Models Assign Probabilities to Sequences of Symbols

Any way of producing predictive probabilities for each symbol in turn will also assign a probability to every *sequence* of symbols. (We'll suppose sequences are terminated by a special EOF symbol.)

We just multiply together the predictive probabilities as we go.

For example, the string "CAT" has probability

$$\begin{aligned}&P(X_1 = \text{`C'})\\ &\quad\times P(X_2 = \text{`A'} \,|\, X_1 = \text{`C'})\\ &\quad\times P(X_3 = \text{`T'} \,|\, X_1 = \text{`C'}, X_2 = \text{`A'})\\ &\quad\times P(X_4 = \text{EOF} \,|\, X_1 = \text{`C'}, X_2 = \text{`A'}, X_3 = \text{`T'})\end{aligned}$$

where the probabilities above are the ones used to code each individual symbol.

With an optimal coding method, the number of bits used to encode the entire sequence will be close to the log of one over its probability.

Probabilities of Sequences with the Laplace Model

The general form of the "add one to all the counts" method uses the following predictive distributions:

$$P(X_n = a_i) \;=\; \frac{1 + \text{Number of earlier occurrences of } a_i}{I + n - 1}$$

where I is the size of the source alphabet. This is called "Laplace's Rule of Succession".

So the probability of a sequence of n symbols is

$$\frac{(I-1)!}{(I+n-1)!} \prod_{i=1}^{I} n_i!$$

where n_i is the number of times a_i occurs in the sequence.

It's much easier to code one symbol at a time (using arithmetic coding) than to encode a whole sequence at once, but we can see from this what the model is really saying about which sequences are more likely.

Models With Multiple Contexts

So far, we've looked at models in which the symbols would be *independent*, if we knew what their probabilities were.

If we don't know the probabilities, our predictions do depend on previous symbols, but the symbols are still "exchangeable" — their order doesn't matter.

Very often, this isn't right: The probability of a symbol may depend on the *context* in which it occurs — eg, what symbol precedes it.

Example: "U" is much more likely after "Q" (in English), than after another "U".

Probabilities may also depend on position in the file, though modeling this is less common.

Example: Executable program files may have machine instructions at the beginning, and symbols to help with debugging at the end.

Markov Sources

An *K-th order Markov source* is one in which the probability of a symbol depends on the preceding K symbols.

We can write the probability of a sequence of symbols, $X_1, X_2, \ldots, X_n$ from such a source with $K = 2$ as follows (assuming we know all the probabilities):

$$\begin{aligned}&P(X_1 = a_{i_1}, X_2 = a_{i_2}, \ldots, X_n = a_{i_n})\\ &\quad= P(X_1 = a_{i_1}) \times P(X_2 = a_{i_2} \mid X_1 = a_{i_1})\\ &\qquad\times P(X_3 = a_{i_3} \mid X_1 = a_{i_1}, X_2 = a_{i_2})\\ &\qquad\times P(X_4 = a_{i_4} \mid X_2 = a_{i_2}, X_3 = a_{i_3})\\ &\qquad\quad\cdots\\ &\qquad\times P(X_n = a_{i_n} \mid X_{n-2} = a_{i_{n-2}}, X_{n-1} = a_{i_{n-1}})\\ &\quad= P(X_1 = a_{i_1}) \times P(X_2 = a_{i_2} \mid X_1 = a_{i_1})\\ &\qquad\times M(i_1, i_2, i_3) M(i_2, i_3, i_4) \cdots M(i_{n-2}, i_{n-1}, i_n)\end{aligned}$$

Here, $M(i, j, k)$ is the probability of symbol a_k when the preceding two symbols were a_i and a_j.

Adaptive Markov Models

Some sources may really be Markov of some order K, but usually not.

We can nevertheless use a Markov *model* for a source as the basis for data compression.

Usually, we don't know the "transition probabilities", so we estimate them adaptively, using past frequencies. Eg, for $K = 2$, we accumulate frequencies in each context, $F(i, j, k)$, and then use probabilities

$$M(i, j, k) = F(i, j, k) / \sum_{k'} F(i, j, k')$$

After encoding symbol a_k in context a_i, a_j, we increment $F(i, j, k)$.

A K-th order Markov model has to handle the first $K-1$ symbols specially. One approach: Imagine that there are K symbols before the beginning with some special value (eg, space).

Markov Models of Order 0, 1, and 2 Applied to English Text

I applied adaptive Markov models of order 0, 1, and 2, using arithmetic coding, to three English text files (Latex), of varying sizes.

Markov Model of Order 0

Uncompressed file size	Compressed file size	Compression factor	Bits per character
2344	1431	1.64	4.88
20192	12055	1.67	4.78
235215	137284	1.71	4.67

Markov Model of Order 1

Uncompressed file size	Compressed file size	Compression factor	Bits per character
2344	1750	1.34	5.97
20192	11490	1.76	4.55
235215	114494	2.05	3.89

Markov Model of Order 2

Uncompressed file size	Compressed file size	Compression factor	Bits per character
2344	2061	1.14	7.03
20192	13379	1.51	5.30
235215	111408	2.11	3.79

How Large an Order Should be Used?

We can see a problem with these results.

A Markov model of high order works well with long files, in which most of the characters are encoded after good statistics have been gathered.

But for small files, high-order models don't work well — most characters occur in contexts that have occurred only a few times before, or never before.

For the smallest file, the zero-order model with only one context was best, even though we know that English has strong dependencies between characters!