

CE 263

W99

Data Compression

Glen G. Langdon, Jr.
Department of Computer Engineering
University of California at Santa Cruz

©1999, Glen G. Langdon, Jr.

Chapter 4

Introduction to Information Theory

Introduction

This chapter presents some *theory* behind statistical data compression, based on foundations from the previous chapters. In *Chapter One*, we covered the important points and properties required of probability distributions. Random variables X or Y have properties one can calculate, such as the mean, variance and standard deviation. The mean is denoted μ_X or μ_Y , also denoted \bar{X} or \bar{Y} . The variance of X is denoted as σ_X^2 or $\text{Var}(X)$. The standard deviation is denoted σ_X .

In *Chapter Two*, data compression was demonstrated by using a prefix code on 2-bit symbols such that the data compressed to 1.75 bits per symbol. The demonstration used an FVL code. In *Chapter Three*, the Shannon-Fano algorithm and the Huffman algorithm provide a design method for VL Codes. A critical factor to compression with these algorithms is the *best* estimate of the symbol probabilities. Moreover, the best compression is achieved by data compression algorithms when at least one symbol is very popular. On the other hand, the probability distributions themselves may not be cooperative. Very little compression is achieved when the statistical events to be compressed are equally likely.

Although the idea of short codewords for frequent symbols was known prior to Shannon [Sha 48], it was Shannon who provided the theoretical basis for the field of information theory. Information theory is a field whose *initial* development is largely attributed to Claude Shannon [Sha 48], and others such as Norbert Wiener [Wie 48] (logarithmic measure), [Wie 49] (credited in [Sha 48]), Robert Fano [Fan 61] (credited in [Sha 48]), David Huffman [Huf 52], Gilbert [Gil 52] and McMillan [McM 53].¹

The mid-1940s through the mid-1950s laid down a mathematical basis not only for data compression but

¹Although the statistician R. A. Fischer in 1925 introduced in the technical sense a definition of “information” [Kul 83], Fischer’s information applies to statistical estimation and differs from Shannon’s definition of information.

also the problem of handling a “noisy” *channel* or communications link.

Definition 1 (Channel) *Shannon defined a channel as a communication medium, which could be communication by radio, telephone lines, telegraph lines, etc.*

Today, we can view a communication channel as any analog or digital communication media such as a SCSI or modem connection, ethernet, satellite link, connections on a printed circuit board, and even a magnetic tape or hard drive storage device. The “static” or “noise” on a channel such as a telephone line or radio link can alter the values of the bits in code strings. Our treatment assumes a noiseless channel, or the absence of noise, since error correction is beyond the scope of lossless data compression. However, a service performed by data compression is to reduce the number of bits transmitted through a noisy channel. Error detection and correction codes are available for various environments, and the choice of how to protect the integrity of encoded data is based upon the expected noise (burst errors, random errors, etc). We note that the field of error detection and correction has its roots in information theory.

4.1 The Abstract Information Source

The notion of *information source* was formalized in [Sha 48]. Shannon defined an abstraction called a *discrete information source*. In Figure 4.1, we illustrate a six-symbol alphabet \mathcal{A} , with the uniform distribution, from which several samples have been drawn at random. In the example of 6 rolls of one die in Figure 4.1, values 2, 4, 5, and 6 appear once, and value 3 appears twice. Value 1 does not appear at all.

Here, we treat only discrete sources with a finite number of symbols, although Shannon theory also treats information in the “continuous” (analog signal)

Information Source

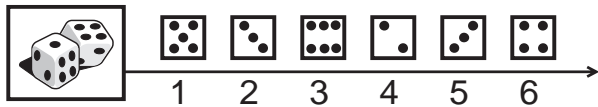


Figure 4.1: Example of an Information Source

domain. Shannon’s major contributions to information theory were in the late 1940s and the 1950s, when the analog world reigned. Today, the world is mercifully digital, and digital systems handle most of the needs of data compression .

In the half-century after Shannon’s pioneering work, the current technology converts analog signals to a sequence of digital values. An analog to digital (A/D) converter² uses a clock to “sample”: the continuous (*analog*) waveforms at a pre-determined frequency, and digitizes each sample. Thus, analog waveforms are transformed into sequences of digital samples. Sampled analog signals has become the most popular technique for processing continuous waveforms such as audio and visual information, so they can be handled and manipulated on a *digital* computer.

Conceptually, an information source (or simply *source*), captures the statistical nature of the symbol occurrences in an arbitrarily large sequence. Imagine a gremlin emitting the symbols s_k in a sequence $S = s_1, s_2, \dots, s_k$, etc.

Let the symbol in each position k come from the set (or *M-ary alphabet*) \mathcal{A} of M distinct symbols a_m , where m runs from 1 to M . Symbols a_m form the *source alphabet*. The simplest case is when each symbol of the source alphabet occurs in the source output sequence according to its own *relative frequency* (probability) $p(a_m)$. (Note that the subscript 2 in s_2 indicates position 2 in the sequence, and the number within parentheses, as in a_2 , indicates the second member of the symbol alphabet \mathcal{A} .)

The simplest source is called a *memoryless* source, or a *zero memory* source by Abramson [Abr 63], and a *zero-order Markov* source by others.

Observation 1 (Beware) *The reader should beware: the terminology in [Sha 48] employs a now obsolete definition of the order numbering of higher-order sources. Shannon called the uniform distribution a zero-order model; but today the uniform distribution is simply a probability distribution all of whose probabilities are the same.*

Today, a *zero-order* source is no more than a se-

²A/D is pronounced “A to D”.

quence S whose symbols appear randomly drawn from a single distribution. Statisticians call such a sequence an *independent and identically distributed* (i.i.d.) sequence. For the statistician, the symbols are drawn from the same (identical) distribution: no symbol position is filled from some other distribution. The term *independent* means the production of the next symbol does not depend on (or ignores) any conditions or events based on previously produced symbols.

The relative frequencies of symbols emitted from a memoryless source are described by a single probability distribution $\mathcal{D}_{\mathcal{A}}$ on an alphabet \mathcal{A} . The conditional probability that some symbol u of \mathcal{A} has a different probability of occurrence when it follows symbol q is the same as its probability for following any of the other symbols. Therefore: a zero-order model (or Markov source, or simply “source”) is the *simplest* meaningful assumption for us to consider relative to a code such as the Huffman code. The simplicity of the memoryless source is because we only deal with a single probability distribution.

Technically, the term “memoryless” means the source has *no memory* of the *past history* of the symbols emitted. A so-called “memoryless” (zero-order) source doesn’t forget to produce each symbol a_m according to its probability value $p(a_m)$ that never changes.

Observation 2 (Memoryless) *A memoryless source remembers its probability distribution, but forgets the values of previous symbols.*

Basically, the meaning of a memoryless, or *zero-order* source, is the following.

Definition 2 *Successive symbols in sequence S emitted from a memoryless source have the property of statistical independence. See Eq 4.1, which assumes a single (unconditional) probability distribution $\mathcal{D}_{\mathcal{A}}$ for alphabet \mathcal{A} . This means, for all symbols a_m :*

$$p(a_m|a_{m-1}) = p(a_m). \quad (4.1)$$

In other words, for a zero-order source, the probability of each symbol a_m is the same no matter what symbol a_{m-1} precedes it. A memoryless source is assumed to have no inter-symbol influences (or dependencies, or conditional probabilities) on probability distribution $\mathcal{D}_{\mathcal{A}}$.

Definition 3 (Stationary) *Sequence S emitted from a memoryless source is stationary. The property of stationarity means that sequence S exhibits symbol*

histograms of almost all relatively long subsequences of S , and these subsequence histograms are close to the histogram of the entire sequence S itself.

Since each symbol's relative frequency is the same regardless of when or where in the sequence any given symbol is emitted from the source, the memoryless source is very easy to characterize. Given a data sequence (or computer file) S^{12} of 12 bytes, where $S^{12} = s_1, \dots, s_{12}$, the probability of the message is the product of the probabilities of the symbols comprising the message. Independence follows from a memoryless source, and we avoid the complication of dealing with conditional probabilities or which symbol follows the preceding symbol.³ We can reorder the factors of a product without changing the value of the product.

$$p(S^{12}) = p(s_1, \dots, s_{12}) = p(s_1) \times p(s_2) \times \dots \times p(s_{12}). \quad (4.2)$$

In Eq 4.2, the *right-hand side* comes from probability theory in that the probability of a sequence of outcomes, is the product of the appropriate (dependent or independent) probability for each outcome of the sequence.

