Arithmetic Coding

Arithmetic coding extends the Shannon-Fano-Elias coding technique to allow letter-by-letter encoding and decoding that compresses very close to the entropy. Arithmetic coding is more elegant than the cumbersome generation of a the Huffman code for $X^n$. In particular, it does not require knowledge of the distribution for a given symbol until that symbol is to be encoded.

The version of Arithmetic Coding presented here is described in more detail in *Text Compression* by Bell, Cleary, and Witten, published by Prentice Hall. This is an excellent book on lossless compression techniques.

Consider the alphabet $X = \{ a, b, c, d \}$. We require an additional end-of-text character $\langle \text{eot} \rangle$. Each letter $x$, including $\langle \text{eot} \rangle$, is assigned a probability and a half-open segment of the unit interval with length equal to its probability. $L(x)$ and $U(x)$ represent the lower and upper limits of the segment associated with $x$.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$P(x)$</th>
<th>segment</th>
<th>$L(x)$</th>
<th>$U(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>0.3</td>
<td>$[0,0.3)$</td>
<td>0</td>
<td>0.3</td>
</tr>
<tr>
<td>$b$</td>
<td>0.3</td>
<td>$[0.3,0.6)$</td>
<td>0.3</td>
<td>0.6</td>
</tr>
<tr>
<td>$c$</td>
<td>0.1</td>
<td>$[0.6,0.7)$</td>
<td>0.6</td>
<td>0.7</td>
</tr>
<tr>
<td>$d$</td>
<td>0.2</td>
<td>$[0.7,0.9)$</td>
<td>0.7</td>
<td>0.9</td>
</tr>
<tr>
<td>$\langle \text{eot} \rangle$</td>
<td>0.1</td>
<td>$[0.9,1)$</td>
<td>0.9</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1: Probabilities and segments for alphabet in arithmetic coding example.

Each letter is thus assigned a segment of the unit interval. To transmit $a$, send any number in the segment $[0,0.3)$.

Just as individual letters have segments associated with them, each possible string of letters also has a segment associated with it. The segment associated with a string is computed letter-by-letter. Each successive letter chooses the sub-segment of the current segment associated with the previous letters by treating the current segment as if it were the unit interval.

Note that any point in the interval corresponds to an infinite string of characters since each point lies within an infinite sequence of subsegments. In practical applications where a finite string of letters is being transmitted, the $\langle \text{eot} \rangle$ character instructs the decoder to stop.
The encoding and decoding procedures are best described by an example. We will now encode the sequence \( \text{bad} \langle \text{eot} \rangle \). Encoding consists of two operations. The first operation is to identify the sub-segment that corresponds to the desired string. The second operation is to send a number in that sub-segment so as to use the least possible number of bits.

The algorithm for identifying the new sub-segment \([L_{\text{new}}, U_{\text{new}}]\) corresponding to the new symbol \(x\) in a string whose previous symbols correspond to sub-segment \([L_{\text{previous}}, U_{\text{previous}}]\) consists of the following computations.

\[
\begin{align*}
\text{range} &= U_{\text{previous}} - L_{\text{previous}} \\
L_{\text{new}} &= L_{\text{previous}} + \text{range} \times L(x) \\
U_{\text{new}} &= L_{\text{previous}} + \text{range} \times U(x)
\end{align*}
\]

We apply this algorithm to find the sub-segment corresponding to \( \text{bad} \langle \text{eot} \rangle \).

1. From Table ?? we see that the segment associated with \( b \) is \([0.3, 0.6)\).

2. Applying the algorithm to \( a \) with previous segment \([0.3, 0.6)\), we find that the new sub-segment (corresponding to \( ba \)) is \([0.3, 0.39)\).

3. Applying the algorithm to \( d \) with previous segment \([0.3, 0.39)\), we compute the sub-segment corresponding to \( \text{bad} \) to be \([0.363, 0.381)\).

4. Finally, applying the algorithm to \( \langle \text{eot} \rangle \) with previous segment \([0.363, 0.381)\), we compute the final sub-segment to be \([0.3792, 0.381)\).

Thus, any point that is in \([0.3792, 0.381)\) will be unambiguously decoded as \( \text{bad} \langle \text{eot} \rangle \) (because decoding ceases after \( \langle \text{eot} \rangle \) is detected).

