Brief review from last time

Minimum Code Length

- \( L_c = E[l(x)] \)
- Optimal if \( L_c^* \) is as short as possible
- \( L_c \geq \frac{H(X)}{\log_2 D} \)

Outline

- Fano code
- Shannon code
- Huffman code

Fano code

- Put the probabilities in decreasing order
- Split as close to 50-50 as possible. Repeat with each half.
\[
\begin{array}{cccc}
\text{a} & 0.20 & 0 & 0 \\
\text{b} & 0.19 & 1 & 0 \\
\text{c} & 0.17 & 1 & \\
\text{d} & 0.15 & 0 & 100 \\
\text{e} & 0.14 & 0 & 1 \\
\text{f} & 0.06 & 1 & 110 \\
\text{g} & 0.05 & 1 & 1110 \\
\text{h} & 0.04 & 1 & 1111 \\
\end{array}
\]

Computing the expected value: 
\[
L_c = \sum x p(x) l(x) = (.2) \times 2 + (.19) \times 3 + (.17) \times 3 + (.15) \times 3 + (.14) \times 3 + (.06) \times 3 + (.05) \times 4 + (.04) \times 4 = 2.89 \text{ bits} > 2.81 \text{ bits},
\]
which was the lower bound.

**Conditions for optimal prefix code (assuming binary)**

An optimal prefix must satisfy:

- \( p(x_i) > p(x_j) \rightarrow l(x_i) \leq l(x_j) \) Otherwise, since \( E[\sum_i p(x_i) l(x_i)] = L_c \), we would produce a smaller sum by swapping the two.

- The two longest codewords must have the same length Follows from the tree structure. Otherwise, we can chop a bit off a codeword.

- In the tree corresponding to the optimal code, there must be two branches stemming from each intermediate node. Otherwise we would have free “parasyte” slots, filling up which would produce a code of shorter length.

**Huffman code construction**

1. Take the two smallest \( p(x_i) \) and assign each a different last bit. Then merge into a single symbol with summed probability.

2. Repeat step 1, until only one symbol remains.
Note that this assignment is not unique.

**Huffman optimality proof**

*Proof.* As mentioned before, an optimal code must satisfy \( p(x_i) > p(x_j) \rightarrow l(x_i) \leq l(x_j) \).

We prove by contradiction. Suppose \( \exists m > 2 \), s. t. \( c_m \) is the first sub-optimal code. (The case for \( m = 2 \) is trivial). Then \( \exists c'_m \), s. t. \( c'_m \) is optimal and \( L_{c'_m} < L_{c_m} \). Now let’s take \( c'_m \) and look at the two longest codewords, i.e. the two codewords with the lowest probabilities. Then merge them according to the Huffman recipe to create a new code \( c'_{m-1} \).

\[
L_{c'_{m-1}} = L_{c'_m} - (p_i + p_j) \times 1, \tag{1}
\]

where \( i \) and \( j \) are the two codewords with the longest length.

\[
L_{c_{m-1}} = L_{c_m} - (p_i + p_j) \tag{2}
\]

Combining (1) and (2) with the original assumption that \( L_{c'_m} < L_{c_m} \), we get \( L_{c'_{m-1}} < L_{c_{m-1}} \), which is a contradiction, since \( c_m \) was assumed to be the first suboptimal code.

**Optimal codes (non-integer length)**

Minimize \( \sum_i p(x_i)l(x_i) \)

subject to: \( \sum_i D^{-l(x_i)} \leq 1 \)

Use Lagrange multiplier method.
refresher on Lagrange (simplest form):
\[
\begin{align*}
\min f(x) \\
\text{s.t. } g(x) \leq 0 \\
f(x) + \lambda g(x)
\end{align*}
\]

\[
L(l(x), \lambda) = \sum p(x_i) l(x_i) + \lambda(\sum D^{-l(x_i)} - 1). \quad \text{Taking the derivative with respect to } l(x_i):
\]

\[
\frac{dL}{dl(x_i)} = p(x_i^*) - \lambda D^{-l(x_i)} = 0, \quad \text{in case of optimality.}
\]

\[
p(x_i) = \lambda D^{-l(x_i)} \sum p(x_i) = \lambda \sum D^{-l(x_i)} 1 = \lambda \sum D^{-l(x_i)} \lambda = \frac{1}{\sum D^{-l(x_i)}} = 1
\]

Therefore: \(l(x_i) = -\log_D \frac{p(x_i)}{\lambda} = -\log_D p(x_i).\)

**Shannon code**

- Round up optimal code lengths: \(l_i = \lceil -\log_D p(x_i) \rceil \leq -\log_D p(x_i) + 1\)
- \(l_i\) satisfy the Kraft Inequality \(\rightarrow\) Prefix code exists. Construction for such code is done earlier.
- Average length: \(\frac{H(X)}{\log D} \leq L_S \leq \frac{H(X)}{\log D} + 1.\)

**Proof. Average length:**

\[-\log_D p(x_i) \leq l_i = \lceil -\log_D p(x_i) \rceil \leq -\log_D p(x_i) + 1. \text{ Summing over both sides:} \]

\[\frac{H(X)}{\log D} \leq L_S \leq \frac{H(X)}{\log D} + 1.\]

So optimal code also must satisfy the bounds.

**Existence of the prefix code:**

Let’s take a look at \(\sum_{x_i} D^{-l(x_i)} = \sum_{x_i} D^{\lceil -\log_D p(x_i) \rceil} \leq \sum_{x_i} D^{-\log_D p(x_i)} \leq 1.\) Therefore prefix code must exist and here’s the construction for it: \(c_k = \sum_{i=1}^{k-1} p(x_i), \) for which all codes are different to \(l_i\) places.

**Shannon code examples:**

- good example
\[
\begin{array}{|c|c|c|c|}
\hline
x_1 & x_2 & x_3 & x_4 \\
\hline
p(x_i) & 1/2 & 1/4 & 1/8 \\
\hline
\end{array}
\]

\[-\log_2 p(x) = [1; 2; 3; 3] \text{ and } \lceil -\log_2 p(x) \rceil = [1; 2; 3; 3]. \text{ Therefore } L_S = 1.75 \text{ bits, which is exactly equal to } H(X), \text{ therefore for this example the code is optimal.}\]

• bad example

\[
\begin{array}{|c|c|}
\hline
x_1 & x_2 \\
\hline
p(x_i) & 0.99 & 0.01 \\
\hline
\end{array}
\]

\[-\log_2 p(x) = [0.0145; 6.64] \text{ and } \lceil -\log_2 p(x) \rceil = [1; 7]. \text{ Therefore } L_S = 1.06 \text{ bits, whereas } H(X) = 0.08 \text{ bits, so Shannon code does two orders of magnitude worse than the optimal.}\]