If a zero-order source alphabet \mathcal{A} has M members, typical member a_m , then the information source “can be” characterized by specifying only $M-1$ probability values, $p(a_1), \dots, p(a_m) \dots p(a_{M-1})$ because the M -th probability is unity less the sum of the $M-1$ known probabilities:

$$\sum_{m=1}^{M-1} p(a_m) = 1 - p(a_M),$$

$$\text{and } p(a_M) = 1 - \sum_{m=1}^{M-1} p(a_m) \quad (4.3)$$

One infers the so-called *empirical* probability distribution of an N -symbol zero-order sequence \mathcal{S} from its symbol counts $\text{Ct}(a_m)$. For example, if $\text{Ct}(a_m)$ is the frequency of symbol a_m in length N sequence \mathcal{S}^N , then the relative frequency (or probability) $p(a_m) = \text{Ct}(a_m)/N$. Any long sequence \mathcal{S} of symbols from alphabet \mathcal{A} emitted from the memoryless source is expected to exhibit relative frequency $p(a_m)$ for symbol a_m . In other words, each next symbol in the sequence is obtained (or “drawn”) from probability distribution \mathcal{D} , as reflected in the relative frequencies.

³For a memoryless source, recall that symbols are drawn from the same probability distribution.

4.2 Self-information and Ideal Length†

† Portions of this subsection, and Tables A1 and A2 of Appendix II at the end of this Chapter were contributed by the author's colleague David Huffman

Definition 4 (Self-information)

The self-information of a data item or symbol a_m , denoted $il(a_m)$, is the negative of the logarithm base 2 of $p(a_m)$:

$$il(a_m) = -\log_2 a_m \text{ bits.} \quad (4.4)$$

By using logarithm base 2, the value $il(a_m)$ has the units of bits. When the logarithm base is the natural logarithm base e , the self-information units are called *nats*. In early texts (some decades ago), the logarithm to base 10 was also used, and the unit of measure for the self-information was called *hartleys*.

When an event \mathcal{E} with an *a priori* probability $p = \frac{1}{2}$ occurs, by definition, one bit of “Shannon” information has occurred. More generally, if the event has an *a priori* probability p , then the corresponding information measure is $\log_2 \frac{1}{p}$ bits where \log_2 means logarithm to the base 2.

Observation 3 Often an identity related to logarithms, $-\log \frac{1}{x} = \log x$, is used to express the self-information as $il(a_m) = \log \frac{1}{p(a_m)}$, a form that avoids the confusion of a minus sign preceding an ideal length value we know is positive.

See Appendix I of this chapter for a refresher on manipulating logarithms. Appendix II has tables of probabilities versus self-information.

Shannon's notion of information requires the existence of a probability distribution. The smaller the event's probability, i.e. the rarer the event, and hence, the larger the *surprise*. The popularity of an event is not related to its information content or meaning. It was Shannon who connected the self-information with the degree of surprise when the event occurs.

A message that is *very rare*, i.e., occurs very infrequently, has a large value of self-information. An event of probability one is a certainty so the self-information is 0 (no other event can occur). The *nice thing* about a certain event is that is a *certainty* carries zero information, so: *we can encode the message in zero bits!*

Given things or events that occur with known probabilities, Shannon assigns a value of *self-information*, which we denote il , to those events. For notation il , think of the words *ideal length*. The notion of “length” is appropriate here because we measure self-information in bits, and we can also measure the size

of a data file in bits. The word “ideal” is also true because the self-information of an event \mathcal{E} , ($il(\mathcal{E})$), is the optimal length (in the data compression sense) of the codeword for \mathcal{E} . Recall that Huffman coding assigns the most popular event or symbol the shortest codeword, while less popular events are assigned the longer codewords. We anticipate the inverse relationship where the larger the probability, the smaller the self-information.

The self-information function il has as its domain a probability value varying between 0 and 1. Self-information values range respectively from an arbitrarily large number for rare events, to the values 0 for a certainty (probability of 1). The unit of self-information, when the logarithm is to the base 2, is called the *bit*, a contraction of binary digit. If the logarithm for self-information is the natural logarithm, the logarithm base is e , and information is measured in “natural units” or *nats*. Similarly, logarithm base 256 yields “bytes”.

The most common form of compressed data is the *binary code string* (a sequence of bits of value 0 and 1). Technically, the self-information unit of “bit” is not the same as a digit of the binary number system. However, the length (in bits) of a binary code string composed of 0s and 1s, is the number of binary values or bit positions that comprise the code string.

Under “ideal” conditions, *the number of data bits in the compressed version* (code string) for a data sequence is equal (to within a fraction of a bit) to the value, in bits, of *the self-information of that data*. One of Shannon’s major achievements, in non-technical terms, is the definition of the appropriate conditions for optimally encoding a data file: one cannot do better than coding a symbol in such a way that the code string length increase (in bits) is the symbol’s self-information (in bits). Since the self-information for symbol a_m is $\log_2 \frac{1}{p_m}$, the self-information value $il(a_m)$ is termed the *ideal code length* in bits for encoding a_m .

4.2.1 Units of self-information

The units of self-information *scale* in the “expected way”, when changing the base of the logarithm. A physical length of 1 inch does not change after conversion to a length of 2.54 centimeters. Neither does the adjusted value of self-information change when the base of the logarithm changes. The respective *logarithm base 2* measures the self-information il in bits $\{0, 1\}$. For logarithm base 16 (base 2^4), the unit is the 4-bit hexadecimal digit: $\{0, 1, \dots, 9, A, \dots, F\}$. Logarithms to the base 256 use the unit of the 8-bit byte, since $\log_2 256$ gives 8.

As long as we maintain the binary code alphabet,

the *self-information converts between units in the expected way*. Thus, taking $-\log_{256}(p)$ and measuring the result *in bytes* gives the same self-information as $-\log_2(p)$ *in bits*.

To convert self-information in bits to bytes: *divide the number of bits by 8 to get bytes*. To convert self-information from base 256 (in bytes) back to bits (base 2), multiply the bytes (base 256) by $8 = \log_2 256$ to get bits.

If the self-information is in base 2 (in bits), to convert to bytes (base 256), multiply the bits by $\frac{1}{8}$ which is $\log_{256} 2$. In other words, $256^{0.125} = 2$, or put another way:

$$256^{\frac{1}{8}} = 2. \quad (4.5)$$

In general, given a self-information value $x = \log_2 z$ bits, and we actually want the self-information of z in base y units, then we can convert the self-information of z to another logarithm base y (eg., base 256), as follows:

$$\log_y x = \frac{x}{\log_2 y} \quad (4.6)$$

Example 1 (Self-information) Let $p = \frac{1}{256} = 2^{-8} = 16^{-2} = 256^{-1}$.

The self-information of Example 1 is one 8-bit byte, two 4-bit hex digits, or eight bits. One byte has 256 possible values, as do two hex digits, as do eight bits. *Why is this so?* The answer lies in representing $\frac{1}{256}$ in three different number bases, as follows.

Since $\frac{1}{256}$ can be viewed as 256^{-1} , or 16^{-2} , or 2^{-8} , the units of self-information change as we change the number base. We normally use base 2, because of the base 2 coding alphabet. Example 1 expands the opportunities for using a code alphabet other than the binary alphabet.

Example 2 Calculate the value of self-information $il(\mathbf{a})$ of a probability $p(\mathbf{a}) = \frac{1}{256}$ using logarithm base 256.

The self-information unit, with the logarithm to base 256, is the 8-bit byte.

$$il(\mathbf{a}) = -\log_{256} \frac{1}{256} = +\log_{256} 256 = 1 \quad (4.7)$$

The value of $\log_{256}(256)$ is 1, which means 1 byte of self-information. But just as the self-information unit,

when using $-\log_2(p)$, results in the self-information dimension of *bits*. On the other hand, the use of $-\log_{256}(p)$ yields a self-information result for probability p in 8-bit *bytes*. In other words, to represent the self-information of $\frac{1}{256}$ in bits, we can convert 1 byte to 8 bits, or: we can “do it the hard way”, and calculate the self-information $il(\frac{1}{256})$ in bits using the logarithm base 2 as follows:

$$il(\frac{1}{256}) = -\log_2 \frac{1}{256} = +\log_2 256 = 8 \text{ bits} \quad (4.8)$$

Again looking at the situation in base 16:

$$-\log_{16} \frac{1}{256} = +\log_{256} 256 - \log_{256} 1 = 1 \text{ byte} \quad (4.9)$$

In another example, we can calculate the self-information as $-\log_{16}(p)$ in *hexadecimal*, and do a “hex-to-bits” conversion to obtain the self-information in bits⁴.

The base 256 digit is two hex digits, which is also 8 bits. Normally, the base represents the size of the code alphabet. For example if we are coding to 8-bit bytes, then $-\log_{256}(p)$ gives the number of bytes for coding an event of probability p in the ideal case.

If the digits of the code alphabet are hexadecimal digits, the self-information is calculated by the base-16 logarithm. If we assign 4 bits for each hexadecimal digit, which we could (the self-information of 16 equally likely events is 4 bits) the hexadecimal self-information is converted to the same number of bits obtained by calculating the self-information using base-2 logarithms.

