Huffman (source/noiseless) coding

(Emulate a **tree construction** in words.) Huffman encoding takes a list of probabilities and creates a **binary tree** by repeatedly taking the two smallest probabilities and creating a **parent** node for them, assigning the sum of their probabilities to that parent node. One of the two nodes is the **left child** and the other is the **right child**. (Here it doesn't matter which, but a specific decision process is necessary.) Repeat, each time looking at the list of nodes which have no parent. When the tree is complete, when there is a single node (the **root node**) without a parent, the process goes back *down* the tree recursively to assign codewords: the root node is assigned the empty string " as codeword. Left child of node with codeword 'w' is assigned codeword 'w0' and right child is assigned 'w1'. This labeling percolates down the tree, giving the Huffman encoding.

Example: Huffman encode source with probabilities 1/4, 1/5, 1/5, 1/6, 1/8, 7/120.

Two smallest probabilities 7/120 and 1/8 combine to 11/60. List becomes $(11/60, 1/6, 1/5, 1/5, 1/4)$.
Two smallest 1/6 and 11/60 combine to 7/20.
List now (7/20, 1/5, 1/5, 1/4).
Two smallest 1/5 and 1/5 combined to 2/5.
List now (2/5, 1/4, 7/20).
Two smallest 1/4 and 7/20 combined to 3/5.
List now (3/5, 2/5).
2/5 child of 1=" gets codeword '0'.
1/5 child of 2/5='0' gets codeword '00'.
1/5 child of 2/5='0' gets codeword '01'.
3/5 child of 1=" gets codeword '1'.
1/4 child of 3/5='1' gets codeword '10'.
7/20 child of 3/5='1' gets codeword '11'.
1/6 child of 7/20='11' gets codeword '110'.
11/60 child of 7/20='11' gets '111'.
7/120 child of 11/60='111' gets '1110'.
1/8 child of 11/60='111' gets '1111'.
The list of probabilities and codewords is
(1/4='10', 1/5='00', 1/5='01', 1/6='110',
1/8='1111', 7/120='1110').

Example: Huffman encode source with probabilities 1/3, 1/5, 1/6, 1/7, 1/8, 9/280.

Two smallest probabilities 9/280 and 1/8 which get combined to 11/70. List now (11/70, 1/7, 1/6, 1/5, 1/3).
Two smallest 1/7 and 11/70 combine to 3/10. List now (3/10, 1/6, 1/5, 1/3).
Two smallest 1/6 and 1/5 combine to 11/30. List now (11/30, 3/10, 1/3).
Two smallest probabilities 3/10 and 1/3 combine to 19/30. List now (19/30, 11/30).
11/30 child of 1=" gets codeword '0'.
1/6 child of 11/30='0' gets codeword '00'.
1/5 child of 11/30='0' gets codeword '01'.
19/30 child of 1=" gets codeword '1'.
3/10 child of 19/30='1' gets codeword '10'.
1/7 child of 3/10='10' gets codeword '100'.
11/70 child of 3/10='10' gets codeword '101'.
9/280 child of 11/70='101' gets '1010'.
1/8 child of 11/70='101' gets codeword '1011'.
1/3 child of 19/30='1' gets '11'.
Encodings: (1/3='11', 1/5='01', 1/6='00', 1/7='100', 1/8='1011', 9/280='1010').

Remarks: Huffman coding is optimal exactly and only that for *specified* source words with specified probabilities it achieves the lowest possible expected word length (estimated in Shannon's theorem).

• Huffman encoding does not worry about good or bad choices of what to use as source words. The point of many contemporary compression algorithms is exactly this choice. Once that choice is made, the Huffman part is easy.

• We are neglecting the important question of **decoding** algorithms, just because that is a different story. But ease of decoding is an important criterion, sometimes more important than compression. In real life, there is a trade-off between the two considerations.

• In fact, smart decoding algorithms are often fairly wacky, probabilistic, and not obviously intuitive. A topic in their own right. In some cases, there are very good algorithms that *seem to* work far better than anyone can *prove*.

Error detection, error correction

Now a complementary problem, *detection* of errors, and *correction* of errors.

As always, we can certainly think of the literal idea that a message is sent over a channel which is a wire, or wireless, but the idea can be abstracted. Your notes, your mind, and many other things are channels in abstraction.

The simplest type of channels is **discrete memoryless channels** C. There is a finite **input alphabet** $\Sigma_{\text{in}} = \{x_1, \ldots, x_m\}$ and finite **output alphabet** $\Sigma_{\text{out}} = \{y_1, \ldots, y_n\}$. When input x_i is sent into the channel, output y_j is received with probability p_{ij}.

$$p_{ij} = P_C(\text{received} = y_j | \text{sent} = x_i)$$

These are conditional probabilities.

