Electrical Engineering 229A Lecture 10 Notes

Daniel Raban

September 28, 2021

1 Shannon Codes, Huffman Codes, and Shannon-Fano-Elias Codes

1.1 Recap: Shannon codes

We have been discussing symbol by symbol codes $c : \mathcal{X} \to \{0,1\}^* \setminus \{\varnothing\}$, extended to $c : \mathcal{X}^* \setminus \{\varnothing\} \to \{0,1\}^* \setminus \{\varnothing\}$ by $c(x_1, \dots, x_n) = c(x_1) \cdots c(x_n)$.

Definition 1.1. We say that c is **uniquely decodable** if its extension is one to one.

Definition 1.2. We say that c is **prefix-free** if $c(x)$ is not a prefix of $c(y)$ for $x \neq y \in \mathscr{X}$.

We showed the Kraft inequality for prefix-free codes:

$$\sum_{x \in \mathscr{X}} 2^{-\ell(c(x))} \leq 1.$$

We saw this by looking at the length associated to the codewords, viewed as sitting in a binary tree. We also proved an extension of this inequality by McMillan to uniquely decodable codes.

We saw that the minimization of the expected length of codewords $\sum_{x \in \mathscr{X}} p(x)\ell(c(x))$ over prefix-free codes is equivalent to the integer programming problem

$$\text{minimize: } \sum_{x \in \mathscr{X}} p(x)\ell(x)$$

$$\text{subject to: } \sum_{x \in} 2^{-\ell(x)} \leq 1,$$

where $\ell(x)$ is a positive integer for each $x \in \mathscr{X}$. This equivalence came from Kraft's inequality and from the fact that for every sequence of lengths $(\ell(x), x \in \mathscr{X})$ satisfying Kraft's inequality, there is a prefix code $c : \mathscr{X} \to \{0,1\}^* \setminus \{\varnothing\}$ with $\ell(c(x)) = \ell(x)$ for each $x \in \mathscr{X}$. Similarly, the Kraft-McMillan inequality says that this is equivalent to the same problem for uniquely decodable codes.

Using Lagrange multipliers, we saw that minimizing $\sum_{x\in\mathscr{X}} p(x)\ell(x)$ subject to $\sum_{x\in\mathscr{X}} 2^{-\ell(x)} \le 1$ with real $\ell(x)$ has optimal solution

$$\ell(x) = \log \frac{1}{p(x)}, \qquad x \in \mathscr{X}$$

with optimal value

$$\sum_x p(x) \log \frac{1}{p(x)} = H(p).$$

This lead to the idea of a Shannon code.

Definition 1.3. A **Shannon code** is a code $c : \mathscr{X} \to \{0,1\}^* \setminus \{\varnothing\}$ with $\ell(c(x)) = \lceil \log \frac{1}{p(x)} \rceil$.

Such codes exist because

$$\sum_{x\in\mathscr{X}} 2^{-\lceil \log \frac{1}{p(x)} \rceil} \le \sum_{x\in\mathscr{X}} 2^{-\log \frac{1}{p(x)}} = 1.$$

For a Shannon code,

$$H(X) \le \sum_{x\in\mathscr{X}} p(x)\ell(c(x)) \le H(X) + 1.$$

So if we create a Shannon code on blocks of length n,

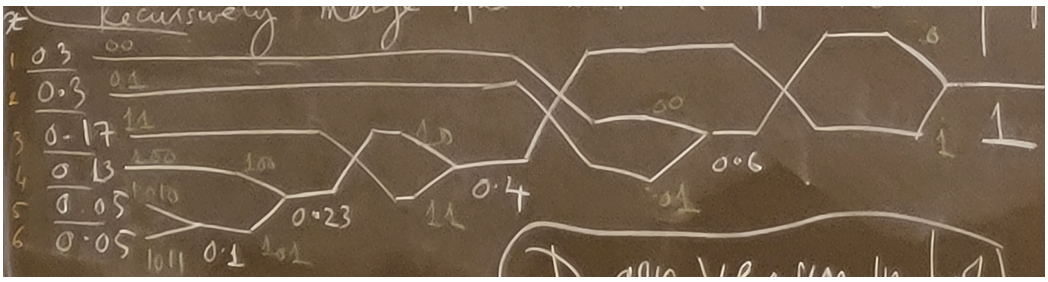
$$H(X_1, \dots, X_n) \le \sum_{x_1^n \in \mathscr{X}^n} p(x_1^n)\ell(c(x_1^n)) \le H(X_1, \dots, X_n) + 1,$$

with $X_i \sim p$. Dividing by n, the penalty is at most $\frac{1}{n}$ bits/symbol.

