The Revenge of a Student - Symbol Codes

Symbol codes

- Notation: \(\{0,1\}^* = \{0,1,00,01,10,11,000,\ldots\}\)
- A symbol code \(C\) is a mapping from \(\mathcal{A}\) to \(\{0,1\}^*\)

\[\begin{align*}
c(x_1x_2x_3\ldots x_n) &= c(x_1)c(x_2)c(x_3)\ldots c(x_n) \\
l(x) &= |x|
\end{align*}\]

Decoding of symbol codes

- A code \(C(X)\) is uniquely decodable if

\(\forall x, y \in \mathcal{A}^*, x \neq y \implies c^*(x) \neq c^*(y)\)
- A code \(C(X)\) is a prefix code if no codeword is a prefix of any other codeword
- The expected length \(L(C,X)\) of a symbol code \(C\) for ensemble \(X\) is

\[L(C,X) = \sum P(x)l(x)\]

Example

\(\mathcal{A} = \{1,2,3,4\}, P_X = \{1/2,1/4,1/8,1/8\}\)

\(C: c(1) = 0, c(2) = 10, c(3) = 110, c(4) = 111\)

The entropy of \(X\) is 1.75 bits: \(L(C,X)\) is also 1.75 bits

Kraft inequality

- Given a list of integer \((l_i)\), does there exist a uniquely decodable code with \((l_i)\)?
- "Market model": total budget 1; cost per codeword of length \(l\) is \(2^{-l}\).

Kraft inequality: For any uniquely decodeable code \(C\) over the binary alphabet \(\{0,1\}\), the codeword lengths must satisfy:

\[\sum 2^{-l_i} \leq 1\]

Conversely, given a set of codeword lengths that satisfy this inequality, there exists a uniquely decodable prefix code with these code lengths.

Limits of unique decodeability

<table>
<thead>
<tr>
<th>Total budget</th>
<th>00</th>
<th>000</th>
<th>0000</th>
<th>0001</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>1000</th>
<th>1001</th>
<th>101</th>
<th>110</th>
<th>1100</th>
<th>1101</th>
<th>111</th>
<th>1110</th>
<th>1111</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What can we hope for?

**Lower bound on expected length:** The expected length \( L(C,X) \) of a uniquely decodable code is bounded below by \( H(X) \).

**Compression limit of symbol codes:** For an ensemble \( X \) there exists a prefix code
\[
H(X) \leq L(C,X) < H(X) + 1.
\]

(What happens if we use the “wrong” code?)

Assume the “true probability distribution” is \( \{p_i\} \). If we use a complete code with lengths \( l_i \), they define a probabilistic model \( q_i = 2^{-l_i} \). The average length is
\[
L(C,X) = H(X) + \sum_i p_i \log p_i / q_i.
\]

"Proof-map" of the lower bound

Define \( q_i = 2^{-l_i} \), where \( z = \sum_i 2^{-l_i} \).
Thus \( l_i = \log 1/q_i - \log z \)

\[
L(C,X) = \sum_i p_i l_i = \sum_i p_i \log 1/q_i - \log z
\]

\[
\geq \sum_i p_i \log 1/p_i - \log z = H(X)
\]

"Optimal" symbol code: Huffman coding

- Take two least probable symbols in the alphabet as defined by \( \{p_i\} \).
- Combine these symbols into a single symbol, \( p_{\text{new}} = p_1 + p_2 \). Repeat (until one symbol)

**Huffman in practice**

\[
\begin{align*}
1.0 & \quad 0.55 \quad 0.45 \quad 0.45 \quad 0.3 \quad 0.3 \quad 0.25 \\
0.25 & \quad 0.25 \quad 0.25 \quad 0.25 \quad 0.25 \quad 0.25 \quad 0.25
\end{align*}
\]

**Huffman for the Linux manual**

\[
\begin{align*}
L(C,X) & = 4.15 \text{ bits} \\
H(X) & = 4.11 \text{ bits}
\end{align*}
\]
Why is this not the end of the story?

- Adaptation: what if the ensemble X changes? (as it does...)
  - calculate probabilities in one pass
  - communicate code + the Huffman-coded message
- "The extra bit": what if H(X) ~1 bit?
  - Group symbols to blocks and design a "Huffman block code"

The guessing game

THERE-IS-NO-GROUP-LIKE-COSCO-GROUP

The number of guesses before the character was identified

Encode: use the number of guesses

"A new alphabet"

Decode: let the twin guess and stop after the communicated number of guesses

History of arithmetic coding

- Does not require that the symbols translate into integral number of bits
- Shannon 1948 discussed binary fractions
- First code of this type discovered by Elias
- 1976 Pasco and Rissanen (independently)
- Rissanen & Langdon 1979 described hardware implementation

An example fixed model

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Probability</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.2</td>
<td>[0,0.2)</td>
</tr>
<tr>
<td>e</td>
<td>0.3</td>
<td>[0.2,0.5)</td>
</tr>
<tr>
<td>i</td>
<td>0.1</td>
<td>[0.5,0.6)</td>
</tr>
<tr>
<td>o</td>
<td>0.2</td>
<td>[0.6,0.8)</td>
</tr>
<tr>
<td>u</td>
<td>0.1</td>
<td>[0.8,0.9)</td>
</tr>
<tr>
<td>!</td>
<td>0.1</td>
<td>[0.9,1.0)</td>
</tr>
</tbody>
</table>

The idea
Arithmetic coding

- with every new symbol produced by the source, the probabilistic model provides a predictive distribution over all possible values of the next symbol
- i.i.d. = predictive distribution does not change
- encoder uses the model predictions to create a binary string

Basics

- Source alphabet \( \mathcal{A} = \{a_1, \ldots, a_I\} \)
- Source stream \( x_1, x_2, \ldots \)
- Model \( M: \ P(x_n = a_j | x_1, \ldots, x_{n-1}) \)
- A binary transmission is viewed defining an interval within the real line from 0 to 1

Encoding example

10011101

Decoder

Calculate the initial \( P(a), P(b), P(!) \) (duplicate the encoder!) and deduce the intervals “a”, “b” and “!”

10 — Deduce that the first symbol was “b”

Calculate \( P(ab), P(bb) \) and \( P(bl) \) and deduce the intervals “ba”, “bb” and “bl”

1001 — Deduce that the second symbol was “b” Etc.

Decoding example

10011101

Decoder

Various codes: the big picture

- fixed length block codes: mappings from a fixed number of course symbols to a fixed length binary message
- symbol codes
  - variable length code for each symbol in the alphabet
  - code lengths integers
  - Huffman code (expectation) optimal
...big picture continued

- **stream codes**
  - not constrained to emit at least one bit for every symbol in the source stream
  - **arithmetic codes** use a probabilistic model that identifies each string with a sub-interval of \([0,1)\).
  - “Good compression requires intelligence”
  - **Lempel-Ziv codes** memorize strings that have already occurred. “No prior assumptions on the world”

---

**Lempel-Ziv coding**

- simple to implement, asymptotic rate approaches the entropy
- widely used (gzip, compress...)
- basic idea: replace a substring with a pointer to an earlier occurrence of the substring
- Example:
  - String: 101101010010...
  - Substrings: 1, 0, 11, 01, 00, 10...
  - Replace 010 with a pointer to “01 + 1”