More Examples:

1. An event (or symbol) whose probability is $\frac{1}{2}$, has one bit of self-information.
2. A symbol of probability $\frac{1}{16}$ has 4 bits of self-information.
3. A two-symbol (binary) data string \mathcal{S} where $p(s_1) = \frac{1}{4}$ and $p(s_2) = \frac{1}{8}$ has probability $\frac{1}{32}$ and ideal length (self-information) of $-\log_2 \frac{1}{32} = \log_2 32 = 5$ bits.

4.3 Entropy of a source

In the field of statistics, a discrete *random variable* Y has two properties:

⁴Given that one byte equals 8 bits.

1. Random variable Y takes on discrete numerical values, y_m , and
2. Y has a probability distribution: each value y_m of Y has an assigned probability $p(y_m)$, such that:

$$\sum_{m=1}^M p(y_m) = 1. \quad (4.10)$$

The mean of random variable Y is denoted \bar{Y} . The symbols y_m of a random variable Y have probability $p(y_m)$ within distribution \mathcal{D}_Y , and therefore have a value of self-information, $il(y_m)$.

Observation 4

For most purposes, the self-information is measured in bits, because of the popularity of the binary code alphabet. The self-information $il(y_m)$ corresponding to $p(y_m)$ of Y of a memoryless source is a random variable.

Definition 5 *The entropy H of a discrete memoryless source whose symbols belong to alphabet \mathcal{A} , and whose distribution is $\mathcal{D}_{\mathcal{A}}$, is the expected value $E(il(a_m))$ (or mean value) of the self-information of distribution $\mathcal{D}_{\mathcal{A}}$.*

The statistical *expectation* $E(Y)$ (or mean \bar{Y}) of random variable Y is calculated in one of two ways. The first assumes we have a sequence \mathcal{S} of N values s_k , where $\mathcal{S} = s_1, s_2, \dots, s_k, \dots, s_N$.

- The average value is determined by adding the N values of \mathcal{S} , $\sum_{k=1}^N s_k$, then dividing the resulting sum by N :

$$\bar{Y} = \frac{1}{N} \sum_{k=1}^N s_k.$$

- Given the probability $p(y_m)$, or (which is the same) relative frequency $\frac{Ct(y_m)}{N}$, of each value y_m of Y , one calculates the *mean*, \bar{Y} , as a *weighted sum* of the values of the variable Y , using weights $p(y_m)$:

$$\bar{Y} = \sum_{m=1}^M p(y_m) \times y_m. \quad (4.11)$$

The “weight” assigned to each value is its probability (or which is the same, its relative frequency). The *entropy*, H , of a given *distribution* \mathcal{D} , is denoted $H(\mathcal{D})$.

$$H(\mathcal{D}_M) = \sum_{m=1}^M p(a_m) \times -\log_2 p(a_m). \quad (4.12)$$

Given a discrete memoryless source comprised of the set of source symbols a_m of alphabet \mathcal{A} , each of respective probability $p(a_m)$, the source entropy H in bits per symbol is directly calculated as a product by Eq 4.12. The first factor $p(a_m)$ is the proportion of occurrences of symbol a_m , and the second factor $i\ell(a_m) = -\log_2 p(a_m)$ is a numerical value for a_m 's self-information.

The summation of Eq 4.12 produces the mean (average) of the self-information. For the right hand side of Eq 4.12, the first factor behind the sum is the "weight" of the second factor: the self-information, such that the left-hand side, the entropy H of the probability distribution \mathcal{D}_M , is the weighted sum of the self-information of distribution \mathcal{D}_M .

Given the symbol counts (frequencies) of a data file \mathcal{S} , the relative frequencies, also known as the *empirical* probabilities, are used to calculate the empirical entropy of file \mathcal{S} under the memoryless source assumption through Equation 4.12.

The self-information value (in bits) is weighted by its relative frequency: the entropy is a weighted sum of the self-information. Since the weights associated with the objects sum to unity, the unit of the entropy value is *bits per symbol* if the objects associated with the relative frequencies are called symbols. If compressing raw bytes of file \mathcal{S} , one could also use the term *bits per byte* to represent the units for the entropy calculation, as well as the result of compressing file \mathcal{S} with some algorithm.

Some Conclusions

1. The notion of *entropy* is a key concept in information theory. Entropy depends upon some underlying assumptions. Here, we have only treated the entropy for a *memoryless* source. Higher order sources are treated later.
2. The source entropy is the *average amount of self-information per symbol*. Coding performance reaches the ideal code length *if* we can assign codes to symbols according to the optimum choice.
3. The optimum choice for each symbol a_m is to assign its *ideal length* $i\ell(a_m)$ as the code length $cl(a_m)$. For prefix codes that use integer-length codewords, the optimal assignment is not always possible.

4.3.1 Entropy of Binary Sources[†]

[†] This subsection contributed by David Huffman

The simplest case for computing the entropy is the *binary distribution* of two possible and statistically independent events or outcomes. An example event \mathcal{E} is a *coin toss* having the probability p of coming up "heads" and the probability $1-p$ of coming up "tails". The notion *independence* here corresponds to the intuitively correct notion that the *outcome* of one toss of the coin is independent of the results of the outcomes of earlier tosses.

Two equivalent forms for calculating the *binary entropy* H are expressed as Eqs 4.13 and 4.14:

$$H = p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{(1-p)} \quad (4.13)$$

$$H = -[p \log_2 p + (1-p) \log_2 (1-p)]. \quad (4.14)$$

In Eq 4.13 we see the information values (the self-information) $\log_2 \frac{1}{p}$ and $\log_2 \frac{1}{(1-p)}$, respectively *weighted* by their relative frequency. Intuitively; self-information $\log_2 \frac{1}{p}$ occurs portion p of the time. In Eq 4.14, the negative value between square brackets is created as a negative value, and then converted to a positive value by the leading minus sign. Note that in each case, the calculation is an *inner product* of a first two-component vector representing a binary probability distribution and a second two-component vector representing the self-information.

The general shape of the *binary entropy function* (generally denoted $H(p)$ in the literature) appears in Figure 4.2. The the *maximum* value of H occurs when the two symbols are *equally likely*: $p = (1-p) = \frac{1}{2}$.

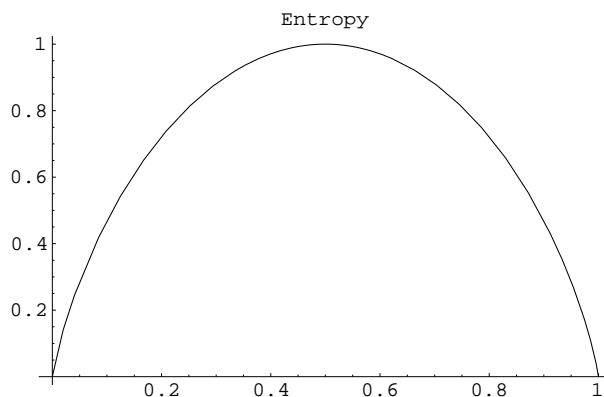


Figure 4.2: Entropy of Binary Source, H versus p

Example calculations of H(p) for a binary distribution

Example 3 Let one of the probabilities \underline{p} of a binary distribution be $p = 0.25$. What is the binary entropy $H(0.25)$?

For $p = \frac{1}{4}$, $H(p) =$

$$\begin{aligned} \frac{1}{4} \log_2 4 + \frac{3}{4} \log_2 \frac{4}{3} &= 0.5 + \frac{3}{4}(2 - \log_2 3) \\ &= 0.5 + 0.75 \times (2 - 1.585) = \underline{0.8113} \end{aligned} \quad (4.15)$$

If $p = \frac{1}{3}$:

$$\begin{aligned} H(p) &= \frac{1}{3} \log_2 3 + \frac{2}{3} \log_2 \frac{3}{2} \\ &= \underline{0.9183}. \end{aligned} \quad (4.16)$$

If $p = \frac{11}{100}$, and $1-p = \frac{89}{100}$:

$$\begin{aligned} H(p) &= \frac{11}{100} \log_2 \frac{100}{11} + \\ &\quad \frac{89}{100} \log_2 \frac{100}{89} \\ &= \underline{0.4999}. \end{aligned} \quad (4.17)$$

Given M symbols or events in M -ary alphabet \mathcal{A} with *a priori* distribution \mathcal{D}_A probabilities $p(a_1), p(a_2), \dots, p(a_m), \dots, p(a_M)$ such that:

$$\sum_{m=1}^M p(a_m) = 1, \quad (4.18)$$

The distribution's resulting entropy H_A is:

$$\begin{aligned} H_A &= \sum_{m=1}^M p(a_m) \log_2 \frac{1}{p(a_m)} \\ &= - \sum_{m=1}^M p(a_m) \log_2 p(a_m) \end{aligned} \quad (4.19)$$

The maximum possible entropy H_{MAX} from Eq 4.19 occurs when all values $p(a_m)$ equal $\frac{1}{M}$. In the maximum entropy case, each of the M terms, following the style of Eq 4.13, is $\log_2(\frac{1}{M})$, or $\forall a_m, i\ell(a_m) = \log_2 M$. The entropy is at a maximum. The entropy at its *maximum* is the sum of the symbols' self-information, $\log_2(\frac{1}{M})$, weighted by the symbols' probability $\frac{1}{M}$:

$$\begin{aligned} H_{MAX} &= \frac{1}{M} \log_2 M \\ &= \sum_{m=1}^M \frac{1}{M} \\ &= \log_2 M. \end{aligned} \quad (4.20)$$

And so we have derived the formula $H_{MAX} = \log_2 M$.

