1 Advantages And Disadvantages Of Huffman Code

1.1 Advantages

- Huffman code is an optimal symbol code, it has shortest average length comparing to other codes.

1.2 Disadvantages

- Bad for skewed probability
- Employ block coding helps, i.e., use all $N$ symbols in a block. The redundancy is now $\frac{1}{N}$. However,
  - Must re-compute the entire table of block symbols if the probability of a single symbol changes. Recall the example about Huffman code tree in Lecture 7, if one symbol’s possibility changes, the whole tree will produce in different way.
  - For $N$ symbols, a table of $|X|^N$ must be pre-calculated to build the tree.
  - Symbol decoding must wait until an entire block is received.

2 Arithmetic Code

2.1 Introduction

Arithmetic code is developed by IBM, and it is the major compression schemes. In arithmetic coding, a different block of source symbols is represented by a disjoint subinterval in the unit interval. The code for a sequence of symbols is an interval whose length decreases as we add more symbols to the sequence.
• Some benefits of arithmetic coding:
  – No need for re-calculating a table of big size which can be inefficient.
  – Easy to adapt to change in probability.
  – Single symbol can be decoded immediately without waiting for the entire block of symbols to be received.

2.2 Encoding

• Each code is represented by a unique interval \([F(x_{i-1}), F(x_i)]\) in \([0,1)\), \(F(x_i) = p(X \leq x_i)\).

• \(F(x_i) - F(x_{i-1}) = p(X \leq x_i) - p(X \leq x_{i-1}) = p(X = x_i)\) represents the probability of \(x_i\) occurring.

• The interval \([F(x_{i-1}), F(x_i)]\) can itself be represented by any number, called a tag, within the half open interval. In this case, we will use the tag that is the number half way between \(F(x_{i-1})\) and \(F(x_i)\), and we also marks the tag as \(T(x_i)\). Then we can find that

\[
T(x_i) = \sum_{k=1}^{i-1} p(X = x_k) + \frac{p(X = x_i)}{2} = F_X(x_{i-1}) + \frac{F_X(x_i)}{2} + \frac{p(X = x_i)}{2} = \frac{F_X(x_{i-1}) + F_X(x_i)}{2}
\]

• Use \(l(x_i) = \left\lceil \log \frac{1}{p(x_i)} \right\rceil + 1\) significant bits of the tag \(0.t_1,t_2,t_3 \ldots, t_k,000\ldots\) is in the interval \([F(x_{i-1}), F(x_i)]\). Figure 1 shows how to define the tag

![Figure 1: Tag](image)

The process we get the code: Firstly, we find \(F(x_{i-1})\) and \(F(x_i)\). Secondly, using the formulae \(T_X([F(x_{i-1}), x_i]) = \frac{F_X(x_{i-1}) + F_X(x_i)}{2}\) and \(l(x_i) = \left\lceil \log \frac{1}{p(x_i)} \right\rceil + 1\) to get the tag and the length of code. Finally, turn the tag into binary and use \(l(x)\) to determines the final code.
2.3 Uniqueness Of The Arithmetic Code

**Theorem 2.1** Arithmetic code is a uniquely decodable code

**Proof** The idea of proving arithmetic code is a uniquely decodable code is to show that the arithmetic code is both nonsingular code and prefix code. To show that the code is non-singular code, we prove that the truncation of the tag lies entirely within \([F(X_{i-1}, F(X_i))\), which means each source symbol is mapped to a unique tag within the interval, there are no way for any tags represent to multiple source symbol, source symbols and tags are one-to-one correspondence. Hence we prove that arithmetic code is non-singular code. Next, we want to show that the truncation of the tag results in a prefix code. Since we prove that arithmetic code is both nonsingular code and prefix code, then arithmetic code is uniquely decodable.

1. In order to prove arithmetic code is non-singular, We show that the truncation of the tag lies entirely within \([F(x_{i-1}), F(x_i))\), that is to prove
\[
F(x_{i-1}) \leq [T(x_i)]_{l(x_i)} < F(x_i)
\]
\([T(x_i)]_{l(x_i)}\): Tag \(T(x)\) being truncated to \(l(x)\) bit.

