0.5228	0.5230	0.5233	0.5235
0.31	0.5238	0.5240	0.5243	0.5245	0.5247	0.5250	0.5252	0.5254	0.5256	0.5258
0.32	0.5260	0.5262	0.5264	0.5266	0.5268	0.5270	0.5272	0.5273	0.5275	0.5277
0.33	0.5278	0.5280	0.5281	0.5283	0.5284	0.5286	0.5287	0.5288	0.5289	0.5291
0.34	0.5292	0.5293	0.5294	0.5295	0.5296	0.5297	0.5298	0.5299	0.5299	0.5300
0.35	0.5301	0.5302	0.5302	0.5303	0.5304	0.5304	0.5305	0.5305	0.5305	0.5306
0.36	0.5306	0.5306	0.5307	0.5307	0.5307	0.5307	0.5307	0.5307	0.5307	0.5307
0.37	0.5307	0.5307	0.5307	0.5307	0.5307	0.5306	0.5306	0.5306	0.5305	0.5305
0.38	0.5305	0.5304	0.5304	0.5303	0.5302	0.5302	0.5301	0.5300	0.5300	0.5299
0.39	0.5298	0.5297	0.5296	0.5295	0.5294	0.5293	0.5292	0.5291	0.5290	0.5289
0.40	0.5288	0.5286	0.5285	0.5284	0.5283	0.5281	0.5280	0.5278	0.5277	0.5275
0.41	0.5274	0.5272	0.5271	0.5269	0.5267	0.5266	0.5264	0.5262	0.5260	0.5258
0.42	0.5256	0.5255	0.5253	0.5251	0.5249	0.5246	0.5244	0.5242	0.5240	0.5238
0.43	0.5236	0.5233	0.5231	0.5229	0.5226	0.5224	0.5222	0.5219	0.5217	0.5214
0.44	0.5211	0.5209	0.5206	0.5204	0.5201	0.5198	0.5195	0.5193	0.5190	0.5187
0.45	0.5184	0.5181	0.5178	0.5175	0.5172	0.5169	0.5166	0.5163	0.5160	0.5157
0.46	0.5153	0.5150	0.5147	0.5144	0.5140	0.5137	0.5133	0.5130	0.5127	0.5123
0.47	0.5120	0.5116	0.5112	0.5109	0.5105	0.5101	0.5098	0.5094	0.5090	0.5087
0.48	0.5083	0.5079	0.5075	0.5071	0.5067	0.5063	0.5059	0.5055	0.5051	0.5047
0.49	0.5043	0.5039	0.5034	0.5030	0.5026	0.5022	0.5017	0.5013	0.5009	0.5004
0.50	0.5000	0.4996	0.4991	0.4987	0.4982	0.4978	0.4973	0.4968	0.4964	0.4959
0.51	0.4954	0.4950	0.4945	0.4940	0.4935	0.4930	0.4926	0.4921	0.4916	0.4911
0.52	0.4906	0.4901	0.4896	0.4891	0.4886	0.4880	0.4875	0.4870	0.4865	0.4860
0.53	0.4854	0.4849	0.4844	0.4839	0.4833	0.4828	0.4822	0.4817	0.4811	0.4806
0.54	0.4800	0.4795	0.4789	0.4784	0.4778	0.4772	0.4767	0.4761	0.4755	0.4750
0.55	0.4744	0.4738	0.4732	0.4726	0.4720	0.4714	0.4708	0.4702	0.4696	0.4690
0.56	0.4684	0.4678	0.4672	0.4666	0.4660	0.4654	0.4648	0.4641	0.4635	0.4629
0.57	0.4623	0.4616	0.4610	0.4603	0.4597	0.4591	0.4584	0.4578	0.4571	0.4565
0.58	0.4558	0.4551	0.4545	0.4538	0.4532	0.4525	0.4518	0.4511	0.4505	0.4498
0.59	0.4491	0.4484	0.4477	0.4471	0.4464	0.4457	0.4450	0.4443	0.4436	0.4429
0.60	0.4422	0.4415	0.4408	0.4401	0.4393	0.4386	0.4379	0.4372	0.4365	0.4357
0.61	0.4350	0.4343	0.4335	0.4328	0.4321	0.4313	0.4306	0.4298	0.4291	0.4283
0.62	0.4276	0.4268	0.4261	0.4253	0.4246	0.4238	0.4230	0.4223	0.4215	0.4207
0.63	0.4199	0.4192	0.4184	0.4176	0.4168	0.4160	0.4152	0.4145	0.4137	0.4129
0.64	0.4121	0.4113	0.4105	0.4097	0.4089	0.4080	0.4072	0.4064	0.4056	0.4048
0.65	0.4040	0.4031	0.4023	0.4015	0.4007	0.3998	0.3990	0.3982	0.3973	0.3965
