WEEK 6

# *Huffman Coding*

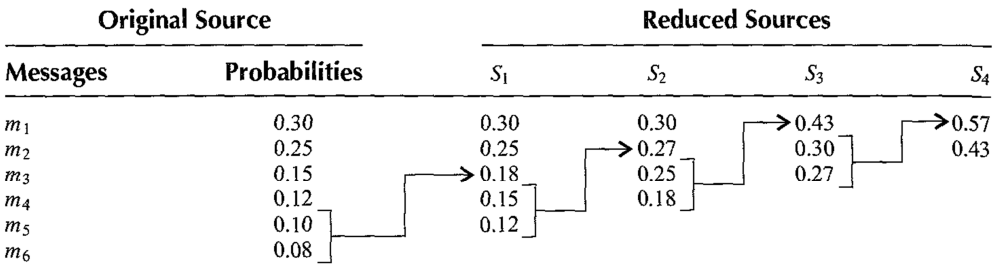
The source encoding theorem says that to encode a source with entropy $H(m)$, we need, on the average, a minimum of $H(\mathrm{m})$ binary digits per message, The number of digits in the code word is the **length** of the code word. Thus, the average word length of an optimum code is H(m). Unfortunately, to attain this length, in general, we have to encode a sequence of $N$ messages $(N \rightarrow \infty)$ at a time. If we wish to encode each message directly without using longer sequences, then, in general, the average length of the code word per message will be greater than H(m). In practice, it is not desirable to use long sequences, as they cause transmission delay and add to equipment complexity. Hence, it is preferable to encode messages directly, even if the price has to be paid in terms of increased word length. In most cases, the price turns out to be small. The following is a procedure, for finding the optimum source code, called the Huffman code.

The Huffman encoding algorithm proceeds as follows:

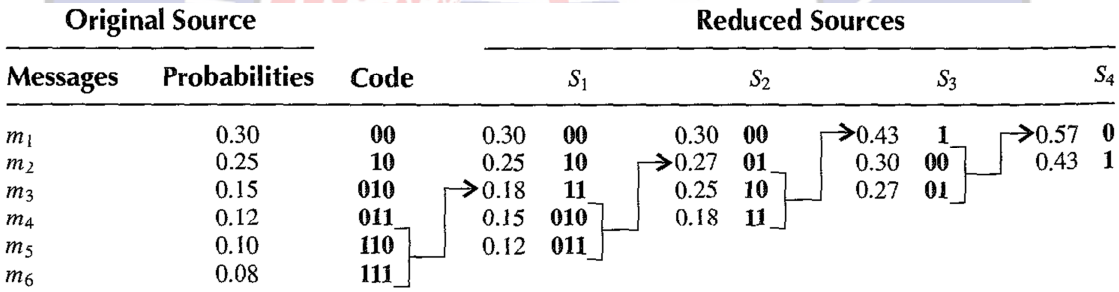
1. The source symbols are listed in order of decreasing probability. The two source symbols of lowest probability are assigned a 0 and a 1. This part of the step is referred to as a ***splitting*** stage.
2. These two source symbols are regarded as being ***combined*** into a new source symbol with probability equal to the sum of the two original probabilities. (The list of source symbols, and therefore source statistics, is thereby ***reduced*** in size by one.) The probability of the new symbol is placed in the list in accordance with its value.
3. The procedure is repeated until we are left with a final list of source statistics (symbols) of only two for which a 0 and a 1 are assigned.

The code for each (original) source symbol is found by working backward and tracing the sequence of 0's and 1's assigned to that symbol as well as its successors.

Refer to the table that shows a discrete source with 6 messages with their probabilities, applying Huffman algorithm:




Six messages with probabilities 0.30, 0.25, 0.15, 0.12, 0.10, and 0.08, respectively. We now combine the last two messages into one message with probability P5 + P6 = 0.18. This leaves five messages with probabilities, 0.30, 0.25, 0.15, 0.12, and 0.18. These messages are now rearranged in the second column in the order of descending probability. We repeat this procedure by combining the last two messages in the second column and rearranging them in the order of descending probability. This is done until the number of messages is reduced to 2. These two (reduced) messages are now assigned **0** and **1** as their first digits in the code sequence. We now go back and assign the numbers **0** and **1** to the second digit for the two messages that were combined in the previous step. We keep regressing this way until the first column is reached. The code finally obtained (for the first column) can be shown to be optimum. The complete procedure is shown below.




The optimum (Huffman) code obtained this way is also called a **compact code.** The average length of the compact code in the present case is given by:

$$L = \sum_{i=1}^{n} P_i L_i = 0.3(2) + 0.25(2) + 0.15(3) + 0.12(3) + 0.1(3) + 0.08(3)$$

$$= 2.45 \text{ binary digits}$$

The entropy $H(\text{m})$ of the source is given by

$$H(\text{m}) = \sum_{i=1}^{n} P_i \log_2 \frac{1}{P_i}$$

$$= 2.418 \text{ bits}$$

Recalling the coding efficiency:

$$\eta = \frac{H(\delta)}{L} = \frac{2.418}{2.45} = 0.976$$

the redundancy will be $\gamma = 1 - \eta$ = 1-0.976=0.024

A similar procedure is used to find a compact $r$-ary code. In this case we arrange the messages in descending order of probability, combine the last $r$ messages into one message, and rearrange the new set (reduced set) in the order of descending probability. We repeat the procedure until the final set reduces to $r$ messages. Each of these messages is now assigned one of the r numbers (0 , 1 , 2, …, r-1). We now regress in exactly the same way as in the binary case until each of the original messages is assigned a code.