See Figure 4.2 as the example case $M = 2$. Observe from Figure 4.2 that the maximum binary entropy value H_2 of a binary source occurs when $p(0) = p(1) = \frac{1}{2}$.

4.3.2 Entropy H for a ternary sources

If $p = \frac{1}{3}$, then:

$$\begin{aligned} H(p) &= \frac{1}{3} \log_2 3 + \frac{2}{3} \log_2 \frac{3}{2} \\ &= \underline{0.9183}. \end{aligned} \quad (4.21)$$

If

$$\begin{aligned} \mathcal{D} &= \left\{ \frac{1}{2}, \frac{1}{3}, \frac{1}{6} \right\}, \\ &= \left\{ \frac{3}{6}, \frac{2}{6}, \frac{1}{6} \right\}. \end{aligned} \quad (4.22)$$

then $H_{\mathcal{D}} =$

$$\frac{3}{6} \log_2 2 + \frac{2}{6} \log_2 3 + \frac{1}{6} \log_2 6 \quad (4.23)$$

(1) Bring out common factor $\frac{1}{6}$:

$$= \frac{1}{6}(3 \log_2 2 + 2 \log_2 3 + \log_2 6) \quad (4.24)$$

(2) Replace $3 \log_2 2$ by "3"; and (3) replace $\log_2 6$ by $(\log_2 3) + (\log_2 2) = \log_2 3 + 1$

$$= \frac{1}{6}(3 + 2 \log_2 3 + \log_2 3 + 1) \quad (4.25)$$

(4) Replace $3+1$ by 4, and (5) replace $2 \log_2 3 + \log_2 3$ by $3 \log_2 3$.

$$= \frac{1}{6}(4 + 3 \log_2 3) \quad (4.26)$$

(6) Multiply out $\frac{1}{6}$:

$$= \frac{1}{6}(4 + 3 \log_2 3) = \frac{2}{3} + \frac{1}{3} \log_2 3 \quad (4.27)$$

$$= \underline{1.459} \text{ bits per symbol.} \quad (4.28)$$

If $\mathcal{D}_3 = \{\frac{2}{3}, \frac{1}{6}, \frac{1}{6}\} = \{\frac{4}{6}, \frac{1}{6}, \frac{1}{6}\}$, then:

$$\begin{aligned} H(\mathcal{D}_3) &= \frac{4}{6} \log_2 \frac{3}{2} + 2 \times \frac{1}{6} \log_2 6 \\ &= \underline{1.252} \text{ bits per symbol.} \end{aligned} \quad (4.29)$$

4.3.3 Relationship of Entropy to Ideal Code Length

A finite sequence of N symbols, where N is relatively large, is expected to have a self-information equal to N times the average information per symbol. If the sequence is encoded so that each symbol coded increases the binary code string by its self-information $i\ell(a_m)$ bits, then the coded sequence should be $N \times H(\mathcal{D})$ bits in length. The value $N \times H(\mathcal{D})$ bits is called the Ideal Code Length, denoted ICL, of the finite sequence.

Thus, another way of viewing entropy H is as the *ideal length* of a symbol sequence \mathcal{S} divided by the number N of symbols in the sequence. The entropy number is the ideal (best) per-symbol code length for compressing sequence \mathcal{S} . Similarly, a symbol a_m having probability $p(a_m)$ has self-information $i\ell(p(a_m)) = -\log_2 p(a_m)$ in bits, which is the self-information or the *ideal length* in bits for encoding the symbol a_m .

4.4 Compression Performance Measures by Symbol Count

4.4.1 Calculating the Ideal Code Length

In the previous section, the ideal code length, ICL, was introduced as a compression performance measure for data file $\mathcal{S} = s_1, s_2, \dots, s_k \dots s_N$. A first step to calculating the ICL, is to calculate the probability of data file \mathcal{S} as the product of symbol probabilities s_k comprising sequence \mathcal{S} :

$$p(\mathcal{S}) = \prod_{k=1}^N s_k \quad (4.30)$$

A sequence \mathcal{S} is also comprised of symbols a_m , each unique symbol of alphabet \mathcal{A} having its own probability $p(a_m)$ based on the frequency $Ct(a_m)$, divided by the number of symbols N in \mathcal{S} :

$$p(a_m) = \frac{Ct(a_m)}{N} \quad (4.31)$$

Using the probability values from Eq 4.31, we can also calculate the value $p(\mathcal{S})$ as:

$$p(\mathcal{S}) = \prod_{m=1}^M Ct(a_m) \times p(a_m) \quad (4.32)$$

Having calculated probability $p(\mathcal{S})$, an estimate of the ideal code length ICL in bits for file \mathcal{S} is:

$$ICL = -\log_2 p(\mathcal{S}). \quad (4.33)$$

Using the identity that the logarithm of the result of a product of factors is equivalent to summing the logarithm of each factor; the ideal code length (or ICL) in bits for \mathcal{S} is calculated a second way:

$$ICL = \sum_{k=1}^N -\log_2 p(s_k). \quad (4.34)$$

Eq 4.32 applies to higher-order compression models as well, *provided* that the probability value for each $p(s_k)$ is the same probability determined by the stochastic process by which file \mathcal{S} is modeled. This section (comprised of several subsections) expands on the notion expressed in Eq 4.32.

4.4.2 Practical Assumptions for ICL

Suppose we have a text data file \mathcal{S} composed of a symbol sequence drawn from the ASCII alphabet. What is the “entropy” of the file? Although a finite-length file violates a condition (i.e., file length N going to ∞) Shannon used in his proofs, we can determine a type of entropy for the file by making certain *practical assumptions*.

First, assume the sequence \mathcal{S} of symbols was produced by a *memoryless* source, i.e., the probability for the next symbol depends only on the single distribution. In other words, the symbols all come from the *same* (or identical) probability distribution. There is no “higher-order” finite state machine that remembers things like if a “q” is produced, that the probability that a “u” will follow is increased.

Although the assumption is unrealistic, we need to be aware of the consequences of any simplifying assumptions. Having assumed a single probability distribution (a table of symbols and their corresponding probabilities), we still don't know what the probabilities are.

Second, we assume the source symbol probabilities $p(a_m)$ are the relative frequencies we observe from data file \mathcal{S} itself. We can now calculate how long the string would be were we to replace each symbol by its *ideal length* in bits, and then sum the numbers representing each such length.

These estimates can be calculated using Eq 4.31 after counts are made, and Eq 4.32 for $p(\mathcal{S})$. The value of ICL can be calculated as $-\log_2 p(\mathcal{S})$.

Another calculation, given the result of Eq 4.31, is to read file \mathcal{S} , convert each symbol read to its probability $p(a_k)$, and use Eq 4.34 to compute the value of $ICL(\mathcal{S})$.

4.4.3 ICL for the Binary Alphabet

Suppose file \mathcal{S} contains only two symbol values, 0s and 1s, and that there are 128 symbol occurrences: 32 0s and 96 1s. Converting count ratios to *empirical probabilities*; $p(0) = 0.25$ and $p(1) = 0.75$. Using Eq 4.13, $H = 0.25 \log_2 4 + 0.75 \log_2 \frac{4}{3}$.

Note: See Eq 4.15, for the result $H(0.25, 0.75) = 0.8113$ bits per symbol.

The ideal length for 0, in positive form is: $il(0) = \log_2 \frac{128}{32}$. We eliminate the minus sign by using identity $-\log_2 \frac{1}{x} = \log_2 x$ that flips the sign by flipping the numerator and the denominator. Similarly, $il(1) = \log_2 \frac{128}{96}$. The ideal code length of this file is 32 instances of length $\log_2 \frac{128}{32}$ and 96 instances of length $\log_2 \frac{128}{96}$. Thus:

$$ICL(32, 96) = 32 \log_2 \frac{128}{32} + 96 \log_2 \frac{128}{96} \quad (4.35)$$

Converting the log of a fraction to a difference by the rule $\log \frac{x}{y} = \log x - \log y$, we now have:

$$ICL(32, 96) = 32 \log_2(128) - 32 \log_2(32) + 96 \log_2(128) - 96 \log_2(96) \quad (4.36)$$

Adding $32 \log_2(128)$ to $96 \log_2(128)$ forms the single term $128 \log_2(128)$. After all this work, this special case produced a result with *desireable symmetry* that shall be generalized:

$$ICL(32, 96) = 128 \log_2(128) - 32 \log_2(32) - 96 \log_2(96) \quad (4.37)$$

Substituting 7 for $\log_2(128)$, 5 for $\log_2 32$, and 6.58496 for $\log_2 96$ in Eq 4.37:

$$ICL(32, 96) = 896 - 160 - 632.156 = \underline{103.844} \text{ bits.} \quad (4.38)$$

The binary entropy $H_2(32, 96)$ for counts 32 and 96 is the per-symbol code length, and is obtained overall length ICL by the total number of symbols 128:

$$H_2(32, 96) = \log_2(128) - \frac{32}{128} \log_2(32) - \frac{96}{128} \log_2(96). \quad (4.39)$$

4.4.4 ICL: M-ary Alphabet

The so-called *ideal length* of a (data) string S^N , N symbols long, or the “*self-information* using the string \mathcal{S} 's own counts”, can be determined by a closed form formula that we call $N \log N$.

