1 Introduction to information theory

1.1 Introduction

In this chapter we present some of the basic concepts of information theory. The situations we have in mind involve the exchange of information through transmission and storage media designed for that purpose. The information is represented in digital formats using symbols and alphabets taken in a very broad sense. The deliberate choice of the way information is represented, often referred to as coding, is an essential aspect of the theory, and for many results it will be assumed that the user is free to choose a suitable code.

We present the classical results of information theory, which were originally developed as a model of communication as it takes place over a telephone circuit or a similar connection. However, we shall pay particular attention to two-dimensional (2-D) applications, and many examples are chosen from these areas. A huge amount of information is exchanged in formats that are essentially 2-D, namely web-pages, graphic material, etc. Such forms of communication typically have an extremely complex structure. The term media is often used to indicate the structure of the message as well as the surrounding organization. Information theory is relevant for understanding the possibilities and limitation of many aspects of 2-D media and one should not expect to be able to model and analyze all aspects within a single approach.

1.2 Entropy of discrete sources

A *discrete information source* is a device or process that outputs symbols at discrete instances from some finite alphabet $\mathcal{A} = \{x_1, x_2, \ldots, x_r\}$. We usually assume that a large number of such symbols will have to be processed, and that they may be produced at regular instances in time or read from various positions in the plane in some fashion. The use of a large number of symbols allows us to describe the frequencies in terms of a probability distribution.

A single symbol can be thought of as a random variable, X, with values from the finite alphabet $\mathcal{A}$. We assume that all variables have the same probability distribution, and the probability of x_i is written as $P(x_i)$ or $P[x = x_i]$. It is often convenient to refer to the probability distribution, $P(X)$, of the variable as a vector of probabilities $\mathbf{p}_X$, where the values are listed in some fixed order.

As a measure of the amount of information provided by X we introduce the entropy of a random variable.

Definition. The *entropy* of the variable X is

$$H(X) = E[\log(1/P(X))] = -\sum P(x)\log P(x), \tag{1.1}$$

where $E[\,]$ indicates the expected value.

The word entropy (as well as the use of the letter H) has its historic origin in statistical mechanics. The choice of the base for the logarithms is a matter of convention, but we shall always use binary logarithms and refer to the amount of information as measured in *bits*.

We note that

$$0 \le H(X) \le \log r.$$

The entropy is always positive since each term is positive. The maximal value of $H(X)$ is reached when each value of X has probability $1/r$ and the minimal when one outcome has probability 1 (See Exercise 1.1). When $r = 2^m$, the variable provides at most m bits of information, which shows that the word bit in information theory is used in a way that differs slightly from common technical terminology. If X^N represents a string of N symbols, each with the same alphabet and probability distribution, it follows from (1.1) that the maximal entropy is NH, and that this maximum is reached when the variables are independent. We refer to a source for which the outputs are independent as a *memoryless source*, and if the probability distributions also are identical we describe it as i.i.d. (independent identically distributed). In this case we say that the entropy of the source is H bits/symbol.

Example 1.1 (Entropy of a binary source). *For a binary memoryless source with probability distribution $P = (p, 1-p)$ we get the entropy (1.1) as*

$$H(p) = -p\log p - (1-p)\log(1-p), \tag{1.2}$$

where $H(p)$ is called the (binary) entropy function. The entropy function is symmetric with respect to $p = 1/2$, and reaches its maximal value, 1, there. For small p, H increases quickly with increasing p, and $H(0.11) = 0.5$.

We note that the same letter, H, is used to indicate the entropy of a random variable and the value of the entropy function for a particular distribution, $(p, 1-p)$. We shall use the same notation $H(P)$ when P is a general discrete probability distribution.