For an r-ary code, we will have exactly $r$ messages left in the last reduced set if, and only if, the total number of original messages is equal to $r + k(r - 1)$, where $k$: is an integer. This is because each reduction decreases the number of messages by $r - 1$. Hence, if there is a total of $k$ reductions, the total number of original messages must be $r + k(r - 1)$. In case the original messages do not satisfy this condition, we must add some dummy messages with zero probability of occurrence until this condition is fulfilled. As an example, if $r = 4$ and the number of messages $n$ is 6, then we must add one dummy message with zero probability of occurrence to make the total number of messages 7, that is, [4 + 1 (4 — 1)], and proceed as usual. The procedure is illustrated in the below Example.

**Example**: A zero-memory source emits six messages with probabilities 0.3, 0.25, 0.15, 0.12, 0.1, and 0.08. Find the 4-ary (quaternary) Huffman code. Determine its average word length, the efficiency, and the redundancy.

Sol:

In this case, we need to add one dummy message to satisfy the required condition of $r + k(r - 1)$ messages and proceed as usual. The Huffman code is found in table shown below.

| Original Source | | | | |
|---|---|---|---|---|
| **Messages** | **Probabilities** | **Code** | **Reduced Sources** | |
| $m_1$ | 0.30 | **0** | 0.30 | **0** |
| $m_2$ | 0.25 | **2** | 0.30 | **1** |
| $m_3$ | 0.15 | **3** | 0.25 | **2** |
| $m_4$ | 0.12 | **10** | 0.15 | **3** |
| $m_5$ | 0.10 | **11** | | |
| $m_6$ | 0.08 | **12** | | |
| $m_7$ | 0.00 | **13** | | |

The length L of this code is:

$$L = 0.3(1) + 0.25(1) + 0.15(1) + 0.12(2) + 0.1(2) + 0.08(2) + 0(2)$$

$$= 1.3 \text{ 4-ary digits}$$

Also,

$$H_4(m) = \sum_{i=1}^{6} P_i \log_4 \frac{1}{P_i}$$

$$= 1.209 \text{ 4-ary units}$$

The code efficiency $\eta$ is given by:

$$\eta = \frac{H(\delta)}{L} = \frac{1.209}{1.3} = 0.93$$

the redundancy $\gamma = 1 - \eta$ = 1-0.93 = 0.07

NOTE: To achieve code efficiency $\eta \rightarrow 1$, we need N $\rightarrow \infty$. The Huffman code uses N = 1, but its efficiency is, in general, less than 1. A compromise exists between these two extremes of N = 1 and N=∞, We can use N = 2 or 3. In most cases, the use of N = 2 or 3 can yield an efficiency close to 1.

**Example**:

A zero-memory source emits messages $m_1$ and $m_2$ with probabilities 0.8 and 0.2, respectively. Find the optimum (Huffman) binary code for this source as well as for its **second order extensions** (that is, for $N = 2$). Determine the code efficiencies in each case.

The Huffman code for the source is simply **0** and **1,** giving $L = 1$, and

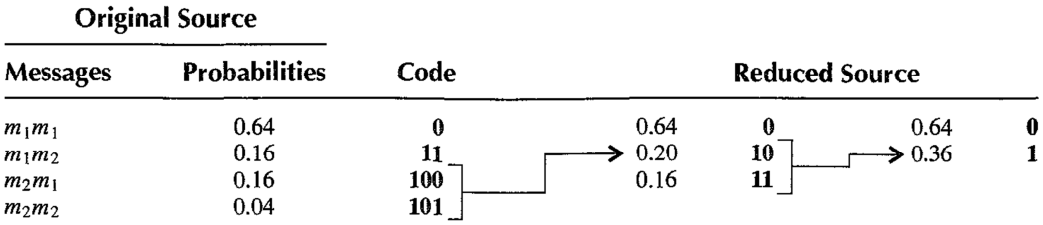
$$H(m) = -(0.8 \log 0.8 + 0.2 \log 0.2)$$

$$= 0.72 \text{ bit}$$

Hence,

$\eta = 0.7$ , $\gamma = 1 - 0.72 = 0.28$

For the second-order extension of the source (N = 2), there are four possible composite messages, $m_1m_1$, $m_1m_2$, $m_2m_1$ and $m_2m_2$ with probabilities 0.64, 0.16, 0.16, and 0.04, respectively. The Huffman code is obtained as shown in the following table:




In this case the average word length L' is:

$$L' = 0.64(1) + 0.16(2) + 0.16(3) + 0.04(3) = 1.56$$

This is the word length for two messages of the original source. Hence L, the word length per message,

$$L = L'/2 = 0.78$$

And $\eta = \frac{0.72}{0.78} = 0.923$ , the efficiency increased with the second order extension

H.W, redo the previous example with $m_1 = m_2$

H.W: show the construction of the Huffman code for a five symbols having the following frequency:

| Symbol | A | B | C | D | E |
|---|---|---|---|---|---|
| Count | 15 | 7 | 6 | 6 | 5 |