Lecture 6 — Jan 26

Lecturer: David Tse *Scribe: Brett Larsen, Nitish Padmanaban, Kevin F,Vivek B*

6.1 Outline

- Kraft's Inequality
- Approaching entropy
- Huffman's Code

6.1.1 Reading

- Cover and Thomas - Sections 7.1 - 7.4

6.2 Recap: Prefix Codes

Up until now, we have considered prefix codes for probability distributions where all the probabilities are powers of $\frac{1}{2}$, i.e. *dyadic distributions*. Recall that we defined a prefix code to be one in which no codeword appears as the prefix of another codeword. These codes can be represented diagrammatically using a tree structure where all the codewords are at the leaves of the tree, as demonstrated for three basic probability distributions in Figure 6.1

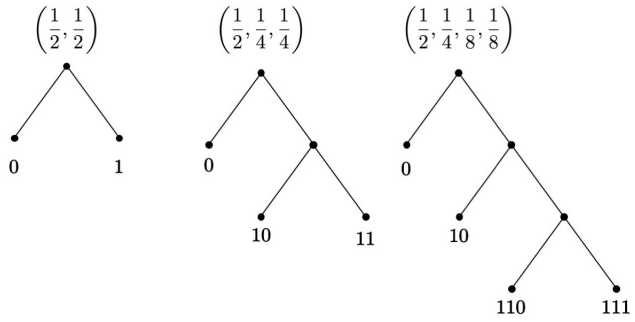


Figure 6.1: Three examples of simple prefix codes

Theorem 1. For any dyadic probability distributions, there exists a code whose codeword-length for symbol i is exactly equal to $\log \frac{1}{p_i(x)}$ and as a result, the expected code length matches the entropy:

$$\ell_i = \log \frac{1}{p_i(x)} \implies \mathbb{E}[\ell(X)] = \mathbb{E}\left[\log \frac{1}{p(X)}\right] = H(X). \tag{6.1}$$

Proof. Proved in HW 3. □

For dyadic distributions, can we obtain codes with compression rate **lower** than $H(X)$? This question is answered in section (6.3.2).

6.3 General probability distributions

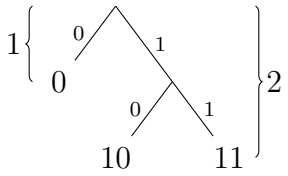
The next logical question is - how to design codes when the probabilities are no longer dyadic distributions? As an example, consider the simple distribution $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. For any valid prefix code, ℓ_i will not be equal to $\log \frac{1}{p_i} = \log 3 \approx 1.58$. This leaves us with two general questions:

1. Are there codes that can achieve compression rates lower than the entropy rate H? Note that this question remains unanswered for dyadic distributions too.
2. If not, what is the best achievable compression rate compared to entropy rate H?

Consider a random variable $X \sim p$ over alphabet $\mathcal{X} = \{1, 2, \cdots, m\}$, and without loss of generality assume $p_1 \geq p_2 \geq \cdots \geq p_m$. Finding the prefix code with minimum expected code length can be posed as the following **optimization problem**.

$$\begin{aligned} \underset{\{\ell_i\}}{\text{minimize}} \quad & \sum_i p_i \ell_i \\ \text{subject to} \quad & \ell_i \text{ a positive integer} \\ & \ell_i\text{: codeword lengths of a prefix code} \end{aligned} \tag{6.2}$$

In the current formulation of the optimization problem, the 2^{nd} condition - ℓ_i being lengths of a prefix code - is not tractable for mathematical analysis. For example, in the case of 3 symbols, we would get one possible tree,



This tree has two possible depths, corresponding to two possible codeword lengths, $\ell_1 = 1$ and $\ell_2 = \ell_3 = 2$. Unfortunately, enumerating all trees becomes quickly intractable and again not very useful as M - the number of symbols, grows large.

However, it turns out that the codewords in prefix codes satisfy a simple constraint, discussed in the following section.

6.3.1 Kraft's Inequality

Lemma 1. (*Kraft's Inequality*) For any prefix code over binary strings, the codeword lengths $\ell_1, \ell_2, \ldots \ell_m$ must satisfy the inequality

$$\sum_i 2^{-\ell_i} \leq 1 \tag{6.3}$$

Conversely, given a set of codeword lengths that satisfy the inequality (6.3), there exists a prefix code with these codeword lengths.

