2 Section Week 2 - Coding with Information Theory

History lesson - Shannon 1948 "A Mathematical Theory of Communication".

Also declassified Shannon 1949 - "Communication theory of secrecy systems". We will talk about this more when we talk about cryptography.

"If I have seen further, it is by standing upon the shoulders of giants" - Newton

2.1 Entropy

In this setting, messages X are coming from the distribution P_X over Ω. This gives us some ability to take advantage of certain messages appearing more often than others. For this talk, generally $\Omega = [n]$.

(Today we work in the non noise model)

ex. how would you encode the set of 100 bit strings if I told you the all zero string shows up half the time?

As usual we want shortest possible codes, and we intuitively want to find the best (prefix free) code for the specific probability distribution we're working with.

In other words, optimizing $\sum_{x \in \Omega} |E(x)| P_X(x)$ for a code E.

Entropy gives us a theoretical lower bound for how parsimonious we can be in encoding from a distribution

English defn of entropy - In the best case, how many bits we need to convey a random message X drawn from the distribution P_X.

Harder english defn - In the best case, averaging over lots of messages all from the same distribution, how small can we make the expected length of our encoding?

Formal definition: first we define a valid encoding. For all n and any Ω

$$
\begin{aligned}
E &: \Omega^n \to \{0,1\}^* \\
D &: \{0,1\}^* \to \Omega^n \times \{\perp\} \\
\forall x \in &\Omega D(E(x)) = x \\
&E \text{ is prefix free}
\end{aligned}
$$

Now for some P_X over Ω, we say the entropy of $X^n \sim P_X^n$ is

$$\lim_{n\to\infty} \left(\min_{\text{valid E, D}} \left(\frac{1}{n} \mathbb{E}_{X^n \sim P_X^n} |E(X^n)| \right) \right)$$

Work through what this means for the entropy of a binomial

Not going to actually prove it as the nitty gritty is not particularly interesting:

Idea is to encode an n string of Bernoulli p bits as follows:

1. Send k, the number of 1s
2. Send an index in the set of $\binom{n}{k}$ possible strings with k 1s.

Using the weak LLN and Stirling's approximation, we get that $H(X) \leq h(p)$ where $h(p) = p \log \frac{1}{p} + (1-p) \log \frac{1}{(1-p)}$. We can also show that $H(X) \geq h(p)$ with some clever use of the Chernoff bound.

When we have a multinomial over a support Ω of size m, a similar calculation gives

$$H(X) = \sum_{\omega \in \Omega} \Pr[X = \omega] \log \frac{1}{\Pr[X = \omega]}$$

2.2 Huffman Codes

First we observe that every prefix free code can be visualized as mapping elements in the domain to nodes of a binary tree, where no node is mapped

to the parent of another node's image. Using this fact, we will draw a lot of binary trees to visualize what's going on.

Huffman codes nicely capture the intuition of using short strings to encode messages that appear often.

Suppose we are trying to encode $[n]$, where $P_X = (p_1, \ldots p_n)$. WLOG, assume $p_1 \geq \cdots \geq p_n$ The Huffman code is defined recursively as:

1. Start out with the base case of a binary tree with just the root.
2. Sort $(p_1, \ldots p_n)$ in decreasing order.
3. Define $(q_1, \ldots q_{n-1})$ where $q_i = p_i$ except $q_{n-1} = p_{n-1} + p_n$
4. Build the Huffman tree for $(q_1, \ldots q_{n-1})$. Let the resulting encoding function be E'
5. Set $E(i) = E'(i)$ for all i in $[m-2]$, then set

$$E(m-1) = E'(m-1) \circ 0$$
$$E(m) = E'(m-1) \circ 1$$

Work out the $(\frac{1}{4}, \ldots \frac{1}{4})$ and $(\frac{1}{2}, \frac{1}{6}, \ldots \frac{1}{6})$ examples on board with the tree

Theorem. *The Huffman code is optimal*

Proof. First we observe that an optimal encoding has lower probability messages map to longer strings (otherwise if there is some $p_i > p_j$ and $|E(i)| > |E(j)|$ we can just improve our code by swapping the encodings of i and j).

Now we prove by induction that for any distribution over n elements, the Huffman code is optimal. The base case is 1 message with the empty string as the encoding, which is trivially optimal.

Now suppose we have that the Huffman code is optimal for any distribution over $n-1$ elements. Let's consider the code for n elements. We know that

WLOG the optimal code on n elements has $n-1$ and n as neighboring leaves by the following observations:

Draw tree for this part of the explanation

First, in any optimal code, the binary tree representation has every single element on a leaf on this tree and every leaf has something mapping to it (otherwise we can just combine this empty leaf with its neighbor and shorten one of the encodings by 1). Next, $n-1$ and n have to be on the same level. Otherwise, if for example $n-1$ is on a higher level than n, then we can change the code by swapping $n-1$ with the neighbor of n and have less or equal expected coding length. Now that the encodings for $n-1$ and n are on the same level, we can swap the encoding for $n-1$ with the neighbor of n without changing the efficiency of the code.

Now let T be the Huffman tree and T' be some non-Huffman tree for the encoding of $[n]$. Let's assume for the sake of contradiction that T' is better than T. Let S and S' be the corresponding trees for the encoding of $[n-1]$ when the nodes for $n-1$ and n are merged. Let E_T denote the encoding defined by tree T. We write $\mathbb{E}[T]$ as shorthand for expected length of an encoding using tree T.