Example 1.2 (Run-length coding). *For a binary source with a small value of $P(1)$ it may be useful to segment a string of symbols into "runs" of 0s ending with a 1. In this way the source is converted into a memoryless source with alphabet $\{0, 1, 2, \ldots, n\}$, where the symbols indicate the length of the segment. The value k represents the string 0^k1, $k < n$. If runs of more than n 0s are possible, the value $k = n$ represents the string*

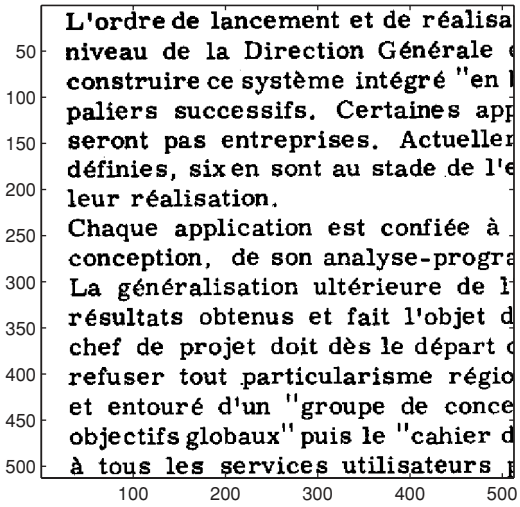

Figure 1.1. A binary image of a printed text.

0^n, *i.e. a run of 0s that is not terminated. Run-length coding may be used to code binary images of text along the rows (Fig. 1.1).*

For a pair of random variables, (X, Y), with values from finite alphabets $\mathcal{A}_x = \{x_1, x_2, \ldots, x_r\}$ and $\mathcal{A}_y = \{y_1, y_2, \ldots, y_s\}$, the conditional probability of y_j given x_i is $P(y_j|x_i) = q_{ij}$. It is often convenient to refer to the conditional probability distribution as the matrix $\mathbf{Q} = P(Y|X) = [q_{ij}]$. The probability distribution of the variable, Y, then becomes

$$\mathbf{p}_Y = \mathbf{p}_X \mathbf{Q},$$

where $\mathbf{p}_X$ and $\mathbf{p}_Y$ are the distributions of X and Y, respectively.

Definition. The *conditional entropy*

$$\begin{aligned} H(Y|X) &= -\sum_{i,j} P(x_i, y_j) \log P(y_j|x_i) \\ &= -\sum_i P(x_i) \sum_j P(y_j|x_i) \log P(y_j|x_i) \end{aligned} \tag{1.3}$$

can be interpreted as the additional information provided by Y when X is known.

For any particular y_j, the average value of $P(y_j|x_i)\log P(y_j|x_i)$ is given by $-P(y_j)\log P(y_j)$. Since the function $-u \log u$ is concave (has negative second derivative), we can use Jensen's inequality, which states that for such a function

$$f(E(u)) \geq E[f(u)].$$

Thus, on applying Jensen's inequality with $u = p(y|x)$ to (1.3), it follows that the entropy is never increased by conditioning, i.e. $H(Y) \geq H(Y|X)$.

For a string of variables, $X_1, X_2, \ldots, X_n$ we have

$$P(X_1, X_2, \ldots, X_n) = P(X_1)P(X_2|X_1)P(X_3|X_1, X_2)\ldots P(X_n|X_1, X_2, \ldots, X_{n-1})$$

and thus, taking the expectation of the logarithm,

$$H(X_1, X_2, \ldots, X_n) = H(X_1) + H(X_2|X_1) + \cdots + H(X_n|X_1, X_2, \ldots, X_{n-1}). \tag{1.4}$$

This result is known as the *chain rule*, and expresses the total amount of information as a sum of information contributions from variable X_i when values of the previous variables are known. When the variables are not independent, the chain rule is often a valuable tool with which to decompose the probabilities and entropies of the entire string of data into terms that are easier to analyze and calculate.