The above inequality (6.3) holds with an **equality** if the prefix codes use **all** the leaves in their tree representation.

Proof. To develop Kraft's inequality, we consider two observations about the trees we have been developing. The **first** observation is that the depth of the leaf in the tree is equal to the length of the codeword it is assigned. Leaves on the first level (the top node would be level 0) are assigned codewords of length 1, leaves on the second level are assigned codewords of length 2, etc. The **second** observation is that we can think of these trees as the result of a **dynamical process** dividing up the unit interval $[0, 1]$. Thus, we can correspond each leaf to a sub-interval, and the length of the sub-interval occupied by a leaf at depth ℓ is $2^{-\ell}$. This is demonstrated for two simple distributions in Figure 6.2.

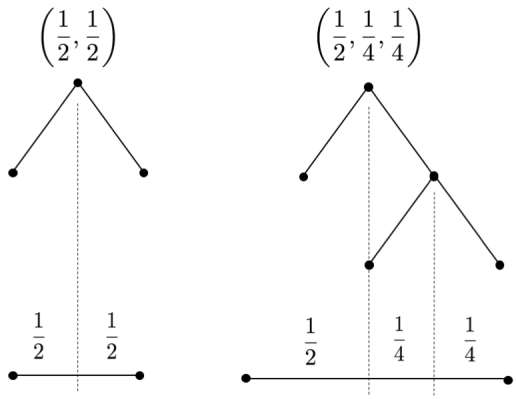


Figure 6.2: Demonstration of how prefix codes map onto an interval of length 1.

Since the subintervals of the codewords do not overlap (the interior nodes are not codewords), they sum up to 1. Thus, we have the following constraint:

$$\sum_i 2^{-\ell_i} = 1 \tag{6.4}$$

If we do not use all the leaves of the tree ("dumb trees"), the codeword lengths will satisfy equation (6.4) with an lesser-than inequality instead of an equality.

The proof of the converse is left as an exercise to the reader. □

Now we use Lemma 1 to reformulate our optimization problem 6.2 as:

$$\begin{aligned} \underset{\{\ell_i\}}{\text{minimize}} \quad & \sum_i p_i \ell_i = \mathbb{E}[\ell_i] \\ \text{subject to} \quad & \ell_i \text{ a positive integer} \\ & \sum_{i=1}^{m} 2^{-\ell_i} = 1 \end{aligned} \tag{6.5}$$

This is an integer programming program, and it is not immediately clear if this can be solved efficiently. Before solving this optimization problem, we will first say a few things about the properties of the optimal solution.

6.3.2 Coding Schemes and Relative Entropy

Using the formulation (6.5), we now ask whether the entropy can be reached, or beaten. This is equivalent to finding the sign achieved the following expression:

$$\sum_{i=1}^{M} p_i \ell_i - H(X) = \sum_{i=1}^{M} p_i \log \frac{1}{2^{-\ell_i}} - H(X). \tag{6.6}$$

Using constraint (6.5), we can define a probability distribution q such that $q_i = 2^{-\ell_i}$. Using this definition, we now expand the equation (6.6) as follows

$$\begin{aligned} \sum_{i=1}^{M} p_i \log \frac{1}{2^{-\ell_i}} - H(X) &= \sum_{i=1}^{M} p_i \log \frac{1}{q_i} - H(X) \\ &= \sum_{i=1}^{M} p_i \log \frac{1}{q_i} - \sum_{i=1}^{M} p_i \log \frac{1}{p_i} \\ &= \sum_{i=1}^{M} p_i \log \frac{p_i}{q_i} \\ &= D(p||q) \geq 0. \end{aligned}$$

At the last step, we recognize that the expression is equivalent to the relative entropy between p and q, which is always nonnegative. This implies that our objective function (6.5) is lower bounded by the entropy, allowing us to conclude that we **cannot beat** the entropy.

Knowing this, we now wish to find how inefficient our optimal code is compared to the entropy. The amount of inefficiency, also called the redundancy, is defined to be the expected number of extra bits per symbol we send (as compared to the entropy).

