In the last few lectures, we examined various ways of modelling or estimating probability functions, both joint probabilities and conditional probabilities. We looked at how to model conditional probabilities of order k given frequency counts, and how we might handle the zero frequency problem. We also looked at PPM which is a clever way for estimating conditional probability of the next symbol, given the previous symbols where we allow the order k to be determined on the fly. Notice that with these methods, we can compute not only the probability of the next symbol appearing, but we can also compute the probability of the entire sequence by treating the conditional probabilities as a Markov chain.

Let's next turn to a important coding method that takes *as given* a method of computing the joint and conditional probabilities and uses the probabilities to encode a sequence.

Introduction to arithmetic coding

We are considering a sequence of n symbols drawn from some alphabet $\{A_1, A_2, \ldots, A_N\}$. To simplify the notation, we assume the alphabet is $\{1, 2, \ldots, N\}$. Consider an enumeration of all N^n such sequences. We would like to define a code on this set of sequences, i.e. one codeword per sequence. But we would like to do so without having to write out *all* the codewords (as in Huffman coding) since there are too many codewords (N^n).

The ordering of sequences (called the *lexicographic ordering*) is the usual ordering of numbers. For example, take the following pairs of sequences of length $n = 5$ and assume $N = 3$.

$$(2, 1, 1, 1, 1) < (2, 1, 1, 1, 3)$$

$$(1, 2, 3, 2, 1) < (1, 3, 1, 1, 1)$$

$$(2, 2, 3, 2, 1) < (2, 3, 1, 1, 1)$$

We can picture this ordering using a list of the N^n sequences:

$$\begin{array}{ccccccc}
1 & 1 & 1 & 1 & \cdots & 1 & 1 \\
1 & 1 & 1 & 1 & \cdots & 1 & 2 \\
\vdots & & & \vdots & & & \vdots \\
1 & 1 & 1 & 1 & \cdots & 1 & N \\
1 & 1 & 1 & 1 & \cdots & 2 & 1 \\
\vdots & & & \vdots & & & \vdots \\
N & N & N & N & \cdots & N & N
\end{array}$$

The sum of probabilities of all these sequences is 1.

The N^n sequences are ordered, so we can define the *cumulative distribution function*:

$$F(i_1,\ i_2,\ \ldots, i_n) = \sum_{(j_1 \ldots j_n)\ \leq\ (i_1 \ldots i_n)} p(j_1, j_2, \ldots, j_n)$$

that is, the summation is taken over all sequences $(j_1 \ldots j_n)$ up to and including $(i_1, i_2, \ldots, i_n)$ in the ordering. Note that this function $F(X_1, \ldots, X_n)$ maps the list of sequences above to a sequence of points in the interval $[0, 1]$.

Notation: For a particular sequence $\vec{x} = (i_1, i_2, \ldots, i_n)$ that we wish to encode, we write:

$$p(\vec{x}) = p(i_1, i_2, \ldots, i_n)$$

$$F(\vec{x}) = F(i_1, i_2, \ldots, i_n).$$

Let $pred(\vec{x})$ be the predecessor of $\vec{x}$, that is, the sequence that comes just before $\vec{x}$ in the lexicographic order. For example,

$$pred(2, 1, 3, 5, 7) = (2, 1, 3, 5, 6)$$

$$pred(2, 1, 3, 1, 1) = (2, 1, 2, N, N)$$

Notice that

$$F(\vec{x}) - F(pred(\vec{x})) = p(\vec{x})$$

The basic idea of arithmetic coding is this: for any $\vec{x} = (i_1, i_2, \ldots, i_n)$, the codeword is a sequence of bits $b_1 b_2 \ldots b_{\lambda(x)}$ such that

$$\sum_{i=1}^{\lambda(\vec{x})} b_i \; 2^{-i} \;\; \in \;\; [\; F(pred(\vec{x}), F(\vec{x}) \;)$$

Note that this interval is closed at the left and open at the right. Thus, the intervals for all N^n possible sequences are disjoint and span $[0, 1)$.

