ECE 7680
Lecture AC – Arithmetic Coding

Objective: To introduce arithmetic coding

Introduction

Arithmetic coding overcomes some of the problems of Huffman coding, in particular the potential 1 bit surplus problem. It operates as a human might, using information already observed to predict what might be coming, and coding based on the prediction. In addition, the technique explicitly separates the prediction portion from the encoding portion.

In AC, a bit sequence is interpreted as an interval on the real line from 0 to 1. For example `01` is interpreted as 0.01...., which corresponds (not knowing what the following digits are) to the interval $[0.01, 0.10)$ (in binary) which is $[0.25, 0.5)$ (bse ten). (Make sure understanding on brackets.) A longer string `01101` corresponds to the interval $[0.01101, 0.01110)$. The longer the string, the shorter the interval represented on the real line.

Assume we are dealing with an alphabet $\mathcal{A} = \{a_1, \ldots, a_I\}$, where a_I is a special symbol meaning "end of transmission." The source produces the sequence $x_1, x_2, \ldots, x_n, \ldots$, and not necessarily i.i.d. We further assume (or model) that there is a predictor which computes, or estimates

$$P(x_n = a_i | x_1, x_2, \ldots, x_{n-1}),$$

which is available at both encoder and decoder.

We divide the segment $[0, 1)$ into I intervals whose lengths are equal to the probabilities $P(x_1 = a_i), i = 1, 2, \ldots, I$. The first interval is

$$[0, P(x_1 = a_1))$$

The second interval is

$$[P(x_1 = a_1), P(x_1 = a_1) + P(x_1 = a_2)),$$

and so forth. More generally, to provide for the possibility of considering other symbols than just x_1, we define the lower and upper cumulative probabilities:

$$Q_n(a_i | x_1, \ldots, x_{n-1}) = \sum_{j=1}^{i-1} P(x_n = a_j | x_1, \ldots, x_{n-1})$$

$$R_n(a_i | x_1, \ldots, x_{n-1}) = \sum_{j=1}^{i} P(x_n = a_j | x_1, \ldots, x_{n-1})$$

Then, for example, a_2 corresponds to the interval $[Q_1(a_2), R_1(a_2))$.

Now we represent the probabilities for the next symbol. Take, for example, the interval for a_1, and subdivide it into intervals $a_1a_1, a_1a_2, \ldots, a_1a_I$, so that the length of the interval for a_1a_j is *proportional* to $P(a_j|a_1)$. In fact, we take the length of the subinterval for a_1a_j to be

$$P(x_1 = a_1, x_2 = a_j) = P(x_1 = a_1)P(x_2 = a_j | x_1 = a_1)$$

Then we note that the sum of the lengths of these subintervals will be

$$\sum_j P(x_1 = a_1, x_2 = a_j) = P(x_1 = a_1),$$

which sure enough is the correct length.

More generally, we subdivide each of the intervals for $a_i a_j$ similarly to have length of

$$P(x_1 = a_i, x_2 = a_j) = P(x_1 = a_1)P(x_2 = a_j | a_1 = a_i).$$

Then, we continue subdividing each subinterval for strings of length N.

The following algorithm (Mackay, p. 151) shows how to compute the interval $[u, v)$ for the string $x_1 x_2, \ldots, x_N$. (Note: this is for demonstration purposes, since it requires infinite precision arithmetic. In practice, the algorithm is arranged so that infinite precision is not required.)

Algorithm 1 Arithmetic coding

$u = 0.0$ — lower side of interval
$v = 1.0$ — upper side of interval
$p = v - u$ — length of current interval = probability of sequence
for $n = 1$ to N {
  Compute upper and lower cumulative probabilities
  Compute $R_n(i|x_1, \ldots, x_{n-1})$ for $i = 1, 2, \ldots, I$
  Compute $Q_n(i|x_1, \ldots, x_{n-1})$ for $i = 1, 2, \ldots, I$
  $u = u + pQ_n(x_n|x_1, \ldots, x_{n-1})$ — update lower side by joint length
  $v = u + pR_n(x_n|x_1, \ldots, x_{n-1})$ — update upper side by joint length
  $p = v - u$ — compute new length = new probability
}

In encoding, the interval is subdivided for each new symbol. To encode the string $x_1, x_2, \ldots, x_N$, we send the binary string whose interval lies *within* the interval determined by the sequence.