Example 1.3 (Entropy by the chain rule). *If X takes values in $\mathcal{A} = \{x_1, x_2, x_3, x_4\}$ with probability distribution $(1/4, 1/4, 1/3, 1/6)$, we can of course calculate the entropy directly from the definition. As an alternative, we can think of the outcome as a result of first choosing a subset of two values, represented by the variable Y. We can then express the entropy by the chain rule:*

$$H(X) = H(Y) + H(X|Y) = H(1/2) + (1/2)H(1/2) + (1/2)H(1/3)$$
$$= 3/2 + (\log 3 - 2/3)/2 = 7/6 + (1/2)\log 3.$$

1.3 Mutual information and discrete memoryless channels

For a pair of random variables, (X, Y), with values from finite alphabets $\mathcal{A}_x = \{x_1, x_2, \ldots, x_r\}$ and $\mathcal{A}_y = \{y_1, y_2, \ldots, y_s\}$, the *mutual information* is defined as

$$I(Y; X) = \sum_{x,y} P(x, y)\log\left(\frac{P(y|x)}{P(y)}\right) = E\left[\log\left(\frac{P(y|x)}{P(y)}\right)\right]. \tag{1.5}$$

The mutual information is a measure of the amount of information about X represented by Y. It follows from the definition that the mutual information can be expressed in terms of entropies as

$$I(X; Y) = H(Y) - H(Y|X) = H(X) - H(X|Y). \tag{1.6}$$

The properties of the conditional entropy imply that the mutual information $I(X; Y) \geq 0$. It is 0 when X and Y are independent, and achieves the maximal value $H(X)$ when each value of X gives a unique value of Y.

It may also be noted that $I(Y; X) = I(X; Y)$. The symmetry follows from rewriting $P(y|x)$ as $P(y, x)/P(x)$ in the definition of I. This is an unexpected property since there is often a causal relation between X and Y, and it is not possible to interchange cause and effect.

From the chain rule for entropy we get

$$I(Y; X_1, X_2, \ldots, X_n) = I(Y; X_1) + I(Y; X_2|X_1) + \cdots + I(Y; X_n|X_1, X_2, \ldots, X_{n-1}), \quad (1.7)$$

which we refer to as the *chain rule for mutual information*.

A discrete information channel is a model of communication where X and Y represent the input and output variables, and the channel is defined by the conditional probability matrix, $\mathbf{Q} = P(Y|X) = [q_{ij}]$, which in this context is also called the transition matrix of the channel. When the input distribution $P(X)$ is given, we find the output distribution as

$$\mathbf{p}_Y = \mathbf{p}_X \mathbf{Q}.$$

Example 1.4 (The binary symmetric channel). *The single most important channel model is the binary symmetric channel (BSC). This channel models a situation in which random errors occur with probability p in binary data. For equally distributed inputs, the output has the same distribution, and the mutual information may be found from (1.6) as*

$$C = I(Y; X) = 1 - H(p), \quad (1.8)$$

where H again indicates the binary entropy function. Thus $C = 1$ for $p = 0$ or 1, C has minimal value 0 for $p = 1/2$, and $C = 1/2$ for $p = 0.11$.

For two variables we can combine (1.4) and (1.6) to get

$$H(X, Y) = H(X) + H(Y) - I(X; Y).$$

Mutual information cannot be increased by data processing. If Z is a variable that is obtained by processing Y, but depends on X only through Y, we have from (1.7)

$$I(X; Z, Y) = I(X; Y) + I(X; Z|Y) = I(X; Z) + I(X; Y|Z).$$

Here $I(X; Z|Y) = 0$, since X and Z are independent given Y, and $I(X; Y|Z) \geq 0$. Thus

$$I(X; Z) \leq I(X; Y). \quad (1.9)$$

1.4 Source coding for discrete sources

In this section we demonstrate that the entropy of a message has a more operational significance: if the entropy of the source is H, a message in the form of a string of N symbols can, by suitable coding, be represented by about NH binary symbols. We refer to such a process as *source coding*.

1.4.1 The Kraft inequality

We shall prove the existence of efficient source codes by actually constructing some codes that are important in applications. However, getting to these results requires some intermediate steps.

A binary variable-length source code is described as a mapping from the source alphabet $\mathcal{A}$ to a set of finite strings, C from the binary code alphabet, which we always denote $\{0, 1\}$. Since we allow the strings in the code to have different lengths, it is important that we can carry out the reverse mapping in a unique way. A simple way of ensuring this property is to use a *prefix code*, a set of strings chosen in such a way that no string is also the beginning (prefix) of another string. Thus, when the current string belongs to C, we know that we have reached the end, and we can start processing the following symbols as a new code string. In Example 1.5 an example of a simple prefix code is given.