If we analyze the expected encoding lengths of our trees, we get

Skip algebra and say that full version is in the lecture notes

$$
\begin{aligned}
\mathbb{E}[T] &= \sum_{i\in[n]} p_i |E_T(i)| \\
&= \left(\sum_{i\in[n-2]} p_i |E_T(i)|\right) + p_{n-1}|E_T(n-1)| + p_n|E_T(n)| \\
&= \left(\sum_{i\in[n-2]} p_i |E_T(i)|\right) + (p_{n-1}+p_n)(|E_S(n-1)|+1) \\
&= \left(\sum_{i\in[n-1]} p_i |E_T(i)|\right) + p_{n-1} + p_n \\
&= \mathbb{E}[S] + p_{n-1} + p_n
\end{aligned}
$$

and similarly $\mathbb{E}[T'] = \mathbb{E}[S'] + p_{n-1} + p_n$. Now if $\mathbb{E}[T'] < \mathbb{E}[T]$, then $\mathbb{E}[S'] < \mathbb{E}[S]$ which contradicts our inductive hypothesis that S is optimal for any distribution over $[n-1]$. Thus, T is optimal for any distribution over $[n]$.

□

2.3 Kraft's Inequality

Theorem (Kraft's Inequality)**.** *Given a sequence $\ell_1, \ldots, \ell_n \in \mathbb{Z}^{\geq 0}$, a prefix-free encoding $E : [n] \to \{0,1\}^*$ satisfying $E(i) \in \{0,1\}^{\ell_i}$ exists if and only if $\sum_{i\in[n]} 2^{-\ell_i} \leq 1$.*

Proof. Let ℓ_{max} be $\max_{i\in n} \ell_i$ and rewrite our inequality as

$$\sum_{i\in[n]} 2^{\ell_{max}-\ell_i} \leq 2^{\ell_{max}}$$

First, we show that given a sequence $\ell_1, \ldots \ell_n$ satisfying $\sum_{i\in[n]} 2^{\ell_{max}-\ell_i} \leq 2^{\ell_{max}}$, we can construct a prefix free encoding $E : [n] \to \{0,1\}^*$ with the desired

encoding lengths.

Draw binary tree for visual

We can just go in increasing order of ℓ_i. Consider a binary tree of height ℓ_{max} with $2^{\ell_{max}}$ leaf nodes. For ℓ_i, go down to the leftmost node on level ℓ_i that is not a parent of any previous encoding, and set that to be the encoding of i. All of this node's children can no longer be used as encodings (to preserve prefix freeness) and we delete exactly $2^{\ell_{max}-\ell_i}$ leaf nodes. This gives prefix freeness and desired encoding length.

To see why we can always find a node to set as the encoding, assume for contradiction we cannot for some ℓ_i. This means every node on level ℓ_i has been previously cut out by higher nodes, but then all the higher nodes have already cut out all $2^{\ell_{max}}$ leaf nodes, and we get a contradiction with the assumption that $\sum_{i\in[n]} 2^{\ell_{max}-\ell_i} \leq 2^{\ell_{max}}$.

Now we show that if a prefix free encoding $E : [n] \to \{0,1\}^*$ with $E(i) \in \{0,1\}^{\ell_i}$ exists, then

$$\sum_{i\in[n]} 2^{\ell_{max}-\ell_i} \leq 2^{\ell_{max}}$$

Draw big square as visual, with cutting out regions of the square of size based on ℓ_i.

Now we consider the space of all $2^{\ell_{max}}$ strings. Each encoding of length ℓ_i cuts out $2^{\ell_{max}-\ell_i}$ (non-overlapping) strings. To see why the regions are non-overlapping, suppose there is some string which has two different encodings as prefixes. One of these encodings must be a prefix of the other.

We can only cut out at most $2^{\ell_{max}}$ strings, giving us the desired inequality.

□

Kraft's inequality lets us show that Huffman codes are in fact tight in their efficiency. Suppose we want to encode $[n]$. I propose encoding i with a string

of length $\lceil \log \frac{1}{p_i} \rceil$.

Kraft's inequality gives

$$\begin{aligned}\sum_{i \in [n]} 2^{-\lceil \log \frac{1}{p_i} \rceil} &\leq \sum_{i \in [n]} 2^{-\log \frac{1}{p_i}} \\ &= \sum_{i \in [n]} p_i \\ &= 1\end{aligned}$$

So such a code must exist. The performance is

$$\sum_{i \in [n]} p_i \lceil \log \tfrac{1}{p_i} \rceil \leq \sum_{i \in [n]} p_i (\log(\tfrac{1}{p_i}) + 1) = H(X) + 1$$

Since we know that the Huffman code is optimal, the Huffman code has expected encoding length at most $H(X) + 1$, so Huffman codes give codes with expected encoding length between $H(X)$ and $H(X) + 1$ which is great!

Can we iterate compression? i.e. compress our compression?

Generally, no, because compression schemes usually aim to be approximately uniformly distributed on their output length.

2.4 Lempel-Ziv Universal Encoding

What happens if we want to encode a stream without knowing the prior over the encoding?

The idea will be take advantage of the fact that "whatever the distribution is, there will be chunks that we tend to see pretty often."

Lempel-Ziv encoding aims to split a long (bit) string w into some m chunks, where

$$w = s_1 \circ s_2 \circ \ldots s_m$$
$$s_i = s_{j_i} \circ b_i$$

for some bit b_i and $j_i < i$. In other words, each chunk is a preceding chunk plus one extra bit. We can then simply encode the list $((j_1, b_1), (j_2, b_2), \ldots (j_m, b_m))$ in some prefix free way.

Lempel-Ziv loves to be used to encode hidden markov models.