Example 1 (Mackay, p. 151) Suppose a bent coin with outcomes `a` and `b` is to be transmitted. We throw in the outcome □ to represent 'end of transmission'.

Suppose we want to code `bbba`□. We obtain the following table (the reasoning behind the probability values will be discussed below).

Context (sequence thus far)	Probability of next symbol		
	$P(\texttt{a}) = 0.425$	$P(\texttt{b}) = 0.425$	$P(\square) = 0.15$
`b`	$P(\texttt{a}\|\texttt{b}) = 0.28$	$P(\texttt{b}\|\texttt{b}) = 0.57$	$P(\square\|\texttt{b}) = 0.15$
`bb`	$P(\texttt{a}\|\texttt{bb}) = 0.21$	$P(\texttt{b}\|\texttt{bb}) = 0.64$	$P(\square\|\texttt{bb}) = 0.15$
`bbb`	$P(\texttt{a}\|\texttt{bbb}) = 0.17$	$P(\texttt{b}\|\texttt{bbb}) = 0.68$	$P(\square\|\texttt{bbb}) = 0.15$
`bbba`	$P(\texttt{a}\|\texttt{bbba}) = 0.28$	$P(\texttt{b}\|\texttt{bbba}) = 0.57$	$P(\square\|\texttt{bbba}) = 0.15$

The subdivision of the interval is portrayed in the following figure (Mackay, fig. 4.3)

.

When the first `b`is observed, the encoded knows that the encoded string will start as `01`, `10`, or `00` (see the figure), but doesn't know which. No output symbol is produced. For the next symbol `b`, the interval lies wholly within the interval `1`, so the encoder outputs `1`. However, no additional bits can be sent yet. At the third symbol `b`, there is still not quite enough information (but very close)! When the `a`is read, the interval lies entirely within `1001`, which can now be output. When □, we obtain the division shown on the right. The interval `10011101` is contained in the interval for `bbba`□, so we send that sequence.

The **decoder** receives `100111101` and parses it one bit at a time. The probabilities are built up exactly as before. Once `10` have been parsed, it is known that the original string must have contained a `b`, since `10` lies within the interval `b`. Knowing this, the decoding computes $P(\mathtt{a}|\mathtt{b}), P(\mathtt{b}|\mathtt{b})$ and $P(\square|\mathtt{b})$, conditioned upon the symbol that has been received, then deduce the boundaries of the intervals `ba`, `bb` and `b`$\square$. Eventually, the second `b`is decoded, which is used to condition the probabilities. This continues until decoding is complete. When the $\square$is reached the decoder knows that the end of file has been reached. $\square$

One of the benefits of arithmetic coding is that the worst case redundancy *for an entire bit string* (which may, for example, consist of an entire file) is **at most two bits**, assuming the probabilistic model is correct. Given a probabilistic model $\mathcal{H}$, the ideal message length for a sequence $\mathbf{x}$ is $l(\mathbf{x}|\mathcal{H}) = -\log[P(\mathbf{x}|\mathcal{H})]$. Suppose that $P(\mathbf{x}|\mathcal{H})$ is *just barely* between two binary intervals. Then the next smaller binary intervals contained in $P(\mathbf{x}|\mathcal{H})$ are smaller by a factor of 4. This factor of 4 corresponds to $\log_2 4 = 2$ bits overhead worst case.

Probability models

Performance of the AC depends on having a good model for the source probabilities. The better the model, the better it might be expected that the code performs. In principle, any probabilistic model can be used. We mention here some useful concepts in developing one.

Suppose, as before, we deal with the case of independent events. We have outcomes `a`, `b`, and $\square$, with probabilities $p_\mathtt{a}, p_\mathtt{b}$ and $p_\square$. Let l be the number of outcomes (number of coin tosses). $p_\mathtt{a}$ could be anywhere in the range $[0, 1]$, and we may not have any predisposition toward one value. We model this ambivalence by saying that

$$P(p_\mathtt{a}) = 1 \qquad \text{for } p_\mathtt{a} \in [0, 1].$$

That is, it is uniformly distributed. This is a *prior probability*. If we had some predisposition about $p_\mathtt{a}$, this could be incorporated into the prior model (using something like a β distribution, for example).

