Bounds on Code Length

Theorem: Let $l^*_1, l^*_2, \ldots, l^*_m$ be optimal codeword lengths for a source distribution **p** and a D-ary alphabet, and let L^* be the associated expected length of an optimal code. Then

$$H_D(X) \le L^* < H_D(X)+1$$

there is an overhead that is at most 1 bit, due to the fact that $\log 1/p_i$ is not always an integer. We can reduce the overhead per symbol by spreading it out over many symbols. With this in mind, let us consider a system in which we send a sequence of n symbols from X.

The symbols are assumed to be drawn i.i.d. according to $p(x)$. We can consider these n symbols to be a supersymbol from the alphabet X^n. Define L_n to be the expected codeword length per input symbol, that is, if $l(x_1, x_2, \ldots, x_n)$ is the length of the binary codeword associated with $(x_1, x_2, \ldots, x_n)$ (for the rest of this part, we assume that $D = 2$, for simplicity), then

$$L_n = \frac{1}{n}\sum p(x_1, x_2, \ldots x_n) l(x_1, x_2, \ldots x_n) = \frac{1}{n} El(X_1, X_2, \ldots X_n)$$

Bounds on Code Length

We can now apply the bounds derived above to the code:

$$H(X_1, X_2, \ldots X_n) \le El(X_1, X_2, \ldots X_n) \le H(X_1, X_2, \ldots X_n)+1$$

Since $X_1, X_2, \ldots, X_n$ are i.i.d., $H(X_1, X_2, \ldots, X_n) = H(X_i) = nH(X)$. Dividing by n we obtain:

$$H(X) \le L_n < H(X)+\frac{1}{n}$$

Hence, by using large block lengths we can achieve an expected codelength per symbol arbitrarily close to the entropy. We can use the same argument for a sequence of symbols from a stochastic process that is not necessarily i.i.d. In this case, we still have the bound

$$H(X_1, X_2, \ldots X_n) \le El(X_1, X_2, \ldots X_n) \le H(X_1, X_2, \ldots X_n)+1$$

Bounds on Code Length

Dividing by n again and defining L_n to be the expected description length per symbol, we obtain

$$\frac{H(X_1, X_2,...X_n)}{n} \leq L_n < \frac{H(X_1, X_2,...X_n)}{n} + \frac{1}{n}$$

If the stochastic process is stationary, then $H(X1,X2, \ldots, Xn)/n \rightarrow H(X)$, and the expected description length tends to the entropy rate as $n \rightarrow \infty$. Thus, we have the following theorem:

Theorem The minimum expected codeword length per symbol satisfies

$$\frac{H(X_1, X_2,...X_n)}{n} \leq L_n^* < \frac{H(X_1, X_2,...X_n)}{n} + \frac{1}{n}$$

Moreover, if $X_1, X_2, \ldots, X_n$ is a stationary stochastic process,

$$L_n^* \rightarrow H(\chi)$$

where $H(\chi)$ is the entropy rate of the process. This theorem provides another justification for the definition of entropy rate: it is the expected number of bits per symbol required to describe the process.

McMillan

We have proved that any instantaneous code must satisfy the Kraft inequality. The class of uniquely decodable codes is larger than the class of instantaneous codes, so one expects to achieve a lower expected codeword length if L is minimized over all uniquely decodable codes. In this section we prove that the class of uniquely decodable codes does not offer any further possibilities for the set of codeword lengths than do instantaneous codes.

Theorem (*McMillan*) The codeword lengths of any uniquely decodable D-ary code must satisfy the Kraft inequality.

Conversely, given a set of codeword lengths that satisfy this inequality, it is possible to construct a uniquely decodable code with these codeword lengths.

Moreover, a uniquely decodable code for an infinite source alphabet χ also satisfies the Kraft inequality.

McMillan

The theorem implies a rather surprising result: that the class of uniquely decodable codes does not offer any further choices for the set of codeword lengths than the class of prefix codes.