0.66	0.3956	0.3948	0.3940	0.3931	0.3923	0.3914	0.3905	0.3897	0.3888	0.3880
0.67	0.3871	0.3862	0.3854	0.3845	0.3836	0.3828	0.3819	0.3810	0.3801	0.3792
0.68	0.3783	0.3775	0.3766	0.3757	0.3748	0.3739	0.3730	0.3721	0.3712	0.3703
0.69	0.3694	0.3685	0.3676	0.3666	0.3657	0.3648	0.3639	0.3630	0.3621	0.3611
0.70	0.3602	0.3593	0.3583	0.3574	0.3565	0.3555	0.3546	0.3537	0.3527	0.3518
0.71	0.3508	0.3499	0.3489	0.3480	0.3470	0.3460	0.3451	0.3441	0.3432	0.3422
0.72	0.3412	0.3403	0.3393	0.3383	0.3373	0.3364	0.3354	0.3344	0.3334	0.3324
0.73	0.3314	0.3305	0.3295	0.3285	0.3275	0.3265	0.3255	0.3245	0.3235	0.3225
0.74	0.3215	0.3204	0.3194	0.3184	0.3174	0.3164	0.3154	0.3144	0.3133	0.3123
0.75	0.3113	0.3102	0.3092	0.3082	0.3072	0.3061	0.3051	0.3040	0.3030	0.3020
0.76	0.3009	0.2999	0.2988	0.2978	0.2967	0.2956	0.2946	0.2935	0.2925	0.2914
0.77	0.2903	0.2893	0.2882	0.2871	0.2861	0.2850	0.2839	0.2828	0.2818	0.2807
0.78	0.2796	0.2785	0.2774	0.2763	0.2752	0.2741	0.2731	0.2720	0.2709	0.2698
0.79	0.2687	0.2676	0.2664	0.2653	0.2642	0.2631	0.2620	0.2609	0.2598	0.2587
0.80	0.2575	0.2564	0.2553	0.2542	0.2530	0.2519	0.2508	0.2497	0.2485	0.2474
0.81	0.2462	0.2451	0.2440	0.2428	0.2417	0.2405	0.2394	0.2382	0.2371	0.2359
0.82	0.2348	0.2336	0.2325	0.2313	0.2301	0.2290	0.2278	0.2266	0.2255	0.2243
0.83	0.2231	0.2219	0.2208	0.2196	0.2184	0.2172	0.2160	0.2149	0.2137	0.2125
0.84	0.2113	0.2101	0.2089	0.2077	0.2065	0.2053	0.2041	0.2029	0.2017	0.2005
0.85	0.1993	0.1981	0.1969	0.1957	0.1944	0.1932	0.1920	0.1908	0.1896	0.1884
0.86	0.1871	0.1859	0.1847	0.1834	0.1822	0.1810	0.1797	0.1785	0.1773	0.1760
0.87	0.1748	0.1736	0.1723	0.1711	0.1698	0.1686	0.1673	0.1661	0.1648	0.1635
0.88	0.1623	0.1610	0.1598	0.1585	0.1572	0.1560	0.1547	0.1534	0.1522	0.1509
0.89	0.1496	0.1484	0.1471	0.1458	0.1445	0.1432	0.1420	0.1407	0.1394	0.1381
0.90	0.1368	0.1355	0.1342	0.1329	0.1316	0.1303	0.1290	0.1277	0.1264	0.1251
0.91	0.1238	0.1225	0.1212	0.1199	0.1186	0.1173	0.1159	0.1146	0.1133	0.1120
0.92	0.1107	0.1093	0.1080	0.1067	0.1054	0.1040	0.1027	0.1014	0.1000	0.0987
0.93	0.0974	0.0960	0.0947	0.0933	0.0920	0.0907	0.0893	0.0880	0.0866	0.0853
0.94	0.0839	0.0826	0.0812	0.0798	0.0785	0.0771	0.0758	0.0744	0.0730	0.0717
0.95	0.0703	0.0689	0.0676	0.0662	0.0648	0.0634	0.0621	0.0607	0.0593	0.0579
0.96	0.0565	0.0552	0.0538	0.0524	0.0510	0.0496	0.0482	0.0468	0.0454	0.0440
0.97	0.0426	0.0412	0.0398	0.0384	0.0370	0.0356	0.0342	0.0328	0.0314	0.0300
0.98	0.0286	0.0271	0.0257	0.0243	0.0229	0.0215	0.0201	0.0186	0.0172	0.0158
0.99	0.0144	0.0129	0.0115	0.0101	0.0086	0.0072	0.0058	0.0043	0.0029	0.0014

Table 1: The Entropy Function $H(P) = P \log_2 \frac{1}{P}$