Suppose we wish to define a code with codeword lengths

$$\tilde{\ell}_i = \left\lceil \log \frac{1}{p_i} \right\rceil. \tag{6.7}$$

We have

$$\sum_{i=1}^{M} 2^{-\tilde{\ell}_i} = \sum_{i=1}^{M} 2^{-\left\lceil \log \frac{1}{p_i} \right\rceil} \leq \sum_{i=1}^{M} 2^{-\log \frac{1}{p_i}} = \sum_{i=1}^{M} p_i = 1.$$

Since the lengths $\tilde{l}_i$ (6.7) satisfy Kraft's inequality (Lemma 6.4), there must exist some valid prefix code with codeword lengths $\tilde{\ell}_i$. Having established this, the expected number of bits per symbol is

$$\mathbb{E}\left[\tilde{\ell}(X)\right] = \mathbb{E}\left[\left\lceil \log \frac{1}{p(X)} \right\rceil\right] \leq \mathbb{E}\left[1 + \log \frac{1}{p(X)}\right] = H(X) + 1,$$

where the inequality follows immediately from the definition of the ceiling function. We now have an upper bound on compression rate as 1 bit more than entropy, for some code. The optimal set of codeword lengths, $\ell^\star$, will also clearly satisfy this upper bound, giving us the inequalities

$$H(X) \leq \mathbb{E}\left[\ell^\star(X)\right] \leq H(X) + 1. \tag{6.8}$$

We've shown that we can compress within 1 bit of entropy. The important question - is 1 bit small or large? The answer, as with most engineering problems, is it depends. In particular, the extra 1 bit is large for low entropy rates.

We now show that there exists a sequence of codes that asymptotically achieve compression rate equal to $H(X)$. We accomplish this by constructing codes over multiple symbols instead of single symbol. For example, if we code over two symbols - using Eq. (6.8) for $X = (X_1, X_2)$, we have

$$H(X_1, X_2) \leq \mathbb{E}\left[\ell^\star(X_1, X_2)\right] \leq H(X_1, X_2) + 1.$$

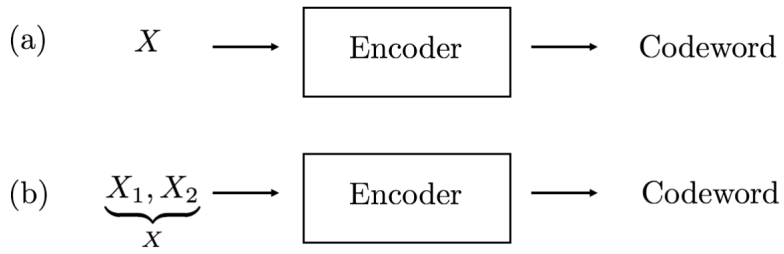


Figure 6.3: Encoding longer sequences of bits

Extending this reasoning, we can code over n symbols at a time, giving

$$H(X_1, X_2, \ldots, X_n) \leq \mathbb{E}\left[\ell^\star(X_1, X_2, \ldots, X_n)\right] \leq H(X_1, X_2, \ldots, X_n) + 1.$$

Finally, the objective we actually care about is bits required per input symbol, not bits required per codewords themselves. Therefore, the bits per symbol satisfies

$$\frac{H(X_1, X_2, \ldots, X_n)}{n} \leq \frac{\mathbb{E}\left[\ell^\star(X_1, X_2, \ldots, X_n)\right]}{n} \leq \frac{H(X_1, X_2, \ldots, X_n)}{n} + \frac{1}{n}.$$

Finally, by taking the limit as n goes to ∞, we get

$$\lim_{n\to\infty} \frac{\mathbb{E}\left[\ell^\star(X_1, X_2, \ldots, X_n)\right]}{n} = H, \tag{6.9}$$

giving us that we can always construct a code such that the number of bits per symbol approaches the entropy rate, H.

Note that this bound on the compression rate is more general than those we have derived in previous sets of lectures because there was no constraint on the model (*i.i.d.*, Markov, etc.). It also demonstrates why it is often beneficial to compress over multiple symbols. The obstacle preventing us from achieving entropy with our code was the integer constraint; as we get to larger and larger alphabets, the entropy loss due to integer constraint becomes smaller and smaller.

Thus, we have provided a fairly effected bound on the expected codeword length of the optimal code ℓ^*, but can we actually find this ℓ^* efficiently? The answer is provided by an algorithm referred to as the Huffman code.

