Arithmetic Code

Contents

- Introduction to Arithmetic Coding
- Arithmetic Coding & Decoding Algorithm
- Generating a Binary Code for Arithmetic Coding
Introduction

Principle:

A message (source string) is represented by an interval of real numbers between 0 and 1. More frequent messages have larger intervals allowing fewer bits to specify those intervals.

The Principle of Arithmetic Coding

Example 5.12

- Same source alphabet as that used in Example 5.9.
- In this example, however, a string of source symbols \( s_1 s_2 s_3 s_4 s_5 s_6 \) is encoded.

<table>
<thead>
<tr>
<th>Source symbol</th>
<th>Occurrence probability</th>
<th>Associated subintervals</th>
<th>CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_6 )</td>
<td>0.3</td>
<td>(0.75, 1.0)</td>
<td>0.75</td>
</tr>
<tr>
<td>( s_5 )</td>
<td>0.1</td>
<td>(0.45, 0.75)</td>
<td>0.65</td>
</tr>
<tr>
<td>( s_4 )</td>
<td>0.1</td>
<td>(0.15, 0.45)</td>
<td>0.15</td>
</tr>
<tr>
<td>( s_3 )</td>
<td>0.2</td>
<td>(0.06, 0.15)</td>
<td>0.06</td>
</tr>
<tr>
<td>( s_2 )</td>
<td>0.1</td>
<td>(0.04, 0.06)</td>
<td>0.04</td>
</tr>
<tr>
<td>( s_1 )</td>
<td>0.3</td>
<td>(0.0, 0.04)</td>
<td>0</td>
</tr>
</tbody>
</table>
Dividing Interval [0,1) into Subintervals

**Cumulative probability (CP)**
Slightly different from that of cumulative distribution function (CDF) in probability theory.

\[
CDF(s_i) = \sum_{j=1}^{i} p(s_j) \quad , \quad (5.12)
\]

\[
CP(s_i) = \sum_{j=1}^{i-1} p(s_j) \quad , \quad (5.13)
\]

where \( CP(s_1) = 0 \).

Dividing Interval [0,1) into Subintervals

- Each subinterval
  - Its **lower end point** located at \( CP(s_i) \).
  - **Width** of each subinterval equal to probability of corresponding source symbol.
  - A subinterval can be completely defined by its **lower end point** and its **width**.
  - Alternatively, it is determined by its **two end points**: the lower and upper end points (sometimes also called the left and right end points).
Arithmetic code: Algorithm to calculate the subinterval

0) Start by defining the current interval as [0,1).
1) REPEAT for each symbol \( s \) in the input stream
   a) Divide the current interval \([L, H)\) into subintervals whose sizes are proportional to the symbols's probabilities.
   b) Select the subinterval \([L, H)\) for the symbol \( s \) and define it as the new current interval
2) When the entire input stream has been processed, the output should be any number \( V \) that uniquely identify the current interval \([L, H)\).

Arithmetic Coding

- Encoding algorithm for arithmetic coding.

\[
\begin{align*}
\text{Low} & = 0.0 ; \ \text{high} = 1.0 ; \\
\text{while} \ \text{not EOF} \ \text{do} \\
\text{range} & = \text{high} - \text{low} ; \ \text{read}(c) ; \\
\text{high} & = \text{low} + \text{range} \times \text{high}\_\text{range}(c) ; \\
\text{low} & = \text{low} + \text{range} \times \text{low}\_\text{range}(c) ; \\
\text{enddo} \\
\text{output}(\text{low}) ;
\end{align*}
\]
Dividing Interval [0,1) into Subintervals

Figure 5.2 Arithmetic coding working on the same source alphabet as that in Example 5.2. The encoded symbol string is $s_1 s_2 s_3 s_4 s_5 s_6$.

Encoding

- **Encoding the First Source Symbol**
  - Refer to Part (a) of Figure 5.2. Since the first symbol is $s_1$, we pick up its subinterval [0, 0.3). Picking up the subinterval [0, 0.3) means that any real number in the subinterval, i.e., any real number equal to or greater than 0 and smaller than 0.3, can be a pointer to the subinterval, thus representing the source symbol $s_1$. This can be justified by considering that all the six subintervals are disjoined.
Encoding

- **Encoding the Second Source Symbol**
  - Refer to Part (b) of Figure 5.2. We use the same procedure as used in Part (a) to divide the interval [0, 0.3) into six subintervals. Since the second symbol to be encoded is $s_2$, we pick up its subinterval [0.09, 0.12).
  - Notice that the subintervals are recursively generated from Part (a) to Part (b). It is known that an interval may be completely specified by its lower end point and width. Hence, the subinterval recursion in the arithmetic coding procedure is equivalent to the following two recursions: **end point recursion** and **width recursion**.