The simplest case, discussed first, is the memoryless model with a single probability distribution. In a first pass of data file \mathcal{S} , we accumulate the symbol counts $Ct(a_m)$ for each member a_m of the alphabet.

The $N \log N$ formula assigns to each symbol a_m of alphabet \mathcal{A} , where $m = 1, 2, \dots, M$, an ideal length $il(a_m)$, where the ideal length based on its relative frequency. Let $Ct(a_m)$ be obtained on a first pass of file \mathcal{S} , and $\sum_{m=1}^M Ct(a_m) = N$, since \mathcal{S} is comprised of N symbols. With N symbols, $p(a_m) = Ct(a_m)/N$.

$$ICL(\mathcal{S}) = \sum_{m=1}^M Ct(a_m) \times -\log_2 \frac{Ct(a_m)}{N}. \quad (4.40)$$

In Eq 4.40 has the form of an *inner product*: each value of m has a product, and each of the m products are summed. The factor to the right of the *times* (\times) *sign* is $il(a_m)$, the self-information of symbol a_m . We can simplify Eq 4.40 by converting $-\log_2 \frac{Ct(a_m)}{N}$ to $+\log_2 \frac{N}{Ct(a_m)}$, and then to: $\log_2 N - \log_2 Ct(a_m)$. The result is Eq 4.41.

$$ICL(\mathcal{S}) = \sum_{m=1}^M Ct(a_m) \times [\log_2 N - \log_2 Ct(a_m)] \quad (4.41)$$

Eq 4.41 has $Ct(a_m)$ multiplying two sums indexed by m , so we can break the equation into two summations:

$$ICL(S) = \sum_{m=1}^M Ct(a_m) \times [\log_2 N] - \sum_{m=1}^M Ct(a_m) \times \log_2 Ct(a_m) \quad (4.42)$$

The first summation just sums the symbol counts in S , which we know to be N , and we have: $N \log_2 N$. The second summation cannot be further simplified, and the final equation is:

$$N \log_2 N - \sum_{m=1}^M Ct(a_m) \times \log_2 Ct(a_m) \quad (4.43)$$

If we calculate the self-information for each symbol, $il(a_m)$ which is the result of the $\log_2()$ function, we can use Eq 4.44 to calculate the ICL:

$$ICL(S) = \sum_{m=1}^M Ct(a_m) \times il(a_m). \quad (4.44)$$

The factor to the left of the \times is the number of instances that self-information value $il(a_m)$ appeared in file S . There are M products to be *summed*, one for each symbol. The M individual products represent, respectively, the number of bits that each of the M symbols has contributed to the ideal code length $ICL(S)$ of the code string $Cd(S)$, and their sum is the value $ICL(S)$.

4.4.5 The Enumerative Code Length (ECL)

The previous section on the ICL assumes we *already know* the total counts for each symbol in the file. We present a method using the *Laplacian estimator* to estimate the probability for each symbol for an M -ary symbol alphabet. The technique is also known as “Laplace’s Law of Succession”, so named by the British mathematician Venn. The following description is based on Laplace’s 1814 treatise *Essai philosophique sur les probabilités*.

The basic idea is to determine the probability from counting outcomes from past observations. The example used by Laplace is the rising of the sun each morning. In Laplace’s day, people assumed that the Old Testament of the Bible, beginning with Adam and Eve, constituted the life time of the Earth. Laplace

placed the origin of time, hence the origin of the first sun rise, at approximately 5,000 years or 1,826,213 days earlier. Therefore, he reasoned as follows:

The probability the sun would rise again the next morning was 1,826,213/1,826,214. In other words, the chance that the sun would not rise the next morning was 1 out of 1,826,214.

Laplace also mentioned that his belief in laws that regulated days and seasons implies the probability the sun would continue to rise each morning was much closer to 1 than his example indicates.

The Laplacian estimator is initialized to a count of 1 for each symbol, and the total count initialized to value M . Thus the initial probability for each symbol is its *count ratio*: $\frac{1}{M}$. This technique is attributed to the French mathematician Pierre Simon, Marquis de Laplace, and a pioneer of probability theory.

A data file to be compressed is read one symbol at a time. After handling each next symbol value, that symbol’s count is incremented by one, as is the total count incremented by 1. Each symbol now has an updated probability via the new count ratios, which dynamically forms the estimates for the next symbol probability. We can convert the count ratio of the next symbol to its self-information (in bits) and dynamically keep a running sum of the self-information for the data file seen so far.

In a similar vein, an adaptive Huffman code has been devised that uses the dynamically generated counts using Laplace’s method, to maintain a code tree. When the counts suggest a change in the code tree, the system changes the Huffman codeword set. In arithmetic coding, one can use the counts directly in the coding operation.

Consider an alphabet \mathcal{A} , where the number of symbols $M = |\mathcal{A}|$ is 3 and $\mathcal{A} = \{a, b, c\}$ is the symbol set. Consider compressing string $S = a a a b a c a b b a b c$. In a one-pass technique the probability distribution is known before the next symbol is encoded.

The Laplacian estimator initializes the *counts* of each symbol to 1, even though none have been seen. This technique falls under the category of a “cumulative count” technique. The technique has been studied by John Cleary and Ian Witten in [CW 84], and identified as an *enumerative* technique, since we count instances.

The coding of the first symbol a should ideally, under the Laplacian estimator, increase the Enumerative (Code) Length (*ECL*) for data string S by value $\log_2 3$ (or $-\log_2 \frac{1}{3}$), since the count of a is 1 and the total count is 3. For the second a , the ratio is $\frac{2}{4}$, so the *ECL* code length increase for the Laplacian estimator is: $\log_2 2 = 1$ bit. Similarly, for the third a in a row,

the code length increase is $\log_2 \frac{5}{3}$.

Is it feasible to reassign probabilities for each next symbol of the data being encoded? Although in this example, a Huffman code would assign symbol **a** length 1, and symbols **b** and **c** length 2, an *arithmetic code*⁵ works in rational (non-integral) lengths and would assign a length as close to $\log_2 3$ as the precision of the number system allows. Also, arithmetic coding can use the probabilities directly in the encoding and decoding operations, so the probabilities can be changed before each new symbol without going through a Huffman coding algorithm.

Continuing the example, after encoding the symbol **a** three times at the beginning of symbol sequence \mathcal{S} , the total count is 6 (the 3 initial counts plus 3 more for the three symbols seen). The individual count is 4 for symbol **a** (the initial count of 1, plus the three times **a** was seen). The other individual symbol counts remain at 1 for **b**, and 1 for **c**. So when symbol **b** is seen next, the respective probability distribution is $\{\frac{2}{3}, \frac{1}{6}, \frac{1}{6}\}$. The first occurrence of symbol **b**, following its encoding, adds code length $\log_2 6$ to the code string. Then its individual count is incremented from 1 to 2, and the total count is incremented to 6.

In this way, we are determining probability distributions and code length increases “on the fly”, i.e., as we see symbols for the first time: a *one-pass* technique, or *fully adaptive* technique. There are two alternatives for determining the enumerative code length ECL of string \mathcal{S} under the Laplacian estimator *model* for obtaining the symbol statistics: (1) sum the code length increases by adding $\log_2 \frac{1}{p(a_m)}$ to the current sum when symbol $a_{(m)}$ of probability $p(a_m)$ is seen, or (2) count symbol instances as they occur and determine the probability of the string itself by a closed-form formula called *Logfac*.

Example. Rewrite the string as follows

$$\mathcal{S} = a^1 a^2 a^3 b^1 a^4 c^1 a^5 b^2 b^3 a^6 b^4 c^2. \quad (4.45)$$

The superscript i indicates the i^{th} time the symbol has been seen. If we consider the product of the probabilities of the first four symbols, we have $\frac{1}{3} \times \frac{2}{4} \times \frac{3}{5} \times \frac{1}{6}$. Notice that the denominator of each fraction increases by 1 from one symbol to the next. String \mathcal{S} has length $N = 12$ symbols, so the denominators run from 3, 4, ..., 14, and so the denominator for the fraction representing the probability of \mathcal{S} is proportional to $14!$. To be exact, the denominator is $\frac{(N+|A|-1)!}{(|A|-1)!}$, or in our case, $14! \div 2!$. Since our denominator is $3 \times 4 \times \dots \times 14$, when we replace this value by $14!$, we also need to “cancel” the factor of 2 in $14!$ that our denominator

⁵Treated in a future chapter.

does not have.