If c_i is a string in C and $l(c_i)$ its length in binary symbols, the *expected length* of the source code per source symbol is

$$L(C) = \sum_{i=1}^{N} P(c_i) l(c_i).$$

If the set of lengths of the code is $\{l(c_i)\}$, any prefix code must satisfy the following important condition, known as the *Kraft inequality*:

$$\sum_i 2^{-l(c_i)} \leq 1. \tag{1.10}$$

The code can be described as a binary search tree: starting from the root, two branches are labelled 0 and 1, and each node is either a leaf that corresponds to the end of a string, or a node that can be assumed to have two continuing branches. Let l_m be the maximal length of a string. If a string has length $l(c)$, it follows from the prefix condition that none of the $2^{l_m - l(c)}$ extensions of this string are in the code. Also, two extensions of different code strings are never equal, since this would violate the prefix condition. Thus by summing over all codewords we get

$$\sum_i 2^{l_m - l(c_i)} \leq 2^{l_m}$$

and the inequality follows. It may further be proven that any uniquely decodable code must satisfy (1.10) and that if this is the case there exists a prefix code with the same set of code lengths. Thus restriction to prefix codes imposes no loss in coding performance.

Example 1.5 (A simple code). *The code* $\{0, 10, 110, 111\}$ *is a prefix code for an alphabet of four symbols. If the probability distribution of the source is (1/2, 1/4, 1/8, 1/8), the average length of the code strings is* $1 \times 1/2 + 2 \times 1/4 + 3 \times 1/4 = 7/4$*, which is also the entropy of the source.*

If all the numbers $-\log P(c_i)$ were integers, we could choose these as the lengths $l(c_i)$. In this way the Kraft inequality would be satisfied with equality, and furthermore

$$L = \sum_i P(c_i) l(c_i) = -\sum_i P(c_i) \log P(c_i) = H(X)$$

and thus the expected code length would equal the entropy. Such a case is shown in Example 1.5. However, in general we have to select code strings that only approximate the optimal values. If we round $-\log P(c_i)$ to the nearest larger integer $\lceil -\log P(c_i) \rceil$, the lengths satisfy the Kraft inequality, and by summing we get an upper bound on the code lengths

$$l(c_i) = \lceil -\log P(c_i) \rceil \leq -\log P(c_i) + 1. \tag{1.11}$$

The difference between the entropy and the average code length may be evaluated from

$$\begin{aligned} H(X) - L &= \sum_i P(c_i) \left[\log\left(\frac{1}{P(c_i)} \right) - l_i \right] = \sum_i P(c_i) \log\left(\frac{2^{-l_i}}{P(c_i)} \right) \\ &\leq \log \sum_i 2^{-l_i} \leq 0, \end{aligned}$$

where the inequalities are those established by Jensen and Kraft, respectively. This gives

$$H(X) \leq L \leq H(X) + 1, \tag{1.12}$$

where the right-hand side is given by taking the average of (1.11).

The loss due to the integer rounding may give a disappointing result when the coding is done on single source symbols. However, if we apply the result to strings of N symbols, we find an expected code length of at most $NH + 1$, and the result per source symbol becomes at most $H + 1/N$. Thus, for sources with independent symbols, we can get an expected code length close to the entropy by encoding sufficiently long strings of source symbols.

1.4.2 Huffman coding

In this section we present a construction of a variable-length source code with minimal expected code length. This method, *Huffman coding*, is a commonly used coding technique. It is based on the following simple recursive procedure for binary codewords.

(1) Input: a list of symbols, S, and their probability distribution.
(2) Let the two smallest probabilities be p_i and p_j (maybe not unique).
(3) Assign these two symbols the same code string, α, followed by 0 and 1, respectively.
(4) Merge them into a single symbol, a_{ij}, with probability $p(a_{ij}) = p_i + p_j$.
(5) Repeat from Step 1 until only a single symbol is left.