We want to send the value in \([0.3792, 0.381)\) that can be represented with the smallest number of bits. (All bits that are not transmitted are assumed to be zero.) The answer for this particular case is .011000011 which corresponds to decimal 0.380859375.
We now present an algorithm that finds an efficient binary representation of our interval. The algorithm is designed to work on the intermediate intervals so that bits can be sent before all symbols have been encoded. After each symbol has been encoded, as many bits as possible are transmitted and the segment is scaled to reflect the fact that these bits have been sent. This aspect is important because it allows a fixed precision implementation of arithmetic coding.

Here is the algorithm:

1. Encode a symbol (unless \texttt{<eot>} has been encoded) to determine either the original segment or a subsequent subsegment.

2. While $U_{\text{new}} < 0.5$ or $L_{\text{new}} > 0.5$:
   
   - If $U_{\text{new}} < 0.5$, send a zero and rescale the subsegment as follows:
     \begin{align*}
     L_{\text{new}} &= L_{\text{new}} \times 2 \\
     U_{\text{new}} &= U_{\text{new}} \times 2
     \end{align*}

   - If $L_{\text{new}} > 0.5$, send a one and rescale the subsegment as follows:
     \begin{align*}
     L_{\text{new}} &= (L_{\text{new}} - 0.5) \times 2 \\
     U_{\text{new}} &= (U_{\text{new}} - 0.5) \times 2
     \end{align*}

3. If \texttt{<eot>} has been sent, send a final one and exit. Otherwise, go to step 1.

If we can send a bit according to step 2, we will have to send that bit to unambiguously specify the segment if interest. If we have encoded \texttt{<eot>} and cannot send a bit according to step 2, the current segment straddles 0.5. By sending a terminating 1 (which by convention is followed by all zeros), we are sending the value 0.5, which is in the interval as desired.
Now we apply this algorithm to $bad\langle eot \rangle$, specifying the segment at each stage and when each bit gets transmitted.

1. Encode $b$, original segment is $[0.3, 0.6)$. Can’t send a bit.
2. Encode $a$. New subsegment is $[0.3, 0.39)$. Can send a 0.
3. Send a 0. New subsegment is $[0.6, 0.78)$. Can send a 1.
4. Send a 1. New subsegment is $[0.2, 0.56)$. Can’t send a bit.
5. Encode $d$. New subsegment is $[0.452, 0.524)$. Can’t send a bit.
6. Encode $\langle eot \rangle$. New subsegment is $[0.5168, 0.524)$. Can send a 1.
7. Send a 1. New subsegment is $[0.0336, 0.0448)$. Can send a 0.
8. Send a 0. New subsegment is $[0.0672, 0.096)$. Can send a 0.
9. Send a 0. New subsegment is $[0.1344, 0.192)$. Can send a 0.
10. Send a 0. New subsegment is $[0.2688, 0.384)$. Can send a 0.
11. Send a 0. New subsegment is $[0.5376, 0.768)$. Can send a 1.
12. Send a 1. New subsegment is $[0.0752, 0.536)$. Can’t send a bit.
13. Send final 1 to terminate.
Finally, we need a decoding algorithm that recovers the transmitted string from the received binary sequence. In real systems, this is done using fixed point (or even binary) arithmetic. Furthermore, decoding is done as the bits arrive decoding each symbol as soon as its segment is unambiguously identified.

However, for simplicity of exposition, we will be content to recover the string from the final received value $v$, which we assume is a floating point decimal number in $[0, 1)$.

Our algorithm for decoding:

1. Find the interval containing the $v$, and output the corresponding symbol $x$.
2. Rescale the value as if the identified interval were the unit interval. i.e.
   \[ v_{\text{new}} = \frac{v - L(x)}{U(x) - L(x)}. \]  
   (8)
3. Unless $\langle \text{eot} \rangle$ has been detected, go to step 1.

Now we apply the decoding algorithm to our example.

1. The binary fraction 0.011000011 is 0.380859375 in decimal.
2. $v = 0.380859375$ so $x = b$ and
   \[ v_{\text{new}} = \frac{0.380859375 - 0.3}{0.6 - 0.3} = 0.26953125. \]  
   (9)
3. $v = 0.26953125$ so $x = a$ and
   \[ v_{\text{new}} = \frac{0.26953125 - 0.0}{0.3 - 0.0} = 0.8984375. \]  
   (10)
4. $v = 0.8984375$ so $x = d$ and
   \[ v_{\text{new}} = \frac{0.8984375 - 0.7}{0.9 - 0.7} = 0.9921875. \]  
   (11)
5. $v = 0.9921875$ so $x = \langle \text{eot} \rangle$ and we are done.