First, notice that from
\[
\begin{aligned}
\left\{ \begin{array}{l}
[T(x_i)]_{l(x_i)} \leq T(x_i) \\
T(x_i) \leq F(x_i)
\end{array} \right.
\]
Therefore \([T(x_i)]_{l(x_i)} < F(x_i)\), the upper bound in (1) is proved.

Second, we want to show the lower bound, since
\[
\begin{aligned}
l(x_i) = \left\lceil \log \frac{1}{p(x_i)} \right\rceil + 1 \\
\left\lceil \log \frac{1}{p(x_i)} \right\rceil > \log \frac{1}{p(x_i)} \\
\log \frac{1}{p(x_i)} + 1 = \log \frac{1}{p(x_i)} + \log 2 = \log \frac{2}{p(x_i)}
\end{aligned}
\]
we have:
\[
\frac{1}{2l(x_i)} = \frac{1}{2 \left\lceil \log \frac{1}{p(x_i)} \right\rceil + 1} < \frac{1}{2 \log \frac{1}{p(x_i)} + 1} = \frac{p(x_i)}{2}
\]

Next, here is the fact:
\[
0 \leq T(x_i) - [T(x_i)]_{l(x_i)} \leq \frac{1}{2l(x_i)}
\]
For example, we get

\[
T(x) = 0.11011
\]

\[
[T(x)]_{l(x)=4} = 0.1101
\]

\[
T(x) - [T(x)]_{l(x)=4} = 0.00001
\]

\[
\frac{1}{2^{l(x)}} = \frac{1}{2^4} = 0.0001
\]

From above, we can find that the equation (5) is always true. Back to our proof, to combine equation (4) and (5), we get

\[
T(x_i) - [T(x_i)]_{l(x_i)} \leq \frac{1}{2^{l(x_i)}} \leq \frac{p(x_i)}{2}
\]

Thus,

\[
[T(x_i)]_{l(x_i)} \geq T(x_i) - \frac{p(x_i)}{2}
\]

From definition,

\[
T(x_i) = \frac{F_X(x_i) + F_X(x_i-1)}{2}
\]

\[
p(x_i) = \frac{F_X(x_i) - F_X(x_i-1)}{2}
\]

Then,

\[
T(x) - \frac{p(x)}{2} = \frac{F_X(x_i) + F_X(x_i-1)}{2} - \frac{F_X(x_i) - F_X(x_i-1)}{2}
\]

\[
= F_X(x_{i-1})
\]

Just substitute the equation (10) into (7), the lower bound is proved.

By proving

\[
F(x_{i-1}) \leq [T(x_i)]_{l(x_i)} < F(x_i)
\]

which means each source symbol is mapped to a unique tag within the interval, there are no way for any tags represent to multiple source symbol, source symbols and tags are one-to-one correspondence. So, we show that arithmetic code is non-singular code. Now, we have to show that arithmetic code is prefix code and then arithmetic code is uniquely decodable.

2. In order to prove that the truncation of the tag results in prefix code. We have to know a fact that for any \(a \in [0, 1)\) with a binary representation is \(0.b_1b_2\ldots b_n\) and for any \(c \in [0, 1)\) with a binary representation whose binary representation does has \(a\) as its prefix then \(c\) must lie in the interval \([a, a + \frac{1}{2^n})\).
For example, suppose we have:

\[
\begin{align*}
  a &= 0.1101 \quad (n = 4) \\
  c &= 0.11010
\end{align*}
\]

In this case \( c \) has \( a \) as a prefix, and clearly \( c \) lies inside the interval \( [a, a + \frac{1}{2^n}] \) as:

\[
a + \frac{1}{2^n} = + 0.1110
\]

For two different strings \( x_1, x_2 \), we already showed that \( \lfloor T(x_1) \rfloor_{l(x_1)} \) and \( \lfloor T(x_2) \rfloor_{l(x_2)} \) lie in different intervals, So now all we have to do is to show that any strings

\[
\lfloor T(x_i) \rfloor_{l(x_i)} , \lfloor T(x_i) \rfloor_{l(x_i)} + \frac{1}{2^{l(x_i)}}
\]

lie strictly in the interval

\[
[F(x_{i-1}), F(x_i)]
\]