The result of the procedure can be interpreted as a decision tree, where the code string is obtained as the labels on the branches that lead to a leaf representing the symbol. It

Table 1.1. Huffman coding (S denotes symbols and the two bits determined in each stage are indicated by italics)

Stage 1 (five symbols)			Stage 2 (four Symbols)			Stage 3 (three symbols)			Stage 4 (two symbols)		
Prob	S	C	Prob	S	C	Prob	S	C	Prob	S	C
0.50	a_1	0	0.50	a_1	0	0.50	a_1	0	0.50	a_1	*0*
0.17	a_2	100	0.19	a_{45}	11	0.31	a_{23}	1*0*	0.50	a_{2-5}	*1*
0.14	a_3	101	0.17	a_2	10*0*	0.19	a_{45}	1*1*			
0.12	a_4	11*0*	0.14	a_3	10*1*						
0.07	a_5	11*1*									

follows immediately from the description that the code is a prefix code. It may be noted that the algorithm constructs the tree by going from the leaves to the root.

To see that the Huffman code has minimal expected length, we first note that in any optimal prefix code there are two strings with maximal length differing in the last bit. These strings are code strings for the symbols with the smallest probabilities (Steps 1 and 2). If the symbols with the longest strings did not have minimal probabilities, we could switch the strings and get a smaller expected length. If the longest string did not have a companion that differs only in the last bit, we could eliminate the last bit, use this one-bit-shorter codeword and preserve the prefix property, thus reducing the expected length.

The expected length can be expressed as

$$L = L' + p_i + p_j,$$

where L' is the expected length of the code after the two symbols have been merged, and the last terms were chosen to be minimal. Thus the reduced code must be chosen as an optimal code for the reduced alphabet with α (Step 3) and the same arguments may be repeated for this (Step 4). Thus, the steps of Huffman coding are consistent with the structure of an optimal prefix code. Further, the code is indeed optimal, insofar as restricting the class of codes to prefix codes gives no loss in coding performance, as previously remarked.

It follows from the optimality of the Huffman code that the expected code length satisfies (1.12).

An example of the Huffman code procedure is given in Table 1.1. For convenience the symbols are ordered by decreasing probabilities.

By extending the alphabet and code N symbols at a time, an average code length per original symbol arbitrarily close to the entropy may be achieved. The price is that the number of codewords increases exponentially in N.

1.5 Arithmetic codes

Arithmetic coding can code a sequence of symbols, x^n, using close to $-\log_2 p(x^n)$ bits, in accordance with the entropy, with a complexity which is only linear in n.

This is achieved by processing one source symbol at a time. Furthermore, arithmetic coding may immediately be combined with adaptive (dynamically changing) estimates of probabilities.

Arithmetic coding is often referred to as coding by interval subdivision. The coding specifies a number (pointing) within a subinterval of the unit interval $[0; 1[$, where the lower end point is closed and the upper end point is open. The codeword identifying the subinterval, and thereby the sequence that is coded, is a binary representation of this number.

1.5.1 Ideal arithmetic coding

Let x^n denote a sequence of symbols to be coded. $x^n = x_1 \dots x_n$ has length n. Let x_i be the symbol at i. Let $P(x^n)$ denote a probability measure of x^n. At this point we disregard the restriction of finite precision of our computer, but this issue is discussed in the following sections. We shall associate the sequence of symbols x^n with an interval $[C(x^n); C(x^n) + P(x^n)[$. The lower bound $C(x^n)$ may be used as the codeword. This interval subdivision may be done sequentially, so we need only calculate the subinterval and thereby the codeword associated with the sequence we are coding.

If we were to order the intervals of all the possible sequences of length n one after another, we would have partitioned the unit interval $[0; 1[$ into intervals of lengths equal to the probabilities of the strings. (Summing the probabilities of all possible sequences adds to unity since we have presumed a probability measure.)

