1 Arithmetic Coding

When we discussed variable length source codes, and the optimal Huffman algorithm for constructing them, we concluded by pointing out two practical and theoretical problems with Huffman codes.

These defects are rectified by *Arithmetic codes*, first invented by Rissanen and by Pasco, and subsequently promoted by Witten, Neal and Cleary. In an arithmetic code, the probabilistic modelling is clearly separated from the encoding operation. As each symbol is produced by the source, our probabilistic model of that source supplies a predictive distribution over all possible symbols. If we choose to model the source as producing i.i.d. symbols, then the predictive distribution is the same every time; but it will be equally convenient to handle complex adaptive models.

CONCEPTS

Let the source alphabet be $\mathcal{A}_X = \{a_1 \ldots a_I\}$, and let the Ith symbol a_I have the special meaning 'end of transmission'. I assume that a model is provided which assigns a predictive probability distribution over a_i, $P(x_n{=}a_i|x_1, \ldots, x_{n-1})$, and that there is a source that spits out the $x_1, x_2, \ldots, x_n, \ldots$. The source emphatically does not necessarily produce i.i.d. symbols.

The receiver has access to an identical program that produces the same predictive probability distribution $P(x_n{=}a_i|x_1, \ldots, x_{n-1})$ and uses it to interpret the received message.

A binary transmission can be viewed as defining an interval within the real line from 0 to 1. For example, the transmission 01 is interpreted as a binary real number 0.01..., which corresponds to the interval $[0.01, 0.10)$ (binary), *i.e.*, the interval $[0.25, 0.50]$ (base ten). A longer transmission 01101 corresponds to the interval $[0.01101, 0.01110)$; because 01101 has the first string 01 as a prefix, the new interval is a sub-interval of the interval $[0.01, 0.10]$. A one megabyte binary file ($2^{2}3$ bits) is thus viewed as specifying a number between 0 and 1 to a precision of about 10^7 decimal places.

Similarly, we can divide the real line [0,1] into I intervals of lengths equal to the probabilities $P(x_1{=}a_i)$. The first interval, which we will call 'a_1', is $[0, P(x_1{=}a_1)]$; subsequent intervals are defined in terms of the cumulative probability

$$R_{n,i|x_1,\ldots,x_{n-1}} \equiv \sum_{i'=1}^{i} P(x_n{=}a_{i'}|x_1, \ldots, x_{n-1}).$$

The interval 'a_2' is $[R_{1,1}, R_{1,2}]$, and the interval a_I is $[R_{1,(I-1)}, 1.0]$. We may then take the interval a_i and subdivide it into intervals denoted $a_i a_1, a_i a_2, \ldots a_i a_I$, such that the length pf $a_i a_j$ is proportional to $P(x_2{=}a_j|x_1{=}a_i)$. Indeed the length of the interval $a_i a_j$ will be precisely $P(x_1{=}a_i, x_2 = a_j) = P(x_1{=}a_i)P(x_2{=}a_j|x_1{=}a_i)$.

Iterating this procedure, the interval [0,1] can be divided into a sequence of intervals corresponding to all possible finite length strings $x_1 x_2 \ldots x_n$, such that the length of an interval is equal to the probability of the string given our model.