- **Encoding the Third Source Symbol**
  - The resulting subinterval [0.1058175, 0.1058250) can represent the source symbol string $s_1s_2s_3s_4s_5s_6$.

- **Encoding the Fourth, Fifth and Sixth Source Symbols**
  - The resulting subinterval [0.1058175, 0.1058250) can represent the source symbol string $s_1s_2s_3s_4s_5s_6$. 

\[ L_{\text{new}} = L_{\text{current}} + W_{\text{current}} \cdot CP_{\text{new}} \]

\[ W_{\text{new}} = W_{\text{current}} \cdot p(s_i) \]  \hspace{1cm} (5.14)
Decoding algorithm

\[
\begin{align*}
  r &= \text{input
code} \\
  \text{repeat} \\
  &\quad \text{search } c \text{ such that } r \text{ falls in its range} \\
  &\quad \quad \text{output}(c) \\
  &\quad r = r - \text{low_range}(c) \\
  &\quad r = r/(\text{high_range}(c) - \text{low_range}(c)) \\
  \text{until } r \text{ equal 0}
\end{align*}
\]

Decoding

- Theoretically, any real numbers in the interval can be the code string for the input symbol string since all subintervals are disjoined.
- Often, however, the lower end of the final subinterval is used as the code string.
- Now let us examine how the decoding process is carried out with the lower end of the final subinterval.
  - The decoder knows the encoding procedure and therefore has the information contained in Part (a) of Figure 5.2.
    - Since \(0 < 0.1058175 < 0.3\), the symbol \(s_1\) is first decoded.
Decoding

- Once the first symbol is decoded, the decoder may know the partition of subintervals shown in Part (b) of Figure 5.2. It is then determined that $0.09 < 0.1058175 < 0.12$. That is, the lower end is contained in the subinterval corresponding to the symbol $s_2$. As a result, $s_2$ is the second decoded symbol.

The procedure repeats itself until all six symbols are decoded.

- Note that a terminal symbol is necessary to inform the decoder to stop decoding.

- The above procedure gives us an idea of how decoding works.

Decoding

- The decoding process, however, does not need to construct Parts (b), (c), (d), (e) and (f) of Figure 5.2. Instead, the decoder only needs the information contained in Part (a) of Figure 5.2.

- Decoding can be split into the following three steps: **comparison, readjustment** (subtraction), and **scaling** [Langdon 1984].

- In summary, considering the way in which Parts (b), (c), (d), (e) and (f) of Figure 5.2 are constructed, we see that the three steps discussed in the decoding process: comparison, readjustment and scaling exactly "undo" what the encoding procedure has done.
Observations

- Both encoding and decoding involve only arithmetic operations (addition and multiplication in encoding, subtraction and division in decoding). This explains the name arithmetic coding.
- We see that an input source symbol string $s_1s_2s_3s_4s_5s_6$, via encoding, corresponds to a subinterval $[0.1058175, 0.1058250)$. Any number in this interval can be used to denote the string of the source symbols.
- We also observe that arithmetic coding can be carried out in an incremental manner. That is, source symbols are fed into the encoder one by one and the final subinterval is refined continually, i.e., the code string is generated continually.

Example 1

Example:

<table>
<thead>
<tr>
<th>Source symbol</th>
<th>Probability (binary)</th>
<th>Cumulative Probability (binary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>0.500 0.100</td>
<td>0.000 0.000</td>
</tr>
<tr>
<td>$b$</td>
<td>0.250 0.010</td>
<td>0.500 0.100</td>
</tr>
<tr>
<td>$c$</td>
<td>0.125 0.001</td>
<td>0.750 0.110</td>
</tr>
<tr>
<td>$d$</td>
<td>0.125 0.001</td>
<td>0.875 0.111</td>
</tr>
</tbody>
</table>
Example 1 (cont.)