The set of achievable codeword lengths is the same for uniquely decodable and instantaneous codes. Hence, the bounds derived on the optimal codeword lengths continue to hold even when we expand the class of allowed codes to the class of all uniquely decodable codes.

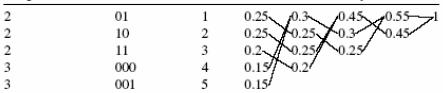
An optimal (shortest expected length) prefix code for a given distribution can be constructed by a simple algorithm discovered by Huffman. We will prove that any other code for the same alphabet cannot have a lower expected length than the code constructed by the algorithm.

Example 1

Example Consider a random variable X taking values in the set $X = \{1, 2, 3, 4, 5\}$ with probabilities 0.25, 0.25, 0.2, 0.15, 0.15, respectively. We expect the optimal binary code for X to have the longest codewords assigned to the symbols 4 and 5. These two lengths must be equal, since otherwise we can delete a bit from the longer codeword and still have a prefix code, but with a shorter expected length.

In general, we can construct a code in which the two longest codewords differ only in the last bit. For this code, we can combine the symbols 4 and 5 into a single source symbol, with a probability assignment 0.30. Proceeding this way, combining the two least likely symbols into one symbol until we are finally left with only one symbol, and then assigning codewords to the symbols, we obtain the following table:

Codeword Length	Codeword	X	Probability
2	01	1	0.25
2	10	2	0.25
2	11	3	0.2
3	000	4	0.15
3	001	5	0.15



This code has average length 2.3 bits.

Example 2

Consider a ternary code for the same random variable. Now we combine the three least likely symbols into one supersymbol and obtain the following table:

Codeword	X	Probability
1	1	0.25 → 0.5 → 1
2	2	0.25 → 0.25
00	3	0.2 → 0.25
01	4	0.15
02	5	0.15

This code has average length 1.5 ternary digits.

If $D \geq 3$, we may not have a sufficient number of symbols so that we can combine them D at a time. In such a case, we add dummy symbols to the end of the set of symbols. The dummy symbols have probability 0 and are inserted to fill the tree. Since at each stage of the reduction, the number of symbols is reduced by $D - 1$, we want the total number of symbols to be $1 + k(D - 1)$, where k is the number of merges. Hence, we add enough dummy symbols so that the total number of symbols is of this form. For example:

Example

Codeword	X	Probability
1	1	0.25 → 0.25 → 0.5 → 1.0
2	2	0.25 → 0.25 → 0.25
01	3	0.2 → 0.2 → 0.25
02	4	0.1 → 0.2
000	5	0.1 → 0.1
001	6	0.1
002	Dummy	0.0

This code has average length 1.5 ternary digits.

The Game of "20 Questions"

The game "20 questions" works as follows. Suppose that we wish to find the most efficient series of yes–no questions to determine an object from a class of objects. Assuming that we know the probability distribution on the objects, can we find the most efficient sequence of questions? We first show that a sequence of questions is equivalent to a code for the object. Any question depends only on the answers to the questions before it. Since the sequence of answers uniquely determines the object, each object has a different sequence of answers, and if we represent the yes–no answers by 0's and 1's, we have a binary code for the set of objects. The average length of this code is the average number of questions for the questioning scheme.

Also, from a binary code for the set of objects, we can find a sequence of questions that correspond to the code, with the average number of questions equal to the expected codeword length of the code. The first question in this scheme becomes: Is the first bit equal to 1 in the object's codeword?

Huffman Codes and "20 Questions"

Since the Huffman code is the best source code for a random variable, the optimal series of questions is that determined by the Huffman code. In the Example 1 the optimal first question is:

Is X equal to 2 or 3? The answer to this determines the first bit of the Huffman code. Assuming that the answer to the first question is "yes," the next question should be "Is X equal to 3?", which determines the second bit. However, we need not wait for the answer to the first question to ask the second. We can ask as our second question "Is X equal to 1 or 3?", determining the second bit of the Huffman code independent of the first. The expected number of questions EQ in this optimal scheme satisfies

$$H(X) \le EQ < H(X) + 1$$

Huffman coding and "slice" questions (Alphabetic codes).