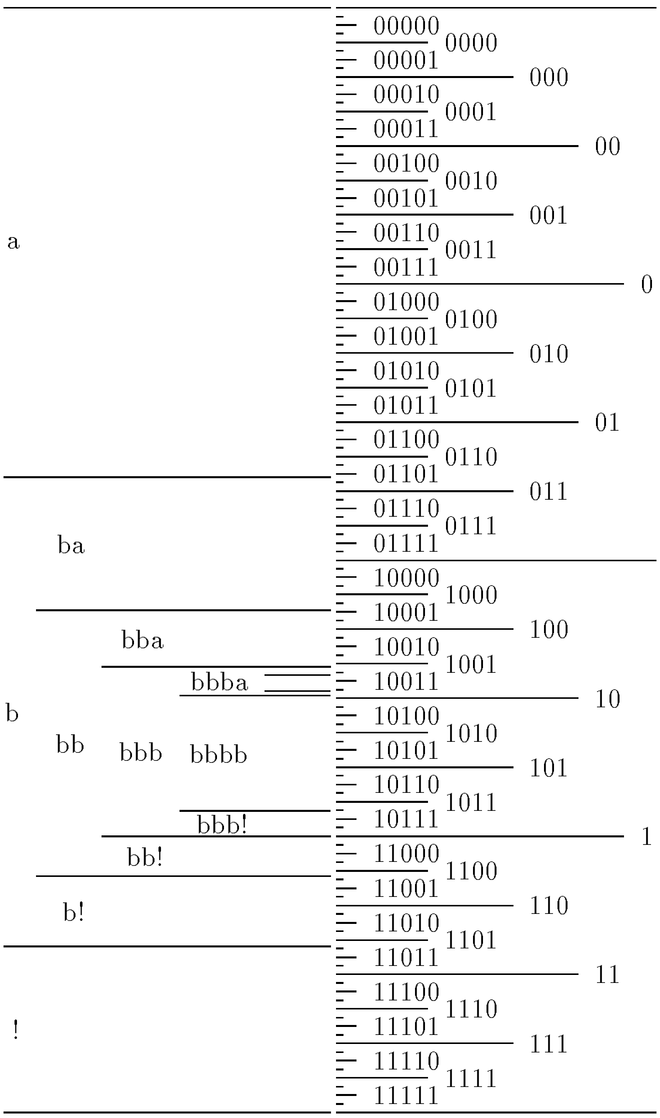
The method of coding a string $x_1 x_2 \ldots x_n$ is straightforward. We simply locate the interval corresponding to $x_1 x_2 \ldots x_n$, and send a binary string whose interval lies within that interval.

Example. Let $\mathcal{A}_X = \{a, b, !\}$, where ! is an 'end of word' symbol.

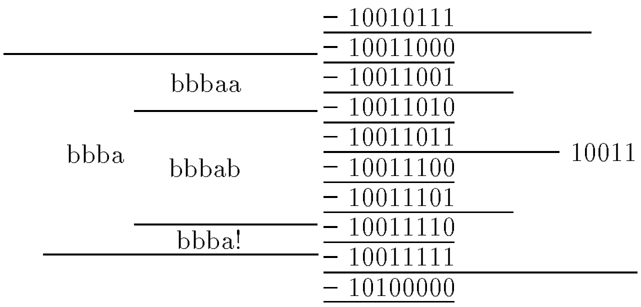
Encoding. Let the source string be '*bbba!*'. We pass along the string one symbol at a time and use our model to assign a probability over the next symbol given the string thus far. Let these probabilities be:

$P(a){=}0.425$	$P(b){=}0.425$	$P(!){=}0.150$
$P(a\|b){=}0.283$	$P(b\|b){=}0.567$	$P(!\|b){=}0.150$
$P(a\|bb){=}0.212$	$P(b\|bb){=}0.637$	$P(!\|bb){=}0.150$
$P(a\|bbb){=}0.170$	$P(b\|bbb){=}0.680$	$P(!\|bbb){=}0.150$
$P(a\|bbba){=}0.283$	$P(b\|bbba){=}0.567$	$P(!\|bbba){=}0.150$

The corresponding intervals are shown here.



When the first symbol 'b' is observed, the encoder knows that the encoded string will either start '01', '10', or '11', but it is not clear which. The encoder takes no action and examines the next symbol, which is 'b'. The interval 'bb' lies wholly within interval '1', so the encoder can write the first bit: '1'. The next symbol 'b' narrows down the interval a little, but not quite enough for it to lie wholly within interval '10'. Only when the next 'a' is read from the source can we transmit some more bits. Interval '$bbba$' lies wholly within the interval '1001', so the encoder writes '001'. Finally when the '!' arrives, we need a procedure for terminating the encoding. Magnifying the interval '$bbba$!' we note that '100111101' is wholly contained by it, so the encoding in completed by writing '11101'. The overhead required to terminate a message is never more than 2 bits, relative to the ideal message length, given the probabilistic model $\mathcal{H}$, $l(\mathbf{x}|\mathcal{H}) = \log 1/P(\mathbf{x}|\mathcal{H})$.