The length of an interval is proportional to its probability

<table>
<thead>
<tr>
<th></th>
<th>0.000</th>
<th>0.100</th>
<th>0.750</th>
<th>0.875</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.000</td>
<td>0.100</td>
<td>0.750</td>
<td>0.875</td>
<td>1.00</td>
</tr>
<tr>
<td>b</td>
<td>0.010</td>
<td>0.110</td>
<td>0.111</td>
<td>1.011</td>
<td>1.011</td>
</tr>
<tr>
<td>c</td>
<td>0.001</td>
<td>0.011</td>
<td>0.111</td>
<td>1.011</td>
<td>1.011</td>
</tr>
<tr>
<td>d</td>
<td>0.000</td>
<td>0.100</td>
<td>0.750</td>
<td>0.875</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Any point in the interval [0.0, 0.5) represents “a”; say, 0.25(binary: 0.01), or 0.0(binary: 0.00)

Any point in the interval [0.75, 0.875) represents “c”; say, 0.8125(binary: 0.1101), or 0.75(binary: 0.110)

Transmitting 3 letters:

1st letter “a”

2nd letter “a”

3rd letter “b”

Any point in the interval [0.001, 0.0011) identifies “aab”; say, 0.00101, or 0.0010.

Need a model (probability distribution)
Example (2)

<table>
<thead>
<tr>
<th>Character</th>
<th>probability</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>(space)</td>
<td>1/10</td>
<td>0.00 ≤ r &lt; 0.10</td>
</tr>
<tr>
<td>A</td>
<td>1/10</td>
<td>0.10 ≤ r &lt; 0.20</td>
</tr>
<tr>
<td>B</td>
<td>1/10</td>
<td>0.20 ≤ r &lt; 0.30</td>
</tr>
<tr>
<td>E</td>
<td>1/10</td>
<td>0.30 ≤ r &lt; 0.40</td>
</tr>
<tr>
<td>G</td>
<td>1/10</td>
<td>0.40 ≤ r &lt; 0.50</td>
</tr>
<tr>
<td>I</td>
<td>1/10</td>
<td>0.50 ≤ r &lt; 0.60</td>
</tr>
<tr>
<td>L</td>
<td>2/10</td>
<td>0.60 ≤ r &lt; 0.80</td>
</tr>
<tr>
<td>S</td>
<td>1/10</td>
<td>0.80 ≤ r &lt; 0.90</td>
</tr>
<tr>
<td>T</td>
<td>1/10</td>
<td>0.90 ≤ r &lt; 1.00</td>
</tr>
</tbody>
</table>

Suppose that we want to encode the message

BILL GATES
Example (2)

<table>
<thead>
<tr>
<th>New character</th>
<th>Low value</th>
<th>high value</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>0.2</td>
<td>0.3</td>
</tr>
<tr>
<td>I</td>
<td>0.25</td>
<td>0.26</td>
</tr>
<tr>
<td>L</td>
<td>0.256</td>
<td>0.258</td>
</tr>
<tr>
<td>L</td>
<td>0.2572</td>
<td>0.2576</td>
</tr>
<tr>
<td>^ (space)</td>
<td>0.25720</td>
<td>0.25724</td>
</tr>
<tr>
<td>G</td>
<td>0.257216</td>
<td>0.257220</td>
</tr>
<tr>
<td>A</td>
<td>0.2572164</td>
<td>0.2572168</td>
</tr>
<tr>
<td>T</td>
<td>0.25721676</td>
<td>0.2572168</td>
</tr>
<tr>
<td>E</td>
<td>0.257216772</td>
<td>0.257216776</td>
</tr>
<tr>
<td>S</td>
<td>0.2572167752</td>
<td>0.2572167756</td>
</tr>
</tbody>
</table>

The final value, named a **tag**, 0.2572167752 will uniquely encode the message ‘BILL GATES’.

Any value between 0.2572167752 and 0.2572167756 can be a **tag** for the encoded message, and can be uniquely decoded.
Decoding - Example (2)

- Decoding is the inverse process.
- Since 0.2572167752 falls between 0.2 and 0.3, the first character must be ‘B’.
- Removing the effect of ‘B’ from 0.2572167752 by first subtracting the low value of B, 0.2, giving 0.0572167752.
- Then divided by the width of the range of ‘B’, 0.1. This gives a value of 0.572167752.

Then calculate where that lands, which is in the range of the next letter, ‘I’.
- The process repeats until 0 or the known length of the message is reached.
Example 3

Using the probability below, encode the message

Il u u r e?
Conclusions

- In summary, the encoding process is simply one of narrowing the range of possible numbers with every new symbol.
- The new range is proportional to the predefined probability attached to that symbol.
- Decoding is the inverse procedure, in which the range is expanded in proportion to the probability of each symbol as it is extracted.
Conclusions

- JPEG, MPEG-1/2 uses Huffman and arithmetic coding
  - preprocessed by DPCM
  - JPEG-LS
- JPEG2000, MPEG-4 uses arithmetic coding only