The reason is that for any tag \( \lfloor T(x_i) \rfloor_{l(x_i)} = n \) as a prefix to another tag \( \lfloor T(x_i) \rfloor_{l(x_i)} = m \) \( (m > n > 0) \). The tag \( \lfloor T(x_i) \rfloor_{l(x_i)} = m \) must lie in the interval

\[
\left[ \lfloor T(x_i) \rfloor_{l(x_i)} = n, \lfloor T(x_i) \rfloor_{l(x_i)} + \frac{1}{2^{l(x_i)}} \right]
\]

(11)

If the interval (11) lie strictly in the interval \( [F(x_{i-1}), F(x_i)] \), there are no way that tag \( \lfloor T(x_i) \rfloor_{l(x_i)} = m \) exists due to arithmetic code is a non-singular code. There are only one \( \lfloor T(x_i) \rfloor_{l(x_i)} \) lie in interval \( [F(x_{i-1}), F(x_i)] \). It mean that code of source symbol can not be the prefix of another source symbol. Now, we have already shown that

\[
\lfloor T(x_i) \rfloor_{l(x_i)} \geq F(x_{i-1})
\]

All we have to do now is to show that

\[
\lfloor T(x_i) \rfloor_{l(x_i)} + \frac{1}{2^{l(x_i)}} < F(x_i)
\]

(12)

But this is equivalent to show

\[
F(x_i) - \lfloor T(x_i) \rfloor_{l(x_i)} > \frac{1}{2^{l(x_i)}}
\]

(13)

Since \( \lfloor T(x_i) \rfloor_{l(x_i)} \) is less than \( T(x_i) \), we get

\[
F(x_i) - \lfloor T(x_i) \rfloor_{l(x_i)} \geq F(x_i) - T(x_i)
\]

(14)
and
\[ F(x_i) - T(x_i) = \frac{p(x_i)}{2} = 2 \cdot \frac{\log \frac{1}{p(x_i)}}{2} > 2^{\log \frac{1}{p(x_i)}} - 1 = \frac{1}{2^{l(x_i)}} \quad (15) \]

To combine (14) and (15) together, we prove that the truncation of the tag results in a prefix code. Since we prove that arithmetic code is both nonsingular code and prefix code, then arithmetic code is uniquely decodable.

### 2.4 Efficiency Of The Arithmetic Code

We use \( L_A \) to evaluate the efficiency of arithmetic code, \( L_A = E[l(C(X))] \), \( l(C(X)) \) is the length of the code. The code is more efficient when \( L_A \) is more smaller.

**Theorem 2.2**  
Arithmetic code is off at most by \( 2/N \)
\[
\frac{H(X_1, X_2, \ldots, X_N)}{N} \leq L_A < \frac{H(X_1, X_2, \ldots, X_N)}{N} + \frac{2}{N} \quad (16)
\]

For i.i.d.:
\[
H(X) \leq L_A < H(X) + \frac{2}{N} \quad (17)
\]

**Proof** For upper bound:

\( L_A^N \): The average length of arithmetic code encoded blocks of \( N \) symbols at a time

\[
L_A^N = \sum_{x_i \in |x|^N} p(x_i) l(x_i)
= \sum_{x_i \in |x|^N} p(x_i) \left( \left\lceil \log \frac{1}{p(x_i)} \right\rceil + 1 \right)
\leq \sum_{x_i \in |x|^N} p(x_i) \left( \left\lceil \log \frac{1}{p(x_i)} \right\rceil + 2 \right)
= H(X_1, X_2, \ldots, X_N) + 2
\]

So
\[
L_A^N \leq H(X_1, X_2, \ldots, X_N) + 2
\]

Or
\[
L_A \leq H(x) + \frac{2}{N} \quad (18)
\]
(18) is only true when $X_1, X_2, \ldots, X_N$ are i.i.d.