Next consider the numerator for $p(\mathcal{S})$, the probability of string \mathcal{S} under the Laplacian estimator for the one-pass statistics. Having handled the first three symbols of \mathcal{S} , (3 instances of symbol **a**), the numerator has progressed as: $1 \times 2 \times 3$, which represent the superscripts of the symbols shown in the example string \mathcal{S} above. Letting $Ct(\mathbf{a})$, $Ct(\mathbf{b})$, and $Ct(\mathbf{c})$ denote the total counts for symbols **a**, **b**, and **c**, the value of the numerator for $p(\mathcal{S})$ is the product $Ct(\mathbf{a})!Ct(\mathbf{b})!Ct(\mathbf{c})!$, which in the case of the example is: $6!4!2!$.

The *Logfac* formula for the general case of the Laplacian estimator provides the associated code length ECL for a memoryless model. Let alphabet \mathcal{A} contain M symbols: $\mathcal{A} = \{\mathbf{a}_1, \dots, \mathbf{a}_m, \dots, \mathbf{a}_M\}$.

$$ECL(\mathcal{S}) = \log_2[(N + M - 1)!] - \log_2[(M - 1)!] - \sum_{m=1}^M \log_2[Ct(\mathbf{a}_m)!] \quad (4.46)$$

where $Ct(\mathbf{a})$ is the number of times symbol **a** was seen. If we ignore the corrective term “ $-\log_2((M - 1)!)$ ”, which should be relatively small when total count N is much larger than M . Thus, the estimate is slightly higher, and hence is conservative.

If the zero-order (memoryless) model applies to a relatively short sequence \mathcal{S} , however, and the number of alphabet symbols M is of the same magnitude as the sequence length, then ignoring the corrective term may yield too large an estimate ECL.

Stirling’s formula for the log factorial function, $\log_{\text{fac}}(n) = \log_e n!$, is adjusted in Equation 4.47 for base-2 logarithms, $\log_2 n!$, and may be used to simplify the calculation for cases where the counts are large. The base of the *natural* logarithm, e , is used below in Eq 4.47.

$$\log_2 n! = \frac{1}{\log_e(2)} \times [(n + 0.5) \log_e(n) + 0.9189 - n] \quad (4.47)$$

The formal derivation of Stirling’s formula actually approximates $\log_e n!$, takes the result, and raises e to that power. Since $\log_e n!$ is the sum of the logs of $n, n - 1, \dots, 1$, the approximation integrates $\log_e x dx$ from 1 to n , then determines upper and lower bounds for the area under the $\log_e x$ curve.

In Stirling’s formula there is an approximation error because for each integer n , there is a small negative error in the formula that varies between $\frac{7}{8}$ and 1.0. However, as n tends to infinity, this term tends to $\frac{1}{2} \log_e 2\pi$, or 0.9189. Thus, the published versions

of Stirling's formula use this asymptotic limit as the correction term. See [Ham 80], pp 158–163.

Since Shannon's definition of source entropy of infinite sequences is based on asymptotic properties, i.e., the Law of Large Numbers, the value of $ECL(S)$ must converge to the ideal code length $ICL(S)$.

4.5 Codewords and the Kraft Length

With the binary alphabet $\{0, 1\}$ as the code alphabet, we can define the *code space* as the set of finite-length binary strings whose length does not exceed 10^{100} symbols, i.e., does not exceed the number of particles in the universe.

Data compression maps information in some source domain to the range of the code space such that, in the code space realm, fewer bits are used to represent the same data.

Ideally, data compression seeks to make the code string as short as possible; so why not encode each symbol with a single bit? The problem is decoding: there are only two binary strings of length 1: 0 and 1. We can encode an alphabet with one out of two choices, symbols $\{a, b\}$ with respective codewords 0 and 1, but we cannot encode a third symbol c without using the continuation space (positions behind the first 0 or 1) for the symbols assigned to 0 and 1. We can revoke the assignment of b to 1, and map b to 10 and c to 11, and our use of the code space is now satisfactory.

Thus, we face a *limitation* to the mapping of an *alphabet* to the *code space*. This limitation has been studied by Kraft, and the result is called the *Kraft Inequality*.

The rule called the *Kraft Inequality*, on the set of allowable code lengths $cl(a_m)$ for encoding symbols a_m , is quite simple for the binary code alphabet.

Observation 5 *Note: a value associated with notation cl , denoting code length, represents a non-ideal value for the ideal length that uses notation il .*

Each symbol a_m 's binary string length $cl(a_m)$ takes up code space $2^{-cl(a_m)}$, and the so-called *code space* cannot exceed unity. The number base is 2 for the length exponents denoted $-cl(a_m)$ because the binary code alphabet uses base 2.

Equation 4.48 defines the Kraft Inequality:

$$1 \geq \sum_{m=1}^M 2^{-cl(a_m)}, \quad (4.48)$$

where the right-hand side of the inequality is called the *Kraft Sum*.

Observation 6 *For Eq 4.48, the number of symbols in the code alphabet is 2.*

Were we to have a ternary (3-symbol) code alphabet, the Kraft sum to be satisfied is Eq 4.48 with value 2 replaced by 3:

$$1 \geq \sum_{m=1}^M 3^{-cl(a_m)}, \quad (4.49)$$

Observation 7 *Code alphabet sizes greater than 2 admit unused code words.*

The binary code tree has as many leaves as we need. Consider a ternary code tree. It can have 3 leaves. If one leaf is replaced by another 3-leaf tree, there are now 5 leaves. If a leaf on the 5-leaf tree is replaced by a 3-leaf tree, and we remove one of the 5 leaves and replace it with a 3 leaf tree, the net gain is 2 leaves. Now we have 7 leaves. If our symbol alphabet has 6 symbols, the ternary code word assigned to one of the leafs is never used.

For example, if the code alphabet has 4 symbols and the data alphabet has 5 symbols, then some leaves of the 4-symbol code tree are not used.

The reader can verify, by assigning a leaf of a binary tree corresponding to each length of a proposed set of lengths for a Kraft Sum, that if the inequality of Eq 4.48 is not met, then at least one symbol must share a common leaf of the tree and an "overlapping use" of the *code space* has occurred, and the code cannot be unambiguously decoded. In other words, at least one of the symbols will never be decoded. When the Kraft Sum is less than unity, there is some coding inefficiency since some "code space" is going unused: a code tree leaf is available to accommodate another symbol of the alphabet, but is not being used.

We now use code space arguments to demonstrate that the self-information is the ideal code length. Consider a memoryless source of known distribution, and a message consisting of a sequence of N symbols, denoted S^N , emitted from the source. To uniquely decode the N -symbol message, the decoder must distinguish "our" message from all others, thus the Kraft Inequality must be met.

Next we need the optimum way to split up the code space: *what portion of the full code space must each symbol receive?* To minimize the average code length (for best compression), we minimize the following sum: the code length $cl(a_m)$ assigned to each symbol a_m , weighted by its relative frequency $p(a_m)$.

For minimum average code length:

$$\sum_{m=1}^M p(a_m) \times cl(a_m). \quad (4.50)$$

For each symbol position j , $cl(s(j))$ is replaced by the code length increase assigned to the symbol $a(i)$ that occupies that position in S^N . The lengths $cl(a(i))$ must satisfy the Kraft Inequality Eq 4.48.

The Kraft Inequality just states a condition that allows decodability, but does not assign probabilities to those lengths. A related question is: if we know the probabilities, and could *choose* a length for each symbol that minimizes the code string length, *how do we calculate the optimal length values?* The answer: Shannon's self-information.

The minimization problem posed by Eq 4.50 above has been studied by Shannon, and by mathematicians in another context. If there are two probability distributions of M symbols, $p1(a_m)$ and $p2(a_m)$, then the inner product

$$-\sum_{m=1}^M p1(a_m) \times -\log_2 p2(a_m) \geq p1(a_m) \times -\log_2 p1(a_m) \quad (4.51)$$

is at a minimum when, for *all* m : $p(a_m) = q(a_m)$. What Shannon is pointing out is the *importance* of the notion of entropy relative to the definition of *self-information*. It means the self-information is special because it is a unary function of the probability value and will minimize the *code length* when used to encode data whose statistics or probability distribution is known.

In other words, if the inner product of Eq 4.51 on the left-hand side is not done with distribution \mathcal{D}_∞ 's self-information, we do not achieve the minimum possible code length.

Actually the logarithm can be to any base and the relationship " $p = q$ " for the minimum still holds. In [CT 91], Eq 4.51 is proved by Lagrangian multipliers. Note that the minimizing value set of $-\log_2 q(m)$ of Eq 4.51 are also called the ideal lengths: $il(m) = -\log_2 p(m)$.