Suppose the channel operates so that transmission and receipt of each character are **independent** of the transmission and receipt of other characters. So probabilities are independent of what comes before or after.

Necessarily

$$\sum_j p_{ij} = 1$$

since the sum of probabilities of possible output characters that might be received (for given input x_i) is 1.

The collection

$$M = \{p_{ij} : 1 \leq i \leq m, 1 \leq j \leq n\}$$

is the **channel matrix**. The probability p_{ij} is the entry in the i^{th} row and j^{th} column. A matrix such that rows and columns are non-negative and sum to 1 is a **stochastic matrix**.

Remark: Stochastic matrices also occur in *Markov processes*, and are called **transition probabilities**.

Example:

The **binary symmetric channel** model is the simplest meaningful example.

Input alphabet is $\Sigma_{\text{in}} = \{0, 1\}$ and output alphabet is $\Sigma_{\text{out}} = \{0, 1\}$.

Probability that the channel transmits '0' as '1' or '1' as '0' (that is, makes a mistake) is p.

Probability that a character is transmitted *correctly* is $1 - p$.

The quantity p is the **(bit) error probability** of the channel.

(Often q is a convenient shorthand for $1 - p$.)

Remark: As simple as this example is, it already can be made to illustrate nearly all interesting phenomena.

Example: Erasure channel: This model includes the idea that a channel may lose some characters entirely.

For example, with input alphabet $\Sigma_{\text{in}} = \{0, 1\}$ the output alphabet would be $\Sigma_{\text{out}} = \{0, 1, *\}$ where $*$ is an **erasure** of a character. Let ε be a small positive real number, and let the transition probabilities be

in\out	0	1	$*$
0	$1-\varepsilon$	0	ε
1	0	$1-\varepsilon$	ε

That is, the two characters '0' and '1' never transmute into each other, but either one may be *erased* with probability ε. This is the **binary erasure channel** with erasure probability ε.

The N^{th} **extension** $C^{(N)}$ of a channel C is a channel whose input alphabet is all N-tuples of characters from the input alphabet of C, whose output alphabet is the collection of N-tuples of characters from the output alphabet of C, and transition probabilities correspond to having N copies of the original channel working independently, in parallel. That is,

$$P_{C^{(N)}}(\text{out} = b_1 \ldots b_N | \text{in} = a_1 \ldots a_N)$$

$$= P_C(\text{out} = b_1 | \text{in} = a_1) \times$$

$$\ldots \times P_C(\text{out} = b_N | \text{in} = a_N)$$

We consider a source X emitting words $w_1, \ldots, w_m$ with probabilities $p_i = P(X = x_i)$ encoded (perhaps by Huffman) into binary, sent across a binary symmetric channel C, and decoded on the other side.

The encoding to binary is noiseless and is known to the decoder.

The channel bit-error probability often denoted p is known.

With independence assumptions, probability that N bits $a_1 \ldots a_N$ will be transmitted correctly is the product of the probability that a_1 is transmitted correctly, that a_2 is transmitted correctly, ..., and that a_N is transmitted correctly:

$$(1-p)^N$$

The fundamental question is **can we do better than this?**

(Of course, the answer is 'yes' or we wouldn't *be* here talking about it.)

There are two parts to the question of improvement: **detection** of errors, and **correction** of errors.

Detection is easier than correction.

Sometimes *detection* is good enough: maybe we can ask for a retransmission.

Sometimes (in deep space transmission, video, audio, and other examples) retransmission is infeasible or impossible or ridiculous.

In other cases *correction* of errors is necessary based on the original but flawed transmission. This is **forward error correction**.

Example of detection: parity checks

To detect **single bit errors** use a **parity check**.

This approach is so simple that it is often used.

But it does not *correct* errors.

As a *detection* device it is somewhat limited.

All **single-bit errors** in a message or block can be detected by a parity-check.

For binary encoding $f : X \to \{0,1\}^*$ of a source into strings of 0s and 1s, we can add a **check-bit** and detect single-bit errors. Replace encoding f by encoding $\tilde{f}$

$$\begin{array}{rcll} \tilde{f}(w) & = & f(w) + \text{`0'} & f(w) \text{ has } \textit{even} \text{ no. 1's} \\ \tilde{f}(w) & = & f(w) + \text{`1'} & f(w) \text{ has } \textit{odd} \text{ no. 1's} \end{array}$$

Decoding rejects a word if the last bit does not correctly reflect the odd/even-ness of the rest of the word. Equivalently, *every* augmented word should have an *even* number of 1's.