We have described the equivalence of source coding with the game of 20 questions. The optimal sequence of questions corresponds to an optimal source code for the random variable.

However, Huffman codes ask arbitrary questions of the form "Is $X \in A$?" for any set $A \subseteq \{1, 2, \ldots, m\}$.

Now we consider the game "20 questions" with a restricted set of questions. Specifically, we assume that the elements of $X = \{1, 2, \ldots, m\}$ are ordered so that $p_1 \geq p_2 \geq \cdots \geq p_m$ and that the only questions allowed are of the form "Is $X > a$?" for some a.

Alphabetic Codes

The Huffman code constructed by the Huffman algorithm may not correspond to *slices* (sets of the form $\{x : x < a\}$).

If we take the codeword lengths ($l_1 \leq l_2 \leq \cdots \leq l_m$) derived from the Huffman code and use them to assign the symbols to the code tree by taking the first available node at the corresponding level, we will construct another optimal code.

However, unlike the Huffman code itself, this code is a *slice code*, since each question (each bit of the code) splits the tree into sets of the form $\{x : x > a\}$ and $\{x : x < a\}$.

Example

Example Consider the Example 1. The code that was constructed by the Huffman coding procedure is not a slice code.

But using the codeword lengths from the Huffman procedure, namely, {2, 2, 2, 3, 3}, and assigning the symbols to the first available node on the tree, we obtain the following code for this random variable:

$1 \rightarrow 00, \quad 2 \rightarrow 01, \quad 3 \rightarrow 10, \quad 4 \rightarrow 110, \quad 5 \rightarrow 111$

It can be verified that this code is a slice code, codes known as *alphabetic codes* because the codewords are ordered alphabetically.

Huffman and Shannon Codes

Using codeword lengths of $\lceil \log 1/pi \rceil$ (which is called *Shannon coding*) may be much worse than the optimal code for some particular symbol.

For example, consider two symbols, one of which occurs with probability 0.9999 and the other with probability 0.0001. Then using codeword lengths of $\lceil \log 1/pi \rceil$ gives codeword lengths of 1 bit and 14 bits, respectively.

The optimal codeword length is obviously 1 bit for both symbols. Hence, the codeword for the infrequent symbol is much longer in the Shannon code than in the optimal code.

Is it true that the codeword lengths for an optimal code are always less than $\lceil \log 1/pi \rceil$? The following example illustrates that this is not always true.

Example

Consider a random variable X with a distribution (1/3, 1/3, 1/4, 1/12). The Huffman coding procedure results in codeword lengths of $(2, 2, 2, 2)$ or $(1, 2, 3, 3)$, depending on where one puts the merged probabilities.

Both these codes achieve the same expected codeword length. In the second code, the third symbol has length 3, which is greater than $\lceil \log 1/p_3 \rceil$.

Thus, the codeword length for a Shannon code could be less than the codeword length of the corresponding symbol of an optimal (Huffman) code.

This example also illustrates the fact that the set of codeword lengths for an optimal code is not unique (there may be more than one set of lengths with the same expected value).

Although either the Shannon code or the Huffman code can be shorter for individual symbols, the Huffman code is shorter on average.

Also, the Shannon code and the Huffman code differ by less than 1 bit in expected codelength (since both lie between H and $H + 1$.)

Fano Codes

Fano proposed a suboptimal procedure for constructing a source code, which is similar to the idea of slice codes. In his method we first order the probabilities in decreasing order. Then we choose k such that

$$\left| \sum_{i=1}^{k} p_i - \sum_{i=k+1}^{m} p_i \right|$$

is minimized. This point divides the source symbols into two sets of almost equal probability. Assign 0 for the first bit of the upper set and 1 for the lower set.

Repeat this process for each subset. By this recursive procedure, we obtain a code for each source symbol. This scheme, although not optimal in general, achieves $L(C) \leq H(X) + 2$.