For upper bound:

$$L_N^A = \sum_{x \in |x|^N} p(x_i) l(x_i)$$

$$= \sum_{x \in |x|^N} p(x_i) \left( \left\lceil \log \frac{1}{p(x_i)} \right\rceil + 1 \right)$$

$$\geq \sum_{x \in |x|^N} p(x_i) \left( \left\lceil \log \frac{1}{p(x_i)} \right\rceil \right)$$

$$= H(X_1, X_2 \cdots, X_N)$$

So

$$L_N^A \geq H(X_1, X_2 \cdots, X_N)$$

Or

$$L_A \geq H(X) \quad (19)$$

(19) is only true when $X_1, X_2, \ldots, X_N$ are i.i.d.

### 2.5 Examples of arithmetic coding

- Figure 2 shows an example of arithmetic coding

According to the encoding process of arithmetic coding, to find out the code for bba, we set the upper point of bba is $F(x_1)$, the lower point of bba is $F(x_2)$. So the tag of bba = $\frac{F(x_1) + F(x_2)}{2}$. From the Figure 2, we can find out that $F(x_2) = 1 - \frac{2}{3} \times \frac{2}{3} = \frac{19}{27}$, $F(x_1) = 1 - \frac{2}{3} \times \frac{2}{3} = \frac{15}{27}$, then we can get the tag of bba = $\frac{\frac{15}{27} + \frac{19}{27}}{2} = \frac{17}{27}$. We set the probability of bba is $p(x)$, $p(x) = F(x_2) - F(x_1) = \frac{19}{27} - \frac{15}{27} = \frac{4}{27}$. Then we convert $\frac{4}{27}$ into binary = .0101000010... The length of the code, $l(x) = \left\lceil \log \frac{27}{4} \right\rceil + 1 = 4$. So we get the code for bba is 1010.
• Figures 3 gives more codes for last example.

- P(a) = 1/3, P(b) = 2/3.

<table>
<thead>
<tr>
<th>Symble</th>
<th>F(x)</th>
<th>T_X(x)</th>
<th>In Binary</th>
<th>\log_2 \left( \frac{1}{p(x)} \right) + 1</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>0.25</td>
<td>0.10</td>
<td>2</td>
<td>01</td>
</tr>
<tr>
<td>2</td>
<td>0.75</td>
<td>0.625</td>
<td>0.101</td>
<td>3</td>
<td>1101</td>
</tr>
<tr>
<td>3</td>
<td>0.875</td>
<td>0.8125</td>
<td>0.1101</td>
<td>4</td>
<td>1111</td>
</tr>
<tr>
<td>4</td>
<td>1.0</td>
<td>0.9375</td>
<td>0.1111</td>
<td>4</td>
<td>1111</td>
</tr>
</tbody>
</table>

Figure 3: More codes for last example

- Another example for arithmetic coding

\[
\begin{array}{c|cccc}
   & 1   & 2 & 3 & 4 \\
p(x) & 0.5 & 0.25 & 0.125 & 0.125 \\
\end{array}
\]

For the tag \( T(x_i) \), we can get it from \( T(x_i) = \frac{F(x_{i-1}) + F(x_i)}{2} \). And \( l(x_i) = \left\lceil \log_2 \left( \frac{1}{p(x_i)} \right) \right\rceil + 1 \) determines the length of the code. \( F(x) \) is the cumulative function of \( p(x) \). For example, if we want to get the code of symbol 2, firstly, we find \( F(2) = p(1) + p(2) = 0.75 \), \( T(2) = \frac{F(1) + F(2)}{2} = \frac{0.5 + 0.75}{2} = 0.625 \). \( l(2) = \left\lceil \log_2 \left( \frac{1}{p(2)} \right) \right\rceil + 1 = 3. \) Secondly, Turn 0.625 into binary is 0.101, and the length of code is 3, so the code is 101.
- Arithmetic code for two-symbol sequences