The whole point of Bayesian estimation (which is what we find we are talking about here) is to merge our prior inclinations in with the observations. This is a problem of inference, which we can state this way: given a sequence of F bits, of which $F_\mathtt{a}$ are `a`s and $F_\mathtt{b}$ are `b`s, infer $p_\mathtt{a}$. The inference is accomplished by the posterior ("after") — the probability of $p_\mathtt{a}$ *after* a measurement $\mathbf{s}$ is made. We write

$$P(p_\mathtt{a}|\mathbf{s}, F) = \frac{P(\mathbf{s}|p_\mathtt{a}, F)P(p_\mathtt{a})}{P(\mathbf{s}|F)}.$$

Now why this? Well, we can write down the conditional probability in the numerator:

$$P(\mathbf{s}|p_\mathtt{a}, F) = p_\mathtt{a}^{F_\mathtt{a}}(1 - p_\mathtt{a})^{F_\mathtt{a}}$$

(describe why). As we have seen elsewhere, it seems that the conditioning is always easiest they way you don't need it. We also find

$$P(\mathbf{s}|F) = \int P(\mathbf{s}|p_\mathtt{a}, F)P(p_\mathtt{a})dp_\mathtt{a} = \int p_\mathtt{a}^{F_\mathtt{a}}(1 - p_\mathtt{a})^{F_\mathtt{b}}P(p_\mathtt{a})dp_\mathtt{a} = \frac{\Gamma(F_\mathtt{a} + 1)\Gamma(F_\mathtt{b} + 1)}{\Gamma(F_\mathtt{a} + F_\mathtt{b} + 2)} = \frac{F_\mathtt{a}!F_\mathtt{b}!}{F_\mathtt{a} + F_\mathtt{b} + 1)!}$$

So we could infer $p_\mathtt{a}$ as the most probable value (the maximizer) of the posterior. For example, we find $P(p_\mathtt{a}|\mathbf{s} = \mathtt{aba}, F = 3) \propto p_\mathtt{a}^2(1-p_\mathtt{a})$, with maximum of $p_\mathtt{a} = 2/3$. Or we could infer based on the mean, which is 3/5.

We also want to be able to make predictions. Given a sequence $\mathbf{s}$ of length F as evidence we find the prediction of drawing an $\mathtt{a}$as

$$P(\mathtt{a}|\mathbf{s},F) = \int P(\mathtt{a}|p_a tt)P(p_a|\mathbf{s},F)dp_{\mathtt{a}}.$$

Note that in this case, we are using the entire posterior probability, so we incorporate all of our uncertainty about p_a. We also have $P(\mathtt{a}|p_{\mathtt{a}}) = p_{\mathtt{a}}$ (by its definition), so our predictor is

$$P(\mathtt{a}|\mathbf{s},F) = \int p_a \frac{p_{\mathtt{a}}^{F_{\mathtt{a}}}(1-p_{\mathtt{a}})^{F_{\mathtt{a}}}}{P(\mathbf{s}|F)} dp_{\mathtt{a}} = \frac{F_{\mathtt{a}}+1}{F_{\mathtt{a}}+F_{\mathtt{b}}+2}.$$

This update rule is known as Laplace's rule, and is the rule that was used in the coder above.

We could write this as

$$P_L(a|x_1,\ldots,x_{n-1}) = \frac{F_{\mathtt{a}}+1}{\sum_i (F_i+1)}$$

Another model, known as the Dirichlet model, is more "responsive":

$$P_D(a|x_1,\ldots,x_{n-1}) = \frac{F_{\mathtt{a}}+\alpha}{\sum_i (F_i+\alpha)}$$

Typically, α is small, like 0.01.

This is not the only possible rule, and doesn't necessarily take into account the relationship that might exist between dependent variables.

Another application of AC

Mackay suggests another application as a way of generating a sample from a model, a sophisticated random number generator. As a simple-minded example, divide the unit interval into lengths p_i. Pick a point at random on the unit interval. The probability that your pin lies in interval i is p_i.

More generally, to generate a sample of bits from some random model, feed ordinary random bits into an arithmetic decoder for that model. This corresponds to picking a point at random in $[0,1]$. The decoder will select a string at random from the modeled distribution, and will use the smallest number of random numbers to do the job.