Today: Huffman Encoding concluded  
Next: Error-detecting and error-correcting codes

Next week:  
   Algorithms: What are they, how do they process digital information, how do we study them?

Probabilistic Compression: Huffman Encoding

Suppose I have three letters: A, B, C

Normally you would need how many bits to encode messages with these three letters?
### Probabilistic Compression: Huffman Encoding

Suppose I have three letters: A, B, C

Normally you would need how many bits to encode messages with these three letters?

Two bits: $2^2 = 4$

BUT: Suppose we know that the probabilities of the letters are not equal, e.g.,

- A occurs with probability 0.8
- B occurs with probability 0.1
- C occurs with probability 0.1

<table>
<thead>
<tr>
<th>Message</th>
<th>Basic Code</th>
<th>Huffman Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>AABACAAAAAAABAAACAAAAABACAAAAAAA</td>
<td>00 00 01</td>
<td>00 00 01</td>
</tr>
<tr>
<td>ABAAACAAAAAABAAACAAAAABAAAAACAAAAA</td>
<td>00.....</td>
<td>00.....</td>
</tr>
<tr>
<td>AACAAABAAABAAACAAAA....</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Then the following Huffman Code is more efficient:

- A 0
- B 10
- C 11

Let's try the message AABACAAAAA with these two codes

**Basic Code:** 00 00 01 00 10 00 00 00 00 00 00 00 00 00 00 00 (20 bits)

**Huffman Code:** 0 1 0 1 1 0 0 0 0 0 (12 bits)

That is 60% compression..... Instead of each letter taking 2 bits, each on average uses

$$0.8 \times 1 + 0.1 \times 2 + 0.1 \times 2 = 1.2 \text{ bits}$$
This is the basic idea of Huffman Encoding:

Use variable length codes, and encode more frequent letters with shorter codes.

This is actually an old idea, and was used in designing Morse Code for telegraph operators to most quickly transmit messages.

It also used, more or less, to figure out the points for letters in Scrabble:

One messy detail: How do we know where the boundaries between letters occur?

0010011000 = 0 0 10 0 11 0 0 0 0 0

How do we know it is not:

0 01 0 11 00 0 0 0 ?
Probabilistic Compression: Huffman Encoding

One messy detail: How do we know where the boundaries between letters occur?

0010011000 = 0 0 10 0 11 0 0 0 0

How do we know it is not:

0 01 0 11 00 0 00 ?

Those aren’t the codes! Note that there is only one way to “parse” a message using this code:

0010011000

Before talking about how computers use these codes we need to talk about trees......

Digression: Binary Trees

A binary tree is a special kind of graph which is helpful in understanding how binary choices are made in some process....

Since we tend to read and understand things from top to bottom, trees are typically drawn upside down....

Graph: Nodes connected by edges:

Should I eat lunch now?

No

Yes

Stay hungry

Warren Towers

Food Truck

Pig out

Eat Healthy

Fries and pizza at Warren!

Have a salad at Warren.

Go to food truck.

Should I pig out or eat healthy?

Where should I eat?

Food Truck

Eat Healthy

Stay hungry

Fries and pizza at Warren!

Have a salad at Warren.

Go to food truck.

Should I eat lunch now?
A binary tree is a special kind of graph which is helpful in understanding how binary choices are made in some process.

Since we tend to read and understand things from top to bottom, trees are typically drawn upside down.

Graph: Nodes connected by edges:

Should I eat lunch now?
- No
- Yes

Where should I eat?
- Stay hungry
- Warren Towers
- Food Truck

Should I pig out or eat healthy?
- Pig out
- Eat Healthy

Stay hungry

No

Yes

Warren Towers

Food Truck

Pig out

Eat Healthy

Fries and pizza at Warren!

Have a salad at Warren.

Go to food truck.
Digression: Binary Trees

A binary tree is a special kind of graph which is helpful in understanding how binary choices are made in some process.... Since we tend to read and understand things from top to bottom, trees are typically drawn upside down....