<table>
<thead>
<tr>
<th>Message</th>
<th>( P(x) )</th>
<th>( \bar{T}_x(x) )</th>
<th>( \bar{T}_x(x) ) in Binary</th>
<th>( \log \frac{1}{P(x)} ) +1</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>.25</td>
<td>.125</td>
<td>.001</td>
<td>3</td>
<td>001</td>
</tr>
<tr>
<td>12</td>
<td>.125</td>
<td>.3125</td>
<td>.0101</td>
<td>4</td>
<td>0101</td>
</tr>
<tr>
<td>13</td>
<td>.0625</td>
<td>.40625</td>
<td>.01101</td>
<td>5</td>
<td>01101</td>
</tr>
<tr>
<td>14</td>
<td>.0625</td>
<td>.46875</td>
<td>.011111</td>
<td>5</td>
<td>011111</td>
</tr>
<tr>
<td>21</td>
<td>.125</td>
<td>.5625</td>
<td>.1001</td>
<td>4</td>
<td>1001</td>
</tr>
<tr>
<td>22</td>
<td>.0625</td>
<td>.65625</td>
<td>.10101</td>
<td>5</td>
<td>10101</td>
</tr>
<tr>
<td>23</td>
<td>.03125</td>
<td>.703125</td>
<td>.101101</td>
<td>6</td>
<td>101101</td>
</tr>
<tr>
<td>24</td>
<td>.03125</td>
<td>.734375</td>
<td>.101111</td>
<td>6</td>
<td>101111</td>
</tr>
<tr>
<td>31</td>
<td>.0625</td>
<td>.78125</td>
<td>.11001</td>
<td>5</td>
<td>11001</td>
</tr>
<tr>
<td>32</td>
<td>.03125</td>
<td>.828125</td>
<td>.110101</td>
<td>6</td>
<td>110101</td>
</tr>
<tr>
<td>33</td>
<td>.015625</td>
<td>.8515625</td>
<td>.1101101</td>
<td>7</td>
<td>1101101</td>
</tr>
<tr>
<td>34</td>
<td>.015625</td>
<td>.8671875</td>
<td>.11011111</td>
<td>7</td>
<td>11011111</td>
</tr>
<tr>
<td>41</td>
<td>.0625</td>
<td>.90625</td>
<td>.11101</td>
<td>5</td>
<td>11101</td>
</tr>
<tr>
<td>42</td>
<td>.03125</td>
<td>.953125</td>
<td>.111101</td>
<td>6</td>
<td>111101</td>
</tr>
<tr>
<td>43</td>
<td>.015625</td>
<td>.9765625</td>
<td>.1111101</td>
<td>7</td>
<td>1111101</td>
</tr>
<tr>
<td>44</td>
<td>.015625</td>
<td>.984375</td>
<td>.1111111</td>
<td>7</td>
<td>1111111</td>
</tr>
</tbody>
</table>

Figure 4: Arithmetic code for two-symbol sequences

For this example, From \( p(1) \) to \( p(4) \) are same value as last one. For symbol 11, the probability of this symbol is \( p(11) = 0.5 \times 0.5 = 0.25, F(11) = 0.25 \). For symbol 12, \( p(12) = 0.5 \times 0.25 = 0.125 \). \( F(12) = F(11) + P(12) = 0.375 \). \( T(12) = \frac{F(11)+F(12)}{2} = 0.3125 \). \( l(12) = \left\lceil \log \frac{1}{p(12)} \right\rceil +1 = 4 \). Turn 0.3125 into binary is 0.0101, so the code is 0101.
2.6 Decoding

Figure 5 shows the process of decoding.

Each additional bit received narrows down the possible interval.

$X = [a, b], \ p = [0.6, 0.4]$
2.7 Adaptive Arithmetic Code

![Adaptive Arithmetic Code Diagram]

Figure 6: Adaptive Arithmetic Code

Figure 6 shows the process of adaptive arithmetic coding. For this example, the probability of \( b \) is 0.5, we can get the probability of \( bb \) and \( ba \) by the formula,

\[
p(n) = \frac{1 + \text{count}(x_i = b)}{1 + n}, \quad bb = \frac{1 + 1 + 1}{1 + 2} = \frac{2}{3}, \quad ba = \frac{1}{1 + 2} = \frac{1}{3}.
\]

3 Dictionary Code

<table>
<thead>
<tr>
<th>index</th>
<th>pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>ab</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>abc</td>
</tr>
</tbody>
</table>

Different index maps to different pattern. Encoder codes the index. Before we transfer the index, we need to decode the index by decoder.