Lecture 4: Entropy and Lossless Compression

ENGR 76 lecture notes — April 11, 2024

Ayfer Ozgur, Stanford University

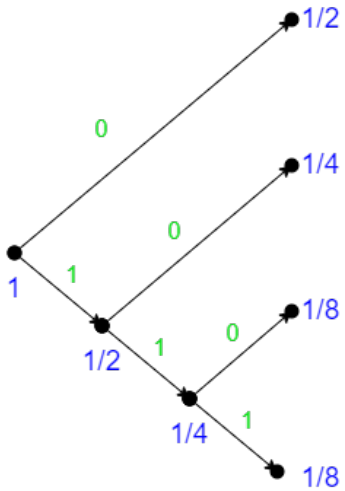
In the last lecture, we talked about the source coding problem and the advantages of prefix-free codes. In this lecture, we describe an algorithm that can be used to generate a prefix-free code for any source with an arbitrary probability distribution. This algorithm is called the Huffman algorithm. We next describe this algorithm.

1 Huffman algorithm

Given a source with an arbitrary alphabet size and probability distribution, the Huffman algorithm proceeds in the following steps:

- Start with the symbols in the alphabet with their respective probabilities. These will be leaf nodes in a binary tree.
- Find the two symbols with the lowest probability and merge them into a single symbol with probability equal to the sum of the probabilities for the two symbols. On the tree, the node for the new symbol has the nodes for the two symbols as its children nodes. Remove the two symbols from consideration for future steps.
- Repeat the previous step until we are left with a single symbol with probability one. This will be the root node in the binary tree. At any step, if you have multiple options for selecting the two lowest probability symbols, make an arbitrary choice.
- Finally, for each internal node in the tree mark the two edges to the two children as `0` and `1`, respectively. For each symbol in the original alphabet, the codeword is given by the path from the root to the leaf node corresponding to that symbol.

An illustration is shown in the figure below for the example discussed in the previous lecture. Observe that we end up with the same code from the previous lecture.



Nucleotide	Probability	Codeword
a	1/2	0
g	1/4	10
c	1/8	110
t	1/8	111

Since, only the leaves of the tree are used for the codewords (rather than using internal nodes), no codeword is a prefix of another codeword. Thus, Huffman algorithm always gives a prefix-free code. Moreover, the

code constructed by the Huffman algorithm is the best prefix-free code for any source, i.e. it has the smallest $\bar{l}$ among all prefix-free codes and even all uniquely decodable codes. We state this fact without proof.

Fact 1: The Huffman code is optimal for any source, i.e., it has the smallest $\bar{l}$ among all uniquely decodable codes.

Not all uniquely decodable codes are prefix-free codes. Since Huffman codes are prefix-free codes, this shows that the optimal $\bar{l}$ can actually be achieved with prefix-free codes, despite uniquely decodable codes being a richer class of codes.

The code above is reproduced in the table below, with $c(x)$ representing the codeword for symbol x, and $l(x)$ representing the length of the codeword for symbol x.

x	$P(X = x)$	$c(x)$	$l(x)$
a	$\frac{1}{2}$	`0`	1
g	$\frac{1}{4}$	`10`	2
c	$\frac{1}{8}$	`110`	3
t	$\frac{1}{8}$	`111`	3

We see that

$$\bar{l} = \frac{1}{2} \times 1 + \frac{1}{4} \times 2 + \frac{1}{8} \times 3 + \frac{1}{8} \times 3 = 1.75$$

and

$$\begin{aligned} H(X) &= \frac{1}{2} \times \log 2 + \frac{1}{4} \times \log 4 + \frac{1}{8} \times \log 8 + \frac{1}{8} \times \log 8 \\ &= \frac{1}{2} \times 1 + \frac{1}{4} \times 2 + \frac{1}{8} \times 3 + \frac{1}{8} \times 3 \\ &= 1.75 \end{aligned}$$

Thus, for this case, we get $\bar{l} = H(X)$. In fact, we observe that the two expressions match term-by-term and that $P(X = x) = \frac{1}{2^{l(x)}}$ for each $x \in \mathcal{X}$. This leads to the definition of dyadic sources and how Huffman codes achieve entropy for dyadic sources.

Dyadic source: A source is dyadic if all its probabilities are integer powers of $\frac{1}{2}$.

Fact 2: For a dyadic source, the number of symbols whose probability is smallest is even.

For example, in the distribution above, we have two symbols with probability $\frac{1}{8}$. In the Huffman code construction, these symbols are merged in the first step to give us a new source with probabilities $(\frac{1}{2}, \frac{1}{4}, \frac{1}{4})$. Again we have two symbols with the lowest probability of $\frac{1}{4}$, and we merge them in the next step. Thus we can make two observations about the Huffman code construction process for a dyadic source:

- We get a dyadic source at each step.
- Each merge step involves two symbols with identical probabilities, and the new node obtained from the merge has double the probability of the children.

Since the root has probability 1, we have exactly k branches to go from the root to a leaf node with probability 2^{-k}. To summarize, for a dyadic source, Huffman code yields $l(x) = k$ for $x \in \mathcal{X}$ with $P(X = x) = 2^{-k}$. In other words,

$$l(x) = \log \frac{1}{P(X = x)}, \ x \in \mathcal{X}$$

We have:

Fact 3: For a dyadic source, the Huffman code is a prefix-free code with $\bar{l} = H(X)$.