A specialization of Jensen's Inequality, which relates in general to any *convex function*, focuses on logarithms. The logarithm is convex because as x increases, the derivative of $\log_2 x$ is negative so the path traced by $\log_2 x$ with increasing x curves to the right. We note that the set of numbers $\{2^{-\ell x}\}$ form a probability distribution (sum to one, no probability outside the range $\{0, 1\}$). This particular application and specialization, based on Jensen's Inequality, is sometimes called Gibb's Inequality.

This theorem applies to Information Theory by noting that value $2^{-il(a_m)}$ corresponds to value $p(a_m)$. Since the code designer decides the values of $il(a_m)$ for the code, we have the freedom to choose the lengths il assigned to each symbol so long as they satisfy the Kraft Inequality Eq 4.48.

Another way of looking at Eq 4.51, is to notice that the inner product to be minimized is the product of a probability from a first distribution $\mathcal{D}1$, where $\mathcal{D}1$ provides probabilities $p1(a_m)$, and the *self-information* il of $p2(a_m)$ comes from a second distribution $\mathcal{D}2$.

The weighted sum of the self-information $il1(a_m)$ from $\mathcal{D}1$ is at a minimum when, multiplied by its own value of $p1(a_m)$. Similarly, when $p2(a_m)$ and $il2(a_m)$ both come from distribution \mathcal{D}_∞ , then its weighted sum of self-information (i.e., its *entropy*) is also at its minimum, according to Gibbs.

The minimum average code length for data strings S^N from the set of symbols $\{a_1, \dots, a_m, \dots, a_M\}$ is attained for each symbol a_m when:

$$p(a_m) = 2^{-il(a_m)}; \quad (4.52)$$

or else (by taking logs), when:

$$il(a_m) = -\log_2 p(a_m). \quad (4.53)$$

Question 1 (The math is very interesting,)
but: "What good is it?"

Basically, the "good" comes from the nice feeling that there is an achievable best limit to compression. A mathematician may find that the notion of *self-information*, as defined by Shannon, is a "beautiful" concept. Many "pure" mathematicians enjoy mathematical beauty for its own sake.

Shannon proved mathematically that the very best you can do, for lossless data compression, is to encode data according to its symbol probabilities, using code length increases that correspond to the self-information. So engineers, applied mathematicians, and computer scientists enjoy "providing the best

product with the available resources (time, equipment, team members)” as a reward. Moreover, a coding technique called *arithmetic coding*, can use rational approximations to the *ideal length* il for very efficient coding relative to the entropy.

Prior to Shannon, people knew they needed to encode frequent symbols with short codes and infrequent symbols with longer codes, but Shannon quantified (or “nailed down”) the limiting values, given the constraint that the decoder at the receiver be able to exactly re-create the original data sequence given the lossless transmission of the code string.

Since $\sum_m p(a_m)$ sums to 1, then $\sum_m 2^{-il(a_m)}$ also sums to 1, since the terms in each sum are the same: the base-2 antilog of $-il(a_m)$ is $p(a_m)$. Therefore, if the set of lengths $cl(a_m)$ that satisfy the Kraft Inequality are equal to the self-information lengths $il(a_m)$ then the mean per-symbol of the code string $\sum_m p(a_m) \times cl(a_m)$ may be as close to the theoretical minimum as the precision of the approximation to $cl(a_m)$ permits.

4.6 Independence and the Memoryless Source

One important assumption for the memoryless source is that the symbol sequence emitted from the source demonstrates *independence* from one symbol to the next. An implication of a memoryless source is that a sequence S^N so emitted has a probability whose value is the product of the probabilities of each symbol that appears in the sequence.

Consider the case where the product of the probabilities of the symbols comprising N-symbol sequence S^N is expressed in the form 2^{-il} , where il is the self-information. Then to take up exactly that 2^{-il} of the code space, we would need a binary string of length il . If we view il as having an integer part and fractional part, we would need perhaps one bit more than the fractional part to encode the message.

4.7 Efficiency Calculations

Consider file S of length N , and the Huffman algorithm applied to alphabet \mathcal{A} according to the symbol frequencies $Ct(a_m)$.

Question 2 *How do we determine the efficiency (and/or the inefficiency) of our Huffman code?*

The basic idea is to calculate the *ideal* length, $IL(S)$ which is based on the empirical entropy $H_e(S)$, in turn based on the *empirical probabilities*, $\frac{Ct(a_m)}{N}$, calculated from the symbol frequency histogram of S . We consider $H_e(S)$, calculated in bits, as the *ideal*. For comparison, we can calculate the actual *code length* $CL(S)$ by associating the code word length, $cl(a_m)$, with its symbol count $Ct(a_m)$.

$$CL(S) = \sum_{m=1}^M Ct(a_m) \times cl(a_m) \quad (4.54)$$

Observation 8 *The mean (average) per-symbol code length of compressed file $CL(S)$, of N coded symbols, may be calculated by summing each symbol’s (a_m ’s) code word length $cl(a_m)$.*

$CL(S)$ is measured in *bits* when the code word length, $cl(a_m)$, is given in *bits*.

Given a Huffman code applied to data file S of length N , and the set of empirical probabilities $p(a_m) = \frac{Ct(a_m)}{N}$ from file S , let H_e denote the *empirical* entropy of the data.

Let $CL(S)$ be the Huffman code length. Under these conditions, the *coding inefficiency* (CI) is calculated as follows:

$$CI = \frac{CL/N - H_e}{H_e} \quad (4.55)$$

In the following, we calculate the entropy H for a given probability distribution \mathcal{D} . The set of probabilities p for \mathcal{D} are also used for the Huffman algorithm to generate a code tree.

Example 4 *Consider the following set of probabilities for distribution \mathcal{D}_{Ex1} : $\{\frac{24}{54}, \frac{9}{54}, \frac{8}{54}, \frac{6}{54}, \frac{6}{54}, \frac{1}{54}\}$.*

Example 4 above is taken from the Huffman coding chapter, section **Example Designs of Huffman Codes**, first example. The equation for entropy H below uses the probabilities of Example 4 for the entropy calculation.

We use the form of each term as $p \log_2 \frac{1}{p}$. In the calculation below, the first factor p is represented in common denominator form: $\frac{n}{54}$. In the second factor, the fraction $\frac{1}{p}$ is represented as $\frac{m}{n}$ with any common factors of n and m removed. See the fractions in the distribution \mathcal{D} of Example 4 above, before changing to the least common denominator.

From Eq 4.55, we have;

$$\begin{aligned} \mathcal{D}_{Ex1} &= \frac{24}{54} \log_2 \frac{9}{4} + \frac{9}{54} \log_2 6 \\ &+ \frac{8}{54} \log_2 \frac{27}{4} + 2 \times \frac{6}{54} \log_2 9 \\ &+ \frac{1}{54} \log_2 54. = \underline{2.170}. \end{aligned} \quad (4.56)$$

From Chapter Three, the *average code length* of the Huffman code for \mathcal{D}_{Ex1} is:

$$\frac{1}{54} \times (54 + 30 + 17 + 13 + 7) = \frac{121}{54} = \underline{2.241} \text{bits}$$

The coding inefficiency for \mathcal{D}_{Ex1} , using Eq 4.55, is: $\frac{2.241 - 2.170}{2.170} = \underline{0.0327}$.

Next, consider the Huffman code of Example 2 from Chapter 3 that has the following distribution. $\mathcal{D}_{Ex2} = \{\frac{12}{48}, \frac{9}{48}, \frac{8}{48}, \frac{6}{48}, \frac{4}{48}, \frac{3}{48}, \frac{2}{48}, \frac{2}{48}, \frac{2}{48}\}$.

The *entropy* calculation H_{Ex2} is:

$$\begin{aligned} H_{Ex2} &= \frac{12}{48} \log_2 4 + \frac{9}{48} \log_2 \frac{16}{3} \\ &+ \frac{8}{48} \log_2 6 + \frac{6}{48} \log_2 8 \\ &+ \frac{4}{48} \log_2 12 + \frac{3}{48} \log_2 16 + 3 \times \frac{2}{48} \log_2 24 \\ &= \underline{2.881}. \end{aligned} \quad (4.57)$$

From the Huffman code for \mathcal{D}_{Ex2} of Chapter 3, the average (per-symbol) code length is 2.917 bits per symbol.

Coding inefficiency CI is covered in Eq 4.55. Having just calculated 2.881 bps as the ideal, the coding inefficiency for the Huffman code of Example 2 of Chapter 3 is:

$$CI = \frac{2.917 - 2.881}{2.881} = \underline{0.0125}. \quad (4.58)$$

Consequently, for this example, the Huffman code is not bad: only 1.15% worse than the ideal.

4.8 Coding to the Entropy

It is not accidental that in the previous (and in mostly all other such) examples that the minimum possible average code length is not less than the entropy. It is possible to contrive the probabilities such that we “code to the entropy”, where the average per-symbol code length of a Huffman code is *equal* to the entropy.

Notice that the entropy H is the inner product of the symbol probability vector with the symbol self-information vector. On the other hand, the average code length is the inner product of the symbol probability vector with the vector of the symbol codeword lengths.