Digression: Binary Trees

Binary trees explain why you need 2 bits to encode four letters using the basic code:

Basic Code for 4 Letters:

- A 00
- B 01
- C 10
- D 11

Stringing together the bits you encounter on the path from the root to the symbol gives you the code.
Probabilistic Compression: Huffman Encoding

A computer would use a tree diagram to parse messages with a Huffman Code:

Huffman Code

A  0
B  10
C  11

Algorithm:

1. Start at top (the "root") of the tree and the first bit of the message;

0 0 1 0 0 1 1 0 0 0
Probabilistic Compression: Huffman Encoding

A computer would use a tree diagram to parse messages with a Huffman Code:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>11</td>
</tr>
</tbody>
</table>

Algorithm:
1. Start at top (the "root") of the tree and the first bit of the message;
2. Scan the next bit and take the appropriate branch of the tree;
3. Repeat step 2 until you reach the bottom (a "leaf node"); write down the symbol there;
4. If you are at the end of the message, stop; else go back to the top of the tree and start again at step 2.

0 0 1 0 0 1 1 0 0 0

A
## Probabilistic Compression: Huffman Encoding

A computer would use a tree diagram to parse messages with a Huffman Code:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>11</td>
</tr>
</tbody>
</table>

**Algorithm:**

1. Start at top (the "root") of the tree and the first bit of the message;
2. Scan the next bit and take the appropriate branch of the tree;
3. Repeat step 2 until you reach the bottom (a "leaf node"); write down the symbol there;
4. If you are at the end of the message, stop; else go back to the top of the tree and start again at step 2.

```
0 0 1 0 0 1 1 0 0 0
```

A A
A computer would use a tree diagram to parse messages with a Huffman Code:

### Huffman Code

- A 0
- B 10
- C 11

### Algorithm:

1. Start at top (the "root") of the tree and the first bit of the message;
2. Scan the next bit and take the appropriate branch of the tree;
3. Repeat step 2 until you reach the bottom (a "leaf node"); write down the symbol there;
4. If you are at the end of the message, stop; else go back to the top of the tree and start again at step 2.

```
0 0 1 0 0 1 1 0 0 0
```

**Output:**

```
A A B
```
A computer would use a tree diagram to parse messages with a Huffman Code:

Huffman Code

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>11</td>
</tr>
</tbody>
</table>

Algorithm:

1. Start at top (the "root") of the tree and the first bit of the message;
2. Scan the next bit and take the appropriate branch of the tree;
3. Repeat step 2 until you reach the bottom (a "leaf node"); write down the symbol there;
4. If you are at the end of the message, stop; else go back to the top of the tree and start again at step 2.

0 0 1 0 0 1 1 0 0 0
A A B A
A computer would use a tree diagram to parse messages with a Huffman Code:

Huffman Code

A  0
B  10
C  11

Algorithm:

1. Start at top (the “root”) of the tree and the first bit of the message;
2. Scan the next bit and take the appropriate branch of the tree;
3. Repeat step 2 until you reach the bottom (a “leaf node”); write down the symbol there;
4. If you are at the end of the message, stop; else go back to the top of the tree and start again at step 2.

0 0 1 0 0 1 1 0 0 0
A A B A

A computer would use a tree diagram to parse messages with a Huffman Code:

Huffman Code

A  0
B  10
C  11

Algorithm:

1. Start at top (the “root”) of the tree and the first bit of the message;
2. Scan the next bit and take the appropriate branch of the tree;
3. Repeat step 2 until you reach the bottom (a “leaf node”); write down the symbol there;
4. If you are at the end of the message, stop; else go back to the top of the tree and start again at step 2.

0 0 1 0 0 1 1 0 0 0
A A B A
A computer would use a tree diagram to parse messages with a Huffman Code:

Huffman Code

A  0
B  10
C  11

Algorithm:
1. Start at top (the "root") of the tree and the first bit of the message;
2. Scan the next bit and take the appropriate branch of the tree;
3. Repeat step 2 until you reach the bottom (a "leaf node"); write down the symbol there;
4. If you are at the end of the message, stop; else go back to the top of the tree and start again at step 2.
Probabilistic Compression: Huffman Encoding

A computer would use a tree diagram to parse messages with a Huffman Code:

Huffman Code

A 0
B 10
C 11

Algorithm:
1. Start at top (the "root") of the tree and the first bit of the message;
2. Scan the next bit and take the appropriate branch of the tree;
3. Repeat step 2 until you reach the bottom (a "leaf node"); write down the symbol there;
4. If you are at the end of the message, stop; else go back to the top of the tree and start again at step 2.