1.2 Huffman coding

It turns out that in this case, the integer programming problem can be solved exactly.[1] The **Huffman coding algorithm**, in one sentence, basically says the "Recursively merge the smallest probability pair of symbols."

Example 1.1. Draw the following diagram, successively merging the two smallest probabilities at each step:



[1] This is unusual. Integer programming problems are generally very computationally difficult.

Then label each branch with a 0 or a 1.

A D-ary version requires us to combine the smallest D probabilities at a time.

Theorem 1.1. *The Huffman code is optimal.*

Proof. Observe some properties that the solution to the integer problem must satisfy:

1. If $p(x) \geq p(y)$, then $\ell(x) \leq \ell(y)$.

   Proof. If not, interchange $\ell(x)$ and $\ell(y)$ to get a better code. □

2. There have to be at least two symbols $x, x' \in \mathscr{X}$ getting the longest length.

   Proof. If not, reducing the length of the longest codeword by removing a bit from the end gives a better code. □

Also, we can arrange the following:

3. There are two symbols $x, x' \in \mathscr{X}$ with longest length representations such that these representations differ only at the last bit, and these are the two smallest probability symbols.

   Proof. If the sibling of a longest length codeword is not present, we can reduce the length of that codeword by removing the last bit. To guarantee that these are the two smallest probability symbols, we can just relabel. □

This is enough to prove by induction that the Huffman coding algorithm is optimal. □

Example 1.2. A Shannon code can be worse than a Huffman code. One way to see this is to note that the expected length of a Huffman code is a continuous function on the probability simplex.

Now consider $\mathscr{X} = \{1, 2, 3, 4\}$ with

$$p(1) = \frac{1}{4}, \qquad p(2) = \frac{1}{4} + \varepsilon, \qquad p(3) = \frac{1}{4} - \varepsilon, \qquad p(4) = \frac{1}{4},$$

for small $\varepsilon > 0$. Here, the Shannon code is worse than the Huffman code because $\lceil \log \frac{1}{1/4-\varepsilon} \rceil = 3$.

1.3 Shannon-Fano-Elias Coding

Shannon-Fano-Elias coding is a precursor to **arithmetic coding** (which is widely used), allowing one to learn the statistics and "improve the code" as one goes along.[2] The idea comes from observing that if Z is a random variable with (invertible) CDF $F_Z(z) = \mathbb{P}(Z \le z)$, then

$$\mathbb{P}(Z \le F_Z^{-1}(u)) = F_Z(F_Z^{-1}(u)) = u, \qquad u \in [0,1].$$

This shows that $F_Z(Z)$ has uniform distribution on $[0,1]$, when F_Z^{-1} is well-defined. This is not true in general but suggests creating codes based on F_X, where $X \in \mathscr{X}$ for a finite $\mathscr{X} \subseteq \mathbb{R}$.

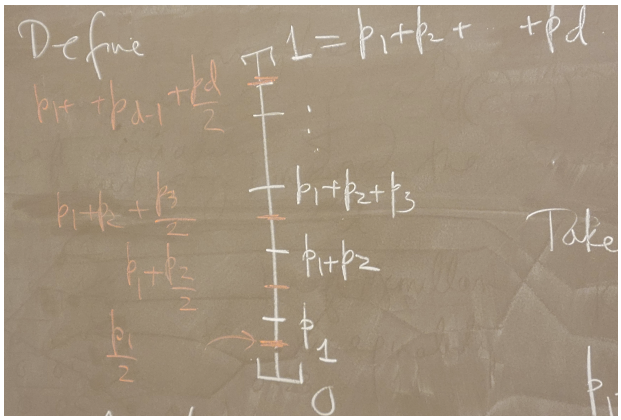
Suppose $|\mathscr{X}| = d$ with $\mathscr{X} = \{a_1, \dots, a_d\}$, and write

$$\mathbb{P}(X = a_i) = p_i, \qquad 1 \le i \le d.$$

For $x \in \mathbb{R}$,

$$F_X(x) = \mathbb{P}(X \le x) = \sum_{i : a_i \le x} p_i.$$

Here is the idea of how the code works. Draw the values of the CDF as follows:



Now draw the midpoints between these points at $p_1, p_1 + \frac{p_2}{2}, \dots$. Take the binary representation as a binary fraction of $p_1 + \cdots + p_{i-1} + \frac{p_i}{2}$. Then truncate it to get $\lceil \log \frac{1}{p_i} \rceil + 1$ bits. This will be prefix-free and within 2 bits of the entropy.

The arithmetic coding scheme is based on updating this procedure as we go.

[2] This is similar to adaptive control in machine learning.