3.2 Huffman Coding

Two Requirements for optimum prefix codes

1. Likely symbols \(\rightarrow\) Short code words
   Unlikely symbols \(\rightarrow\) Long code words
   (Recall entropy discussion)

2. The two least likely symbols have codewords of some length.

Why #2?

Two least likely symbols \(a_i\) \(\rightarrow\) \(d\) codewords
\(a_i\) \(\rightarrow\) \(d\)

Two least likely symbols have codewords that differ only in the last bit.

Additional Huffman Requirement

The two least likely symbols have codewords that differ only in the last bit.

These three requirements lead to a simple way of building a binary tree describing an optimum prefix code – The Huffman Code.

- Build it from the bottom up, starting with two least likely symbols
- The external nodes correspond to the symbols
- The internal nodes correspond to "super symbols" in a "reduced" alphabet
**Huffman Steps**

1. Label each node w/ one of the source symbol probabilities.

2. Merge the nodes labeled by the two smallest probabilities into a parent node.

3. Label the parent node w/ the sum of the two children's probabilities. This parent node is now considered to be a "super symbol" (it replaces its two children symbols) in a reduced alphabet.

4. Among the elements in the reduced alphabet merge two that have the smallest "probabilities." If there is more than one such pair choose the pair that has the "lowest order super symbols" (this assures the minimum variance Huffman code — read sect. 3.2.1)

5. Label the parent node w/ the sum of the two childrens probabilities.

6. Repeat steps 4 & 5 until only a single super symbol remains (w/ Prob. of 1)
Performance of Huffman Codes

3.3.1 - 3.3.3

Sect. 3.3.2: Skip the details, state the main results

How close to entropy $H(S)$ can Huffman get?

Result #1

If all symbol probabilities are powers of two

then

$$\bar{e} = H(S)$$

Result #2

$$H_1(S) \leq \bar{e} < H_1(S) + 1$$

Difference is called redundancy

Result #3. Refined upper bound

$$\bar{e} < \begin{cases} H_1(S) + P_{\text{max}}, & P_{\text{max}} < 0.5 \\ H_1(S) + P_{\text{max}} + 0.086, & P_{\text{max}} \geq 0.5 \end{cases}$$

Definition: Redundancy = difference between $\bar{e}$ and entropy

Note: Large alphabets tend to have small $P_{\text{max}}$

But small $\bar{e}$ can have large $P_{\text{max}}$ $\Rightarrow$ Huffman not so good
Applications of Huffman

We'll look at use in Groups for, later. (Ch. 6)

Lesserless Image Compression Examples

Directly: $1.14 \leq CR \leq 1.67$

Differences: $1.66 \leq CR \leq 2.03$

Not that great

Text Compression Example

Applied to Ch. 3: $CR = 1.63$

Audio Compression Examples

Directly: $1.16 \leq CR \leq 1.3$

Differences: $1.47 \leq CR \leq 1.65$

So why have we looked at something so bad?

- Provides good intro to compression ideas
- Historical context
- Huffman is used as building block in more advanced methods
  - Group 3 FAX (lossless)
  - JPEG Image (lossy)
  - etc.
Block Huffman Codes

"Extended"

(Useful when Huffman not effective \( \Rightarrow \) large \( n \))

Ex. 3.3.3

IID

\[ P(a_1) = 0.8 \quad P(a_2) = 0.02 \quad P(a_3) = 0.18 \]

Book shows need 47% more bits than entropy

Block codes allow better performance \( \Rightarrow \) noninteger bits/symbol

Note: assuming IID \( \Rightarrow \) no context considered

(if source really is not IID we can do better through use of context models)

Group into \( n \)-symbol blocks \( \Rightarrow \) mapping between original alphabet and a new "extended" alphabet

\[ \{a_1, a_2, \ldots, a_m\} \rightarrow \{a, a_1, a_2, a_3, \ldots, a_{n+1}, a_{n+2}, \ldots, a_m\} \]

\( n \) elements

\[ a, a_1, a_2, a_3, \ldots, a_{n+1}, a_{n+2}, \ldots, a_m \]

\( n \) elements

\( \Rightarrow \) need \( n \) code words

use Huffman procedure on probabilities of blocks

\[ P(a_1, a_2, \ldots, a_m) = P(a_1) P(a_2) \cdots P(a_m) \]
Let $S^{(n)}$ denote the block source 

$R^{(n)}$ denote the rate of block Huffman code (bits/blk) 

$H(S^{(n)})$ be the entropy of the block source

Then, using bounds discussed earlier:

$$H(S^{(n)}) \leq R^{(n)} \leq H(S^{(n)}) + 1$$

$\uparrow$

# of bits per n symbols $\Rightarrow R = \frac{R^{(n)}}{n}$

$\uparrow$

# of bits/symbol

$$\Rightarrow \frac{H(S^{(n)})}{n} \leq R < \frac{H(S^{(n)}) + 1}{n}$$

Now, how is $H(S^{(n)})$ related to $H(S)$?

See bottom of p. 38

This is where IID assumption is used.

Uses independence

Uses $\log$ (product) = sum of logs

Eventually

$$H(S^{(n)}) = n \cdot H(S)$$

Makes sense:

- each symbol in block gives $H(S)$ bits of info
- independence $\Rightarrow$ no "shared" info between symbols
- $\log$ is additive for independent sequence

$$H(S^{(n)}) = \frac{H(S) + H(S) + \ldots + H(S)}{n \cdot \text{times}}$$

$$= n \cdot H(S)$$
**Final Result for Block Codes**

\[ H(s) \leq R < H(s) + \frac{1}{n} \]

\( n = 1 \) is case of "ordinary" single symbol Huffman we looked at earlier.

<table>
<thead>
<tr>
<th>( H(s) )</th>
<th>( \frac{H(s)}{2} )</th>
<th>( H(s) + \frac{1}{2} )</th>
<th>( H(s) + 1 )</th>
</tr>
</thead>
</table>

As blocks get larger, Rate approaches \( H(s) \).

Thus, longer blocks lead to the "Holy Grail" of compressing down to the entropy.

**BUT** \( n \) of code words grows exponentially: \( n^m \)

\[ \Rightarrow \text{Impractical!} \]