0 0 1 0 0 1 1 0 0 0
A A B A C A
A computer would use a tree diagram to parse messages with a Huffman Code:

```
0 0 1 0 0 1 1 0 0 0
```

**Algorithm:**

1. Start at top (the "root") of the tree and the first bit of the message;
2. Scan the next bit and take the appropriate branch of the tree;
3. Repeat step 2 until you reach the bottom (a "leaf node"); write down the symbol there;
4. If you are at the end of the message, stop; else go back to the top of the tree and start again at step 2.
Probabilistic Compression: Huffman Encoding

A computer would use a tree diagram to parse messages with a Huffman Code:

Huffman Code

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>11</td>
</tr>
</tbody>
</table>

Algorithm:

1. Start at top (the "root") of the tree and the first bit of the message;
2. Scan the next bit and take the appropriate branch of the tree;
3. Repeat step 2 until you reach the bottom (a "leaf node"); write down the symbol there;
4. If you are at the end of the message, stop; else go back to the top of the tree and start again at step 2.

0 0 1 0 0 1 1 0 0 0

A A B A C A A A
Probabilistic Compression: Huffman Encoding

So, how do we calculate the Huffman Code?

There is... wait for it... an algorithm to generate a binary tree representing the Huffman Code from the probabilities of the symbols:

1. Write down the letters and their probabilities as single “leaf nodes” (which we will combine into a binary tree);

2. Arrange the nodes from largest (on the left) to smallest (on the right);

3. If there is only one node, go to step 4; else combine the rightmost two nodes using a binary branch, with a node at the top having a probability which is the sum of its two “children”; go back to step 2.

4. Label each left branch with a 0 and each right branch with a 1.

Huffman Coding Algorithm:

1. Write down the letters and their probabilities as single “leaf nodes” (which we will combine into a binary tree);

2. Arrange the nodes from largest (on the left) to smallest (on the right);

3. If there is only one node, go to step 4; else combine the rightmost two nodes using a binary branch, with a node at the top having a probability which is the sum of its two “children”; go back to step 2.

4. Label each left branch with a 0 and each right branch with a 1.
Probabilistic Compression: Huffman Encoding

Huffman Coding Algorithm:

1. Write down the letters and their probabilities as single “leaf nodes” (which we will combine into a binary tree);
2. Arrange the nodes from largest (on the left) to smallest (on the right);
3. If there is only one node, go to step 4; else combine the right-most two nodes using a binary branch, with a node at the top having a probability which is the sum of its two “children”; go back to step 2.
4. Label each left branch with 0 and each right branch with 1.

Huffman Code

A  0.8  0
B  0.1  10
C  0.1  11
Probabilistic Compression: Huffman Encoding

Huffman Coding Algorithm:

1. Write down the letters and their probabilities as single “leaf nodes” (which we will combine into a binary tree);

2. Arrange the nodes from largest (on the left) to smallest (on the right);

3. If there is only one node, go to step 4; else combine the right-most two nodes using a binary branch, with a node at the top having a probability which is the sum of its two “children”; go back to step 2.

4. Label each left branch with 0 and each right branch with 1.

Huffman Code

<table>
<thead>
<tr>
<th>Letter</th>
<th>Probability</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.8</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0.1</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>0.1</td>
<td>11</td>
</tr>
</tbody>
</table>

Diagram:

```
    0
   /|
  1 0
 /|
A B C
```

Probabilistic Compression: Huffman Encoding

Huffman Coding Algorithm:

1. Write down the letters and their probabilities as single “leaf nodes” (which we will combine into a binary tree);

2. Arrange the nodes from largest (on the left) to smallest (on the right);

3. If there is only one node, go to step 4; else combine the right-most two nodes using a binary branch, with a node at the top having a probability which is the sum of its two “children”; go back to step 2.