For the sequence $\vec{x}$, define the *tag* to be number in [0,1], specifically,

$$T(\vec{x}) \;\equiv\; F(pred(\vec{x})) + \frac{p(\vec{x})}{2} \; .$$

The tag can be written equivalently as

$$T(\vec{x}) \equiv \frac{F(pred(\vec{x})) + F(\vec{x})}{2} \qquad \text{or} \qquad T(\vec{x}) \equiv F(\vec{x}) - \frac{p(\vec{x})}{2} \; .$$

Since $T(\vec{x}) \in [0, 1]$, we can write $T(\vec{x})$ as a sum of powers of 2 (that is, in binary) as

$$T(\vec{x}) = \sum_{i=1}^{\infty} b_i \; 2^{-i}$$

where b_i is either 0 or 1.

Note that the b_i sequence is not necessarily unique, for example,

$$\frac{1}{2} = \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \ldots$$

i.e. the following two infinite b_i sequences (binary numbers) are identical

$$.100000\ldots \;\; = \;\; .0111111111\ldots$$

This is just the binary equivalent of the decimal fact that:

$$0.9999999\ldots \;\; = \;\; 1.00000\ldots$$

This is not a problem for us. We don't care about uniqueness of the bits b_i. Existence is all we need.

We encode $\vec{x}$ using the first $\lambda(\vec{x})$ bits $b_1 \ldots b_{\lambda(\vec{x})}$ of the binary representation of the tag. How big should $\lambda(\vec{x})$ be? The main idea is that these bits should specify the interval in which the tag lies. We will choose $\lambda(\vec{x})$ so that:

$$F(pred(\vec{x})) \;\leq\; \sum_{i=1}^{\lambda(\vec{x})} b_i \; 2^{-i} \;\leq\; T(\vec{x}) \; .$$

To determine $\lambda(\vec{x})$, consider the following bound:

$$\begin{aligned}
T(\vec{x}) &= \sum_{i=1}^{\infty} b_i \; 2^{-i} \\
&= \sum_{i=1}^{\lambda(\vec{x})} b_i \; 2^{-i} + \sum_{i=\lambda(\vec{x})+1}^{\infty} b_i \; 2^{-i} \\
&\leq \sum_{i=1}^{\lambda(\vec{x})} b_i \; 2^{-i} + 2^{-\lambda(\vec{x})} \sum_{j=1}^{\infty} 2^{-j} \qquad\qquad \text{since } b_i \text{ is 0 or 1} \\
&= \sum_{i=1}^{\lambda(\vec{x})} b_i \; 2^{-i} + 2^{-\lambda(\vec{x})}
\end{aligned}$$

We would like to choose $\lambda(\vec{x})$ so that

$$T(\vec{x}) - \sum_{i=1}^{\lambda(\vec{x})} b_i \; 2^{-i} \;\leq\; \frac{p(\vec{x})}{2}$$

But we know that

$$T(\vec{x}) - \sum_{i=1}^{\lambda(\vec{x})} b_i \; 2^{-i} \;\leq\; 2^{-\lambda(\vec{x})}$$

Thus, it is sufficient that we choose $\lambda(\vec{x})$ so that:

$$2^{-\lambda(x)} \;\leq\; \frac{p(x)}{2}.$$

We can achieve this by defining:

$$\lambda(x) \;\equiv\; \lceil \log(\frac{2}{p(x)}) \rceil.$$

In summary, the codeword for $\vec{x}$ to be

$$C(\vec{x}) \;\equiv\; b_1 b_2 \ldots b_{\lambda(x)}$$

Note that the *the lower the probability* $p(\vec{x})$, the smaller is the interval $[F(pred(\vec{x}), T(x)]$ and thus the more bits are (typically) required to specify the interval. This corresponds to our intuition about how compression works. Low probability events should have longer codewords.

Average codeword length

A codeword is defined for each of the N^n sequences. What is the average codeword length ?

$$\begin{aligned}\overline{\lambda} = \sum_{\vec{x}} p(\vec{x})\ \lambda(\vec{x}) &= \sum_{\vec{x}} p(\vec{x})\ \lceil \log(\frac{2}{p(\vec{x})}) \rceil \\ &\leq \sum_{\vec{x}} p(\vec{x})\ (\log(\frac{1}{p(\vec{x})}) + 2) \\ &= H(X_1, \ldots, X_n) + 2\end{aligned}$$

Yipes, that is good! The average code length is at most 2 bits more than the entropy. Even if we were to construct a Huffman code over all N^n possible sequences (which is impractical, since the number of codewords is astronomical), the average code length would still be greater than the entropy. We are nearly as good as the Huffman code.