Remark: If the code is *instantaneous* then adding this parity-check bit does not create any conflicts.

Change of a single bit (including the check bit) of the new codewords can be **detected**. That is, *all single-bit errors are detected by a parity-check bit.*

Example: A source emits 2-bit binary codewords $00, 01, 10, 11$. Suppose the binary symmetric channel has bit error probability $1/8$. Then the probability that *at least one bit error* occurs (and will necessarily be undetected!) in transmission of one of these 2-bit words is

$$P(\geq 1 \text{ error}) = P(1 \text{ error}) = P(2 \text{ errors})$$
$$= \binom{2}{1} \frac{1}{8} \cdots \frac{7}{8} + \frac{1}{8} \cdot \frac{1}{8} = \frac{15}{64} \approx 0.234$$

Add a parity-check bit by replacing these words with $000, 011, 101, 110$. What errors are detected?

Single bit errors are detected. In fact, if any *odd* number of bits are flipped, this will be detected. But change of 2 bits will *not* be detected. So probability of *undetected* error is the probability of exactly 2 bit errors (out of 3)

$$\binom{3}{2} \frac{1}{8} \frac{1}{8} \frac{7}{8} = \frac{21}{512} \approx 0.041 << 0.234$$

Remark: But this detection scheme does *not* tell how to *correct* an error, only that it *occurred*.

Example: What happens with more noise on the channel? Again use codewords $00, 01, 10, 11$, but take the channel's bit error probability to be $1/3$. The probability that *at least one bit error* occurs is

$$\frac{1}{3} \cdot \frac{2}{3} + \frac{2}{3} \cdot \frac{1}{3} + \frac{1}{3} \cdot \frac{1}{3} = \frac{5}{9} \approx 0.5555 > \frac{1}{2}$$

Unlikely we'd have a succesful transmission!

A single parity-check replaces these words with $000, 011, 101, 110$. Single bit (and 3-bit) errors are detected, but 2-bit errors are not. So the probability of *undetected* error is

$$\binom{3}{2} \left(\frac{1}{3}\right)^2 \left(\frac{2}{3}\right) = \frac{6}{27} \approx 0.22222 << 0.555$$

Remark: The probability of undetected error is now below $1/2$, but still high.

Remark: We still have no means of *correcting* an error even though we may know an error occurred.

Example: A channel as noisy as possible: take bit error probability $1/2$, codewords $00, 01, 10, 11$. The probability that *at least one bit error* occurs in transmission is

$$\binom{2}{1}\frac{1}{2}\frac{1}{2} + \binom{2}{2}\left(\frac{1}{2}\right)^2 = \frac{3}{4} = 0.75 > \frac{1}{2}$$

Adding a single parity-check bit replaces the words with $000, 011, 101, 110$, respectively. Single bit errors (and 3-bit) are detected. 2-bit are not. The probability of at least one *undetected* bit error is

$$\binom{3}{2}\left(\frac{1}{2}\right)^2\left(\frac{1}{2}\right) = \frac{3}{8} = 0.375 < 0.5$$

Thus, parity-check bits reduce the probability of *undetected* bit error within a word to $3/8$, significantly below $1/2$, even with a maximally noisy channel.

Remark: This is *detection*, not *correction*. Perhaps the messages can be *retransmitted*. Retransmission is the mechanism by which TCP makes IP robust, though not just using a simple check-bit.

Example: Consider 3-bit binary words transmitted over a binary symmetric channel with bit error probability 1/8. The probability that at least one bit error will occur in transmission of a 3-bit word is

$$\binom{3}{1}\left(\frac{1}{8}\right)\left(\frac{7}{8}\right)^2+\binom{3}{2}\left(\frac{1}{8}\right)^2\left(\frac{7}{8}\right)+\binom{3}{3}\left(\frac{1}{8}\right)^3$$
$$\approx 0.33$$

With a parity-check bit, the probability of an *undetected* error is the probability of 2 (or 4) bit errors, namely

$$\binom{4}{2}\left(\frac{1}{8}\right)^2\left(\frac{7}{8}\right)^2+\binom{4}{4}\left(\frac{1}{8}\right)^4 \approx 0.072 << 0.33$$

Remark: Mere *detection* of errors may be silly or worthless if it is impossible or expensive to retransmit, as in deep space transmissions, and video, especially live video, or with a high volume of highly structured or synchronized information. So we'll care mostly about not just detection but *correction* of errors.