4. Label each left branch with 0 and each right branch with 1.
Probabilistic Compression: Huffman Encoding

For English, 26 letters would take 5 bits, but with Huffman Encoding, you can reduce this to 4.15 bits, a compression ratio of $\frac{4.15}{5} = 83\%$.

Most practical uses of compression use multiple techniques, e.g., fax machines use Run-Length Encoding combined with Huffman Encoding for black and white documents.

<table>
<thead>
<tr>
<th>Char</th>
<th>$f_x$</th>
<th>Code $e_x$</th>
<th>$f_{e_x}$</th>
<th>Code $e_{e_x}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>850</td>
<td>00</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>665</td>
<td>1111</td>
<td>308</td>
<td>1111</td>
</tr>
<tr>
<td>S</td>
<td>388</td>
<td>1101</td>
<td>275</td>
<td>0110</td>
</tr>
<tr>
<td>I</td>
<td>366</td>
<td>1100</td>
<td>286</td>
<td>0111</td>
</tr>
<tr>
<td>T</td>
<td>356</td>
<td>1011</td>
<td>473</td>
<td>0001</td>
</tr>
<tr>
<td>N</td>
<td>355</td>
<td>1010</td>
<td>320</td>
<td>1100</td>
</tr>
<tr>
<td>R</td>
<td>305</td>
<td>1001</td>
<td>308</td>
<td>1011</td>
</tr>
<tr>
<td>U</td>
<td>295</td>
<td>0111</td>
<td>111</td>
<td>00010</td>
</tr>
<tr>
<td>L</td>
<td>278</td>
<td>0110</td>
<td>153</td>
<td>10100</td>
</tr>
<tr>
<td>Q</td>
<td>255</td>
<td>0100</td>
<td>250</td>
<td>1110</td>
</tr>
<tr>
<td>O</td>
<td>200</td>
<td>1110</td>
<td>171</td>
<td>11011</td>
</tr>
<tr>
<td>C</td>
<td>148</td>
<td>10000</td>
<td>124</td>
<td>00100</td>
</tr>
<tr>
<td>M</td>
<td>145</td>
<td>01101</td>
<td>114</td>
<td>00013</td>
</tr>
<tr>
<td>P</td>
<td>144</td>
<td>01100</td>
<td>89</td>
<td>00000</td>
</tr>
<tr>
<td>V</td>
<td>75</td>
<td>100010</td>
<td>41</td>
<td>101000</td>
</tr>
<tr>
<td>Q</td>
<td>61</td>
<td>1111110</td>
<td>5</td>
<td>11010001</td>
</tr>
<tr>
<td>G</td>
<td>54</td>
<td>1111101</td>
<td>90</td>
<td>00003</td>
</tr>
<tr>
<td>F</td>
<td>46</td>
<td>1111100</td>
<td>132</td>
<td>01011</td>
</tr>
<tr>
<td>B</td>
<td>42</td>
<td>1000111</td>
<td>65</td>
<td>101000</td>
</tr>
<tr>
<td>H</td>
<td>35</td>
<td>1111111</td>
<td>237</td>
<td>01000</td>
</tr>
<tr>
<td>J</td>
<td>29</td>
<td>11111101</td>
<td>6</td>
<td>110100011</td>
</tr>
<tr>
<td>X</td>
<td>17</td>
<td>10001100</td>
<td>7</td>
<td>1101000111</td>
</tr>
<tr>
<td>Y</td>
<td>10</td>
<td>100011010</td>
<td>69</td>
<td>1101001</td>
</tr>
<tr>
<td>Z</td>
<td>8</td>
<td>100010111</td>
<td>3</td>
<td>1101000100</td>
</tr>
<tr>
<td>K</td>
<td>2</td>
<td>1000101101</td>
<td>19</td>
<td>110100000</td>
</tr>
<tr>
<td>W</td>
<td>1</td>
<td>1000110100</td>
<td>68</td>
<td>1101001</td>
</tr>
</tbody>
</table>

Table 1: Letter frequencies and encodings for French, $f_x$, and English, $f_{e_x}$, using the standard distribution. The frequency number is the number of times the letter showed up in a sample of 1,000 words. The encoding is the output of Huffman algorithm.