15-853: Algorithms in the Real World

Data compression continued…

Scribe volunteer?
Recap: Encoding/Decoding

Will use “message” in generic sense to mean the data to be compressed.

The encoder and decoder need to understand common compressed format.
Recap: Lossless vs. Lossy

**Lossless**: Input message = Output message

**Lossy**: Input message \(\approx\) Output message

Lossy does not necessarily mean loss of quality. In fact the output could be “better” than the input.

- Drop random noise in images (dust on lens)
- Drop background in music
- Fix spelling errors in text. Put into better form.
Recap: Model vs. Coder

To compress we need a bias on the probability of messages. The model determines this bias.
Recap: Entropy

For a set of messages \( S \) with probability \( p(s), s \in S \), the **self information** of \( s \) is:

\[
i(s) = \log \frac{1}{p(s)} = -\log p(s)
\]

Measured in **bits if the log is base 2**.

**Entropy** is the weighted average of self information.

\[
H(S) = \sum_{s \in S} p(s) \log \frac{1}{p(s)}
\]
Recap: Conditional Entropy

The **conditional entropy** is the weighted average of the conditional self information

\[ H(S \mid C) = \sum_{c \in C} \left( p(c) \sum_{s \in S} p(s \mid c) \log \frac{1}{p(s \mid c)} \right) \]
PROBABILITY CODING
Assumptions and Definitions

Communication (or a file) is broken up into pieces called messages.

Each message comes from a message set \( S = \{s_1, \ldots, s_n\} \) with a probability distribution \( p(s) \).
(Probabilities must sum to 1. Set can be infinite.)

**Code** \( C(s) \): A mapping from a message set to codewords, each of which is a string of bits

**Message sequence**: a sequence of messages
Uniquely Decodable Codes

A **variable length code** assigns a bit string (codeword) of variable length to every message value.

e.g. \(a = 1, \ b = 01, \ c = 101, \ d = 011\)

What if you get the sequence of bits \(1011\)?

Is it \(a\ b\ a,\ c\ a,\ or,\ a\ d\)?

A **uniquely decodable code** is a variable length code in which bit strings can always be uniquely decomposed into its codewords.
Prefix Codes

A *prefix code* is a variable length code in which no codeword is a prefix of another word.

e.g., $a = 0$, $b = 110$, $c = 111$, $d = 10$

Q: Any interesting property that such codes will have?

All prefix codes are uniquely decodable
Prefix Codes: as a tree

a = 0, b = 110, c = 111, d = 10
Ideas?

Can be viewed as a binary tree with message values at the leaves and 0s or 1s on the edges
Codeword = values along the path from root to the leaf
Average Length

Let \( l(c) \) = length of the codeword \( c \) (a positive integer)

For a code \( C \) with associated probabilities \( p(c) \) the **average length** is defined as

\[
l_a(C) = \sum_{c \in C} p(c)l(c)
\]

Q: What does average length correspond to?

We say that a prefix code \( C \) is **optimal** if for all prefix codes \( C' \),  \( l_a(C) \leq l_a(C') \)
Relationship between Average Length and Entropy

**Theorem (lower bound):** For any probability distribution \( p(S) \) with associated uniquely decodable code \( C \),

\[
H(S) \leq l_a (C)
\]

(Shannon’s source coding theorem)

**Theorem (upper bound):** For any probability distribution \( p(S) \) with associated **optimal** prefix code \( C \),

\[
l_a (C) \leq H(S) + 1
\]
Kraft McMillan Inequality

Theorem (Kraft-McMillan): For any uniquely decodable code \( C \),

\[
\sum_{c \in C} 2^{-l(c)} \leq 1
\]

Also, for any set of lengths \( L \) such that \( \sum_{l \in L} 2^{-l} \leq 1 \), there exists a prefix code \( C \) such that

\[
l(c_i) = l_i (i = 1, \ldots, |L|)
\]

(We will not prove this in class. But use it to prove the upper bound on average length.)
Proof of the Upper Bound (Part 1)

To show: \( l_a (C) \leq H(S) + 1 \)

Assign each message a length: \( l(s) = \lceil \log(1/p(s)) \rceil \)

Now we can calculate the average length given \( l(s) \): <board>

\[
l_a(S) = \sum_{s \in S} p(s)l(s) \\
= \sum_{s \in S} p(s) \cdot \lceil \log(1/p(s)) \rceil \\
\leq \sum_{s \in S} p(s) \cdot (1 + \log(1/p(s))) \\
= 1 + \sum_{s \in S} p(s) \log(1/p(s)) \\
= 1 + H(S)
\]
Proof of the Upper Bound (Part 2)

Now we need to show there exists a prefix code with lengths

\[ l(s) = \left\lfloor \log \left( \frac{1}{p(s)} \right) \right\rfloor \]

\[
\sum_{s \in S} 2^{-l(s)} = \sum_{s \in S} 2^{-\left\lfloor \log \left( \frac{1}{p(s)} \right) \right\rfloor} \\
\leq \sum_{s \in S} 2^{-\log \left( \frac{1}{p(s)} \right)} \\
= \sum_{s \in S} p(s) \\
= 1
\]

So by the Kraft-McMillan inequality there is a prefix code with lengths \( l(s) \).
Another property of optimal codes

**Theorem:** If C is an optimal prefix code for the probabilities \( \{p_1, \ldots, p_n\} \) then \( p_i > p_j \) implies
\[
1(c_i) \leq 1(c_j)
\]

**Proof:** (by contradiction)
Assume \( 1(c_i) > 1(c_j) \). Consider switching codes \( c_i \) and \( c_j \).
If \( l_a \) is the average length of the original code, the length of the new code is
\[
l'_a = l_a + p_j(l(c_i) - l(c_j)) + p_i(l(c_j) - l(c_i))
\]
\[
= l_a + (p_j - p_i)(l(c_i) - l(c_j))
\]
\[
< l_a
\]
This is a contradiction since \( l_a \) is not optimal
Huffman Codes

Invented by Huffman as a class assignment in 1950.