In general, for non-dyadic sources, Huffman codes need not achieve entropy. For example, consider a biased coin with $\mathcal{X} = \{H, T\}$, $P(X = H) = \frac{1}{4}$, $P(X = T) = \frac{3}{4}$. Since this is a binary source, the Huffman code construction is a single merge step, and we get codewords $c(H) = \texttt{0}$ and $c(T) = \texttt{1}$. Clearly, $\bar{l} = 1$ bit, but the entropy $H(X) = \frac{1}{4}\log\frac{4}{1} + \frac{3}{4}\log\frac{4}{3} = 0.811$ bits (recall that the unbiased coin has the highest entropy equal to 1 bit). Thus, we get $\bar{l} > H(X)$.

In general, the entropy serves as a lower bound for the average codeword length of any uniquely decodable code for that source.

Fact 4: For any uniquely decodable code for a source X, $\bar{l} \geq H(X)$.

At this point, we have seen that the Huffman algorithm gives the best uniquely decodable code for that source, which is also prefix-free, and that $\bar{l} \geq H$ for the resultant code. It can also be shown that $\bar{l}$ cannot be too far from H for the Huffman code in the following sense.

Fact 5: The average codeword length of the optimal prefix-free code constructed by the Huffman algorithm is such that $H \leq \bar{l} \leq H + 1$.

In other words, the expected codeword length for the Huffman code is always within 1 bit of the source entropy.

The 1-bit gap between $\bar{l}$ and H may be negligible for sources with large entropy, but quite significant for sources with entropy below 1 bit. Consider the source X to be the outcome of a biased coin toss, such that $P(X = H) = 1/4$ and $P(X = T) = 3/4$. The Huffman code for this source is the trivial code that assigns codewords 0 and 1 to the two possible outcomes; hence $\bar{l} = 1$. However, the entropy of this source is $H = 0.811$ bits/symbol which is 20% smaller than the entropy. Can we do anything beyond using this trivial code in this case?

Recall from Lecture #3 that we are often interested in not just encoding a single outcome of the source but a sequence of outcomes, e.g., $HTTHTTTHT$ in the case of a sequence of coin flips. How about, instead of generating a code for each outcome individually, we consider *pairs of outcomes together* and generate a Huffman code for a block of two outcomes? Assuming coin flips are independent, this will yield a new source (X_1, X_2) with four possible outcomes $\{HH, HT, TH, TT\}$ and the following probabilities:

$$P(X_1X_2 = HH) = \frac{1}{4} \times \frac{1}{4} = \frac{1}{16}, \quad P(X_1X_2 = HT) = \frac{1}{4} \times \frac{3}{4} = \frac{3}{16},$$
$$P(X_1X_2 = TH) = \frac{3}{4} \times \frac{1}{4} = \frac{3}{16}, \quad P(X_1X_2 = TT) = \frac{3}{4} \times \frac{3}{4} = \frac{9}{16};$$

By applying the Huffman algorithm to this "new" source, we can get a code with expected codeword length $\bar{l} = 1.69$ bits per pair of flips, or equivalently $\bar{l} = 1.69/2 = 0.845$ bits per flip. Note that this is already within a few percentage points of the entropy of the source. In general, by applying the Huffman algorithm to a large block of symbols, we can make the average number of bits per source symbol become arbitrarily close to the source entropy. This is known as *Shannon's source coding theorem*, which we state below.

Theorem (Shannon's Source Coding Theorem)**.** *The entropy of a source equals the minimum number of bits per source symbol necessary on average to encode a sequence of independent and identically distributed symbols from that source. In general, this may require the use of block coding, where blocks of symbols are encoded together.*

We next provide a proof for this fact by applying the Huffman algorithm to blocks of n symbols at a time. Note that identically distributed means that all n symbols $X_1, X_2, X_3, \ldots, X_n$ have the same marginal distribution. In particular, this implies that they all individually have the same entropy, i.e.

$$H(X_1) = H(X_2) = \cdots = H(X_n) = H(X).$$

Moreover, since these n random variables are independent from each other, we have

$$H(X_1, \ldots, X_n) = \sum_{i=1}^{n} H(X_i) = nH(X). \tag{1}$$

Now assume that we generate a Huffman code for the block of n symbols $X_1, X_2, X_3, \ldots, X_n$ by treating the block as one "super symbol" taking values in the alphabet $\mathcal{X}^n$. We have proved in the previous lecture that the average codeword length for the Huffman code is bounded between the entropy of the source and the entropy plus 1. In this case because we are treating the block of n symbols as one "super symbol", we need to consider the entropy of the "super symbol", or equivalently the joint entropy of the block $X_1, X_2, X_3, \ldots, X_n$. Thus, by Fact 5 above we have

$$H(X_1, \ldots, X_n) \leq \bar{l}_n \leq H(X_1, \ldots, X_n) + 1,$$

where $\bar{l}_n$ is the average codeword length of the Huffman code. Note that this code encodes n symbols at a time. Therefore to find the average number of bits per single source symbol, we need to normalize $\bar{l}_n$ by n. This yields

$$\frac{H(X_1, \ldots, X_n)}{n} \leq \frac{\bar{l}_n}{n} \leq \frac{H(X_1, \ldots, X_n)}{n} + \frac{1}{n}.$$

Using (1), we get

$$H(X) \leq \frac{\bar{l}_n}{n} \leq H(X) + \frac{1}{n}.$$

We observe that by taking n arbitrarily large, we can make the average number of bits per source symbol arbitrarily close to the entropy of the source.