Observation 9 *Here’s the “key” to probability distributions that yield “coding to the entropy” for Huffman code words.*

The calculations for the entropy H , and the *average code length* are the same when the *codeword length* for a_m , $el(a_m)$ equals the self-information $il(a_m)$ for each symbol a_m .

Huffman codeword lengths are integers, therefore only probability distributions that produce integer values for $il(p)$ can be “coded to the entropy”. So any distribution, all of whose probabilities *are of the form* 2^{-i} , where i is some integer, have Huffman code tables whose average code length per symbol and entropy H are the same value.

4.9 Stationary and Nonstationary Sources

A sequence of data values drawn from a *stationary* information source is a common assumption in information theory. The source is stationary if one examines relatively long subsequences of data and the probabilities of each symbol are relatively close to their long-term average values. A stationary sequence, then, means that one lacks sufficient evidence that the “underlying” probability distribution from which the symbols are drawn has changed.

Although the concept of an imaginary information source enables one to state and prove theorems, the reality of data compression must deal with an individual sequence in itself. Either a data sequence S exhibits stationarity or not. We need to deal with practical options and assumptions. We can always assume stationarity by learning the zero-order symbol frequencies

through counting techniques, and using those to compress the data file. If most of the probabilities $p(a_m)$ of the M symbols are close to value $\frac{1}{M}$, then at least some compression will be achieved.

On the other hand, if a data sequence exhibits a *change* in the relative frequencies of the symbols for a sufficiently long subsequence of data file \mathcal{S} , then data file \mathcal{S} exhibits *non-stationary* behavior. Changes in the relative frequencies of symbols within file \mathcal{S} indicate the file could possibly be compressed even further with the following techniques:

- Use an adaptive probability estimation technique that works by counting symbol instances and “forgetting” the older counts. In a sense, use the analogy of a sliding window into the past history of the counts.
- Devise a higher-order compression model to generate context-dependent probability distributions based upon the already encoded neighboring values of the next event to be encoded (decoded).

These topics are covered in greater depth in subsequent chapters.

Appendix I. Refresher on logarithms

The self-information is defined in terms of taking logarithms to the base 2. In working with formulae that include the log function, several identities are useful. Let x , y , and z be real numbers, and e be the base of the natural logarithms. The *value* of base e is 2.71828...

$$y = 2^x \text{ implies } \log_2 x = y.$$

$$\text{If } \log_x y = z, \text{ Then } y = x^z.$$

$$\text{If } \log_2 x = 3, \text{ what is } x? \quad x = 8, \text{ since } 8 = 2^3.$$

$$\log 1 = 0. \quad \log 0 = -\infty.$$

$$\log(x/y) = \log x - \log y.$$

$$\log(1/y) = \log 1 - \log y = 0 - \log y = -\log y.$$

$$\log(x \times y) = \log x + \log y.$$

$$x \text{ positive; } x^1 \text{ yields } x, \text{ so } \log_x x = 1;$$

To calculate $\log_y(x)$ (logarithm of x using base y) when only the natural log is available on our hand calculator, use the following identity:

$$\log_y x = \frac{\log_e x}{\log_e y}$$

Consider number bases 2 and 256 for base y , with x as value 0.5. The values of \log_e of 2 and 256 are respectively (to 6 digits) 0.693147 and 5.54518. Below we calculate the self-information to base 2 (bits) and base 256 (bytes) of an event of probability $\frac{1}{2}$, where $\log_e 0.5 = -0.693147$. Note that negative factor -0.693147 cancels the *minus* sign that begins each of the two equations, yielding a positive result.

$$-\log_2(0.5) = -\frac{\log_e 0.5}{0.69315} = \frac{0.693147}{0.693147} = 1.0. \quad (4.59)$$

$$-\log_{256}(0.5) = \frac{\log_e 0.5}{5.5452} = \frac{0.693147}{5.54518} = 0.125 (= \frac{1}{8}) \quad (4.60)$$

Note that \log_2 gives a result in *bits* and \log_{256} gives a result in bytes. The self-information of $\frac{1}{2}$ is thus 1 bit or $\frac{1}{8}$ byte, which is the same.

Appendix II. Examples of Self-Information

This appendix presents Tables A.1 and A.2. *Table A.1* presents a table of examples, where 2^n means $2^{(n)}$, or 2 raised to the power n , and \sim means “approximately”.

Table A.1. Probability values related to some rational self-information values

a priori probability	Self-information (bits)
1.000	0
$2^{(-0.01)} \sim 0.993$	1/100
$2^{(-1/20)} \sim 0.966$	1/20
$2^{(-1/10)} \sim 0.933$	1/10

$2^{-1/8}$	~ 0.917		1/8
$2^{-1/6}$	~ 0.891		1/6
$2^{-1/5}$	~ 0.871		1/5
$2^{-1/4}$	~ 0.841		1/4
$2^{-1/3}$	~ 0.794		1/3
$2^{-1/2}$	~ 0.707		1/2
1/4			2
1/8			3
1/16			4
1/32			5
1/64			6
1/256			8
$1/1024=1/2^{10}$			10
$1/2^{1/20}$			20
$1/2^{1/100}$			100
0			Infinity

Note that the less likely the event, the larger the number representing the information. The logarithmic measure assures that when a succession of independent events occurs the total information received is the sum of the measures of information for the individual events. Table A.1 uses *probabilities* that are *powers of two*, so one can easily see the logarithmic relationship of information in bits and the powers of two.

In Table A.2, the value of self-information for other rational numbers are tabulated to four fractional decimal places.

Table A.2. Self-information values related to some rational probabilities

a priori probability	Self-information (bits)
99/100	0.0145
19/20	0.0740
9/10	0.1520
7/8	0.1926
5/6	0.2630
4/5	0.3219
3/4	0.4150
2/3	0.5850
1/3	1.5850
1/4	2.0000
1/5	2.3219
1/6	2.5850
1/8	3.0000
1/10	3.3219
1/20	4.3219
1/100	6.6439

4.10 Problems

- Calculate the entropy of these binary sources, $\mathcal{A} = \{0, 1\}$, whose smaller probability $p(0)$ is:
 - $\frac{1}{16}$.
 - $\frac{1}{8}$.
 - $\frac{1}{4}$.
 - $\frac{1}{2}$.
- In the previous problem, comment on the rate of increase of the entropy H as the probability of the less probable symbol increases.
- Consider a source coding approach for binary source alphabet $\{0, 1\}$, where $M = 2$. Unlike prefix codes, *arithmetic coding* permits code lengths $cl(0)$ and $cl(1)$ to be rational. Probability estimation techniques result in rational number values for the estimated probabilities p . Unless p is a negative power of the logarithm base, its *ideal length* value $il(0) = \log_2 \frac{1}{p(0)}$ is irrational (transcendental).

The Problem to Answer:

If $il(0)$ is a transcendental number, as in $\log_2 \frac{16}{15}$ for the larger binary probability of part 1a of problem 1, practical applications require the code length $cl(0)$ to be either a truncation or rounding of $il(0)$ for two reasons: (1) to convert the number to a rational, and (2) while ensuring that the Kraft Inequality is satisfied.

Which of the two alternatives (1) *truncate* or (2) *round* would you perform? Justify your choice.

Bibliography

- [Abr 63] N. Abramson, *Information Theory and Coding*, McGraw-Hill Book Co, New York, 1963.
- [CW 84] John G. Cleary and Ian H. Witten, "Data Compression using Adaptive Coding and Partial String Matching", *IEEE Trans. Commun.*, vol COM-32, 396–402, April 1984.
- [CT 91] T. Cover and J. Thomas, *Information Theory and Coding*, McGraw-Hill Book Co., John Wiley, New York, 1991.
- [Fan 61] Robert M. Fano, *Transmission of Information*, MIT Press, Cambridge MA, 1961.
- [Gil 52] E. N. Gilbert, "A Comparison of Signalling Alphabets", *Bell Systems Technical Journal*, vo 31, 504–522, May 1952.
- [Ham 80] Richard Hamming, *Coding and Information Theory*, Prentice-Hall, 1980.
- [Huf 52] David Huffman, "A method for the construction of minimum redundancy codes", *Proc. IRE*, vol 40, 1098–1101, Sept 1952.
- [Kul 83] S. Kullback, "Kullback Information", in *Encyclopedia of Statistical Sciences*, vol 4, John Wiley, New York, 421–425, 1983
- [McM 53] Brockway McMillan, "The Basic Theorems of Information Theory", *Annals of Mathematical Statistics*, vol 24, 196–219, June 1953.
- [Sha 48] C. Shannon, "A Mathematical Theory of Communication", *Bell System Technical Journal*, vol. 27, July 1948, 379-423.
- [Wie 48] Norbert Wiener, *Cybernetics*, John Wiley, New York, 1948.
- [Wie 49] Norbert Wiener, *Extrapolation, Interpolation, and Smoothing of Stationary Time Series*, John Wiley, New York, 1949.