Used in many, if not most, compression algorithms gzip, bzip, jpeg (as option), fax compression, Zstd…

Properties:
- Generates optimal prefix codes
- Cheap to generate codes
- Cheap to encode and decode
- \( l_a = H \) if probabilities are powers of 2
Huffman Codes

Huffman Algorithm:
Start with a forest of trees each consisting of a single vertex corresponding to a message $s$ and with weight $p(s)$

Repeat until one tree left:
- Select two trees with minimum weight roots $p_1$ and $p_2$
- Join into single tree by adding root with weight $p_1 + p_2$
### Example

\[ p(a) = .1, \ p(b) = .2, \ p(c) = .2, \ p(d) = .5 \]

- \( a(.1) \)
- \( b(.2) \)
- \( c(.2) \)
- \( d(.5) \)

\[
\begin{align*}
&\text{Step 1} \\
&\quad a(.1) \\
&\quad b(.2) \\
&\quad (3) \\
&\quad \text{Step 2} \\
&\quad a(.1) \\
&\quad b(.2) \\
&\quad (3) \\
&\quad c(.2) \\
&\quad (5) \\
&\quad \text{Step 3} \\
&\quad a(.1) \\
&\quad b(.2) \\
&\quad 0 \\
&\quad (3) \\
&\quad c(.2) \\
&\quad (5) \\
&\quad d(.5) \\
&\quad (1.0) \\
&\quad (1) \\
\end{align*}
\]

\[ a=000, \ b=001, \ c=01, \ d=1 \]
Huffman Codes

Huffman Algorithm:
Start with a forest of trees each consisting of a single vertex corresponding to a message s and with weight $p(s)$

Repeat until one tree left:
- Select two trees with minimum weight roots $p_1$ and $p_2$
- Join into single tree by adding root with weight $p_1 + p_2$
Encoding and Decoding

**Encoding:** Start at leaf of Huffman tree and follow path to the root. Reverse order of bits and send.

**Decoding:** Start at root of Huffman tree and take branch for each bit received. When at leaf can output message and return to root.
Huffman codes are “optimal”

**Theorem:** The Huffman algorithm generates an optimal prefix code.

**Proof outline:**
Induction on the number of messages $n$.
Consider a message set $S$ with $n+1$ messages
1. Can make it so least probable messages of $S$ are neighbors in the Huffman tree
2. Replace the two messages with one message with probability $p(m_1) + p(m_2)$ making $S'$
3. Show that if $S'$ is optimal, then $S$ is optimal
4. $S'$ is optimal by induction
Minimum variance Huffman codes

There is a choice when there are nodes with equal probability.

Any choice gives the same average length, but variance can be different.

<table>
<thead>
<tr>
<th>symbol</th>
<th>probability</th>
<th>code 1</th>
<th>code 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.2</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>b</td>
<td>0.4</td>
<td>1</td>
<td>00</td>
</tr>
<tr>
<td>c</td>
<td>0.2</td>
<td>000</td>
<td>11</td>
</tr>
<tr>
<td>d</td>
<td>0.1</td>
<td>0010</td>
<td>010</td>
</tr>
<tr>
<td>e</td>
<td>0.1</td>
<td>0011</td>
<td>011</td>
</tr>
</tbody>
</table>
Minimum variance Huffman codes

Q: How to combine to reduce variance?

Combine the nodes that were created earliest
Problem with Huffman Coding

Consider a message with probability .999. The self information of this message is

\[-\log(.999) = .00144\]

If we were to send a 1000 such message we might hope to use

\[1000 \times .0014 = 1.44\text{ bits.}\]

Q: Can anybody see the problem with Huffman?
(How many bits do we need with Huffman?)

Using Huffman codes we require at least one bit per message, so we would require 1000 bits.
Discrete or Blended

**Discrete**: each message is a fixed set of bits
- Huffman coding, Shannon-Fano coding

```
01001
11
0001
011
```
message: 1 2 3 4

**Blended**: bits can be “shared” among messages
- Arithmetic coding

```
010010111010
```
message: 1,2,3, and 4
Arithmetic Coding: Introduction

• Allows “blending” of bits in a message sequence.
• Only requires 3 bits for the example above!

• Can bound total bits required based on sum of self information:

<board>

• Used in PPM, JPEG/MPEG (as option), DMM
• More expensive than Huffman coding, but integer implementation is not too bad.
Arithmetic Coding: message intervals

Assign each probability distribution to an interval range from 0 (inclusive) to 1 (exclusive).

e.g. a (0.2), b (0.5), c (0.3)

\[
f(i) = \sum_{j=1}^{i-1} p(j)
\]

The interval for a particular message will be called the message interval (e.g. for b the interval is [.2,.7])
Arithmetic Coding: accumulated prob

E.g.: a (0.2), b (0.5), c (0.3)

Represent message probabilities with \( p(j) \):

\[
p(1) = 0.2, \ p(2) = 0.5, \ p(3) = 0.3
\]

Accumulated probabilities \( f(i) \):

\[
f(i) = \sum_{j=1}^{i-1} p(j)
\]

\[
f(1) = 0.0, \ f(2) = 0.2, \ f(3) = 0.7
\]
Arithmetic Coding: sequence intervals

Code a message sequence by composing intervals. For example: *bac*

The final interval is *[.27,.3)*
We call this the **sequence interval**