For the sake of simplicity we restrict ourselves to the case of a binary source alphabet. The sequence x^n is extended to either x^n0 or x^n1. The codeword subinterval (of x^{n+1}) is calculated by the following recursion:

$$C(x^n0) = C(x^n) \qquad \text{if } x_{n+1} = 0, \tag{1.13}$$

$$C(x^n1) = C(x^n) + P(x^n0) \qquad \text{if } x_{n+1} = 1, \tag{1.14}$$

$$P(x^ni) = P(x^n)P(i|x^n), \quad i = 0, 1. \tag{1.15}$$

The width of the interval associated with x^n equals its probability $P(x^n)$. We note that (1.15) is a decomposition of the probability leading to the chain rule (1.4). Using pointers with a spacing of powers of $1/2$, at least one pointer of length $\lceil -\log_2(P(x^n)) \rceil$ will point to the interval. ($\lceil y \rceil$ denotes the ceiling or round-up value of y.) This suggests that arithmetic coding yields a code length that on average is very close to the entropy.

Before formalizing this statement, the basic recursions of binary arithmetic coding (1.13)–(1.15) are generalized for a finite alphabet, $\mathcal{A}$:

$$C(x^n0) = C(x^n),$$

$$C(x^ni) = C(x^n) + \sum_{j<i} P(x^nj), \quad i = 1, \dots, |\mathcal{A}| - 1,$$

$$P(x^ni) = P(x^n)P(i|x^n), \quad i = 0, \dots, |\mathcal{A}| - 1,$$

where $|\mathcal{A}|$ is the size of the symbol alphabet. Compared with the binary version, we still partition the interval into two for the lower bound, namely $j < i$ and $j \geq i$. The last

equation again reduces the interval to a size proportional to the probability of x^{n+1}, i.e. $P(x^{n+1})$. One difference is that the selected subinterval may have intervals both above and below.

Let a tag $T(x^n)$ denote the center point of the interval, $[C(x^n); C(x^n) + P(x^n)[$, associated with x^n by the recursions. We shall determine a precision of $l(x^n)$ bits, such that the codeword defined belongs to a prefix code for X^n. Now choose

$$l(x^n) = \lceil -\log_2(P(x^n)) \rceil + 1 < \log_2(P(x^n)) + 2. \tag{1.16}$$

Truncating the tag to $l(x^n)$ bits by (1.16) gives the codeword $c = \lfloor T(x^n) \rfloor_{l(x^n)}$, where $\lfloor z \rfloor_k$ denotes the truncation of a fractional binary number z to a precision of k digits.

Choosing $l(x^n)$ by (1.16) ensures not only that this value, c, points within the interval but also that $\lceil T(x^n) \rceil_{l(x^n)}$ $(= \lfloor T(x^n) \rfloor_{l(x^n)} + 2^{-l(x^n)})$ points within the interval.

This means that the codeword c will not be the prefix of the codeword generated by the procedure for any other outcome of X^n. Since this codeword construction is prefix-free it is also uniquely decodable.

Taking the expectation of (1.16) gives the right-hand side of

$$H(X^n) < L < H(X^n) + 2.$$

Thus the expected code length per source symbol, L/n, will converge to the entropy as n approaches ∞.

In addition to the fact that the arithmetic coding procedure above converges to the entropy with a complexity linear in the length of the sequence, it may also be noted that the probability expression is general, such that conditional probabilities may be used for sources with memory and these conditional probabilities may vary over the sequence. These are desirable properties in many applications. The issue of estimating these conditional probabilities is addressed in Chapter 5.

1.5.2 Finite-precision arithmetic coding

A major drawback of the recursions is that, even with finite-precision conditional probabilities $P(i|j)$ in (1.15), the left-hand side will quickly need a precision beyond any register width we may choose for these calculations. (We remind ourselves that it is mandatory that the encoder and decoder can perform exactly the same calculations.) There are several remedies to this. The most straightforward is (returning to the binary case) to approximate the multiplication of (1.15) by

$$\bar{P}(x^n i) = \lfloor \bar{P}(x^n)\bar{P}(i|x^n) \rfloor_k, \quad i = 0, 1, \tag{1.17}$$

i.e. k represents the precision of the representation of the interval. The relative position (or exponent) of the k digits is determined by the representation of $C(x^n)$ as specified below.

Having solved the precision problem of the multiplication, we still face the problem of representing $C(x^n)$ with adequate precision and the related problem of avoiding having to code the whole data string before generating output bits. Adequate precision for unique decodability may be achieved by ensuring that all symbols at all points are associated