6.4 Huffman Code

Up until now, we've stated various results on the optimal code, without actually specifying the code itself! In this section, we present the setup to find the optimal code ℓ^* called the Huffman code. We start with two observations from our construction of prefix codes so far. The **first** observation is that we should assign the highest probability symbols codeword with the smallest length. Thus, if we order the probabilities in descending order, the lengths of an optimal code will be in the reverse order:

$$\begin{aligned} p_1 \geq p_2 \geq p_3 \geq \ldots \geq p_m \\ \ell_1 \leq \ell_2 \leq \ell_3 \leq \ldots \leq \ell_m \end{aligned} \tag{6.10}$$

The **second** observation is that if we use all the leaves of the tree then the deepest two leaves will be on the same level. Let's consider what happens when we perform the following operation, combining the two lowest leaves into a single leaf by summing their probabilities:

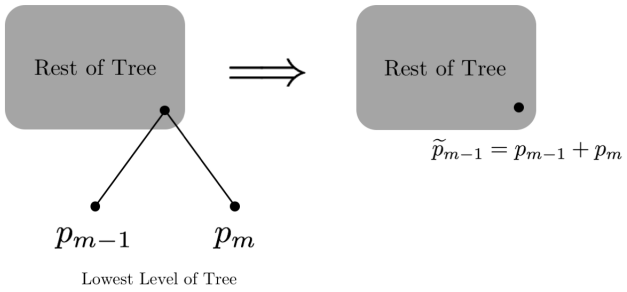


Figure 6.4: Recursive operation for the Huffman code.

We now have a random variable with probabilities $p_1, p_2, \ldots, p_{m-2}, \widetilde{p}_{m-1}$, where $\widetilde{p}_{m-1} = p_{m-1} + p_m$. But since we know that $\ell_{m-1} = \ell_m$, we can solve this smaller system for the optimal code and then add the leaves back! To see this more clearly, let's examine the expression for the expected codeword length:

$$\begin{aligned}\sum_i p_i \ell_i &= \sum_{i=1}^{m-2} p_i \ell_i + p_{m-1}\ell_{m-1} + p_m \ell_m \\ &= \sum_{i=1}^{m-2} p_i \ell_i + p_{m-1}(\ell'_{m-1}+1) + p_m(\ell'_{m-1}+1) \\ &= \underbrace{\sum_{i=1}^{m-2} p_i \ell_i + \widetilde{p}_{m-1}(\ell'_{m-1})}_{\text{Objective function for smaller problem}} + p_{m-1} + p_m\end{aligned} \tag{6.11}$$

where ℓ'_{m-1} is the length of the codewords one level above the bottom two leaves. It is easy to see that this objective function for the smaller problem also satisfies Kraft's inequality and thus we have effectively reduced the size of our problem. The natural next step then is to apply this process recursively and similarly eliminate the new two lowest leaves. Eventually we will end up with a system with two leaves which we know immediately how to solve and then add the leaves back one by one! In summary, the Huffman code will allow us to develop an optimal code by working from the bottom of the tree up.

The complete Huffman algorithm:

Algorithm 1 Huffman Coding

procedure HUFFMAN(p)
  while $length(p) > 1$ **do** ▷ Until we get to the root node
    $p \leftarrow sort(p)$ ▷ Sort in descending order
    $M \leftarrow length(p)$
    Connect the symbols $M-1$ and M with a branch, and replace this branch with a node representing the branching and both symbols.
    $\tilde{p} \leftarrow p_{M-1} + p_M$ ▷ Add lowest probabilities
    $p \leftarrow (p_1, p_2, \ldots, p_{M-2}, \tilde{p})$ ▷ p is still a valid distribution
  end while
  Construct tree by expanding all nodes to their branched symbols.
  $c \leftarrow$ empty codewords
  for all nodes in tree in breadth first order **do**
    $left \leftarrow$ all descendant codes of the left branch
    $right \leftarrow$ all descendant codes of the right branch
    $\forall c_l \in left,\ c_l \leftarrow c_l + 0$ ▷ $+$ denotes appending the bit
    $\forall c_r \in right,\ c_r \leftarrow c_r + 1$
  end for
  return c
end procedure