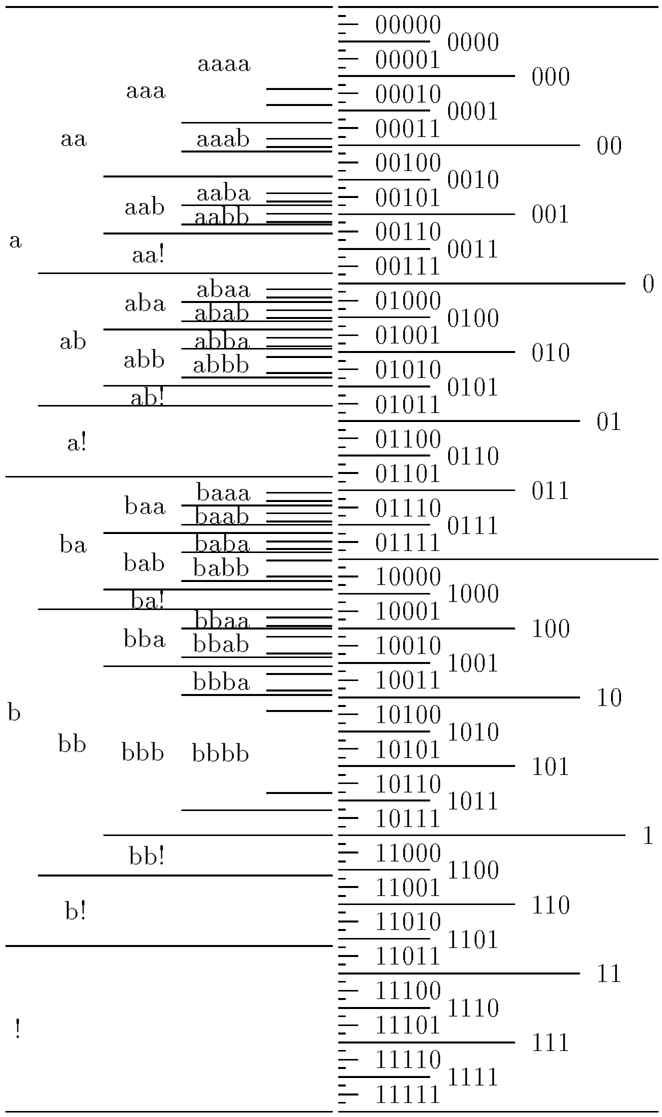
Decoding. The decoder receives the string '100111101' and passes along it one symbol at a time. The probabilities $P(a), P(b), P(!)$ are computed using the identical program used by the encoder. Once the first two bits '10' have been examined, it is clear that the original string must have been a 'b', since the interval '10' lies wholly within interval 'b'. The decoder can then use the model to compute $P(a|b), P(b|b), P(!|b)$. Continuing, we decode the second b once we reach '1001', the third b once we reach '100111', and so forth, with the unambiguous identification of '$bbba$!' once the whole string has been read. With the convention that '!' denotes the end of the message, the decoder knows to stop decoding.

THE BIG PICTURE

The technique of arithmetic coding does not force one to produce the predictive probability in any particular way, but the predictive distributions might naturally be produced by a Bayesian model.

These figures were generated using a model that always assigns a probability of 0.15 to !, and assigns to a and b probabilities proportional to 'Laplace's rule', $P_{\mathrm{L}}(a|x_1, \ldots, x_{n-1}) = (F_a + 1)/(F_a + F_b + 2)$, where $F_a(x_1, \ldots, x_{n-1})$ is the number of times that a has occurred so far, and similarly F_b. This corresponds to a simple Bayesian model that is able to cotton on to a non-equal frequency of usage of the source symbols a and b within a word. The end result will be an encoder that can 'equalize' such a non-uniform source.

The following figure displays the intervals corresponding to a large number of strings. Note that if the string so far has contained a large number of bs then the probability of b relative to a is increased, and conversely if many as occur then as are made more probable. Larger intervals, remember, require fewer bits to encode.



2 Lempel-Ziv Coding

The Lempel-Ziv algorithms, which are widely used for data compression (*e.g.*, the `compress` command in `UNIX`), are totally opposite in philosophy to Arithmetic coding. There is no separation at all between modelling and coding.

The method of compression is to replace a substring with a pointer to an earlier occurrence of the same substring.

The resulting algorithm is very fast, but its compression performance is poor by the standards set in the arithmetic coding literature.

References

WITTEN, I. H., NEAL, R. M., and CLEARY, J. G. (1987) Arithmetic coding for data compression. *Communications of the ACM* **30** (6): 520–540.