Decoding from a noisy channel

One (eventually discarded) try at decoding messages sent through a noisy channel is the following. Let $x_1, \ldots, x_m$ be the source words, and suppose y is received. We might decode y as x_{i_0} where x_{i_0} is the source word such that

$$P(x_{i_0} \text{ sent}|y \text{ received}) \geq P(x_i \text{ sent}|y \text{ received})$$

That is, given that y was received, the (conditional) probability that x_{i_0} was sent is the greatest among the x_is. This is the **ideal observer** or **minimum-error** rule.

Remark: This rule seems reasonable but has a fatal flaw: the receiver must know the probabilities that x_i is sent.

Therefore, do not try to use this rule.

A better rule is the **maximum-likelihood** ('ML') decoding rule, which decodes a received word y into x_i to maximize

$$P(y \text{ received}|x_i \text{ sent})$$

We do not need to know the probabilities that words x_i are sent.

For a binary symmetric channel maximum-likelihood decoding can be described in terms of the **Hamming distance** between strings of 0s and 1s (after proving a little result).

The **Hamming distance** $d(x, y)$ between two binary vectors $x = (x_1, \ldots, x_n)$, $y = (y_1, \ldots, y_n)$ of the same length is

$$d(x, y) = \text{ number of indices } i \text{ so that } x_i \neq y_i$$

The **Hamming weight** of a binay vector is the number of entries that are 1.

Minimum-distance decoding decodes a received word as the codeword x_i closest (in Hamming distance) to y.

Proposition: The Hamming distance $d(,)$ among binary strings of a fixed length behaves like a 'real' distance function in that it has properties

- $d(x, x) = 0$ for any string x, and $d(x, y) = 0$ implies $x = y$.
- (Symmetry) $d(x, y) = d(y, x)$
- (Triangle inequality) $d(x, z) \leq d(x, y) + d(y, z)$

Proof: The first two assertions are easy. For the third, look at the i^{th} bit in all three strings. If x and z differ at the i^{th} bit, then either x and y differ at the i^{th} bit, or z and y differ at the i^{th} bit. Thus, adding up these differences over locations i^{th}, we have an analogous inequality for all i, so the sums satisfy the same inequality.

///

At first glance maximum-likelihood and minimun-distance may not be the same, but they turn out to be identical:

Theorem: For binary symmetric channel with bit error probability $p < \frac{1}{2}$, minimum-distance decoding is equivalent to maximum-likelihood.

Proof: Let x be a possible decoding of a received y. The probability that x became y is $p^{d(x,y)}(1-p)^{n-d(x,y)}$ since $d(x,y)$ bits flip. Since $p < \frac{1}{2}$, $p/(1-p) < 1$, so if $d(z,y) > d(x,y)$

$$p^{d(x,y)}(1-p)^{n-d(x,y)}$$

$$\geq p^{d(x,y)}(1-p)^{n-d(x,y)} \cdot \left(\frac{p}{1-p}\right)^{d(z,y)-d(x,y)}$$

$$= p^{d(z,y)}(1-p)^{n-d(z,y)}$$

So the probability that x became y is greatest when x is closest to the received word y. ///

So always use minimum-distance decoding.

Example: Given codewords $a = 1001$, $b = 0111$, $c = 0001$, and received word $y = 1111$, how should we decode y?

Part of the question is answered by recalling that we use *minimum-distance* (=maximum-likelihood) decoding. That is, use Hamming distance (the number of bits differing in two words) $d(,)$ and decode the received word y as the codeword closest to it in Hamming distance.

Compute the Hamming distances by comparing respective bits, adding 1 for each differing bit:

$$d(a, y) = d(1001, 1111) = 0 + 1 + 1 + 0 = 2$$

$$d(b, y) = d(0111, 1111) = 1 + 0 + 0 + 0 = 1$$

$$d(c, y) = d(0001, 1111) = 1 + 1 + 1 + 0 = 3$$

Thus, the received word y is closest to codeword c (in Hamming distance), so **decode** $y = 1111$ **as** $b = 0111$**.**

Example: A three-word message is encoded by $a = 1000011$, $b = 0100101$, $c = 0010110$, $d = 0001111$, $e = 1100110$, and $f = 1010101$, $g = 1001100$. The message is sent across a noisy channel, and you receive '111011010001101001101'. What was the most likely original message?

The message is considered as three 7-bit words in a row, each of which is a mangled form of one a codewords. We decode each mangled 7-bit received word by *minimum-distance decoding*, using Hamming distance (which counts the differing bits), finding the codeword which differs from it by the least number of bits.

Shortcuts: By a *one-time pre-computation*, the codewords have Hamming distances as little as 3 from each other. Hoping for unambiguous decoding, *only consider codewords of Hamming distance 0 or 1 from the received words.* If there is none, then decoding fails. **And** if we find *one* codeword at distance ≤ 1, we decode as that codeword **and stop**.

The following results illustrate the utility of the intuition attached to the idea of *distance*:

Theorem: In general, when codewords have distances at least 3 from each other, for a given received word y there cannot be two codewords x, z both at distance ≤ 1 from y.

Proof: Suppose $d(x,y) = d(y,z) = 1$ but $d(x,z) \geq 3$. Then by the triangle inequality

$$3 \leq d(x,z) \leq d(x,y) + d(y,z) = 1 + 1$$

contradiction. ///

Similarly:

Theorem: More generally, when codewords have distances at least $2k+1$ from each other, for a given received word y there cannot be two codewords x, z both at distance $\leq k$ from y.

Proof: Suppose $d(x,y) = d(y,z) \leq k$ but $d(x,z) \geq 2k+1$. Then by the triangle inequality

$$2k+1 \leq d(x,z) \leq d(x,y) + d(y,z) = k + k$$

contradiction. ///

In the example, instead of computing the Hamming distance from a received word to *all* codewords, **stop** as soon as distance ≤ 1.

Further: *gradually* compare bits from left to right and reject a codeword as soon as it differs by 2 or more bits from the received word.

And, **again**, as soon as a codeword is at distance $\leq\ 1$ from the received word, we decode as that codeword **and do not continue computing distances**.

The general analogues of these two shortcuts apply when the minimum distance between codewords is $2k+1$:

When a codeword is within k of the received word, decode as that word and stop. This cuts in half the expected number of comparisons.

Further, compare the received word and codewords bit-by-bit, and as soon as the number of differing bits exceeds k, reject that codeword without further comparison. This is another significant speedup.

The first received word '1110110' (the first 7 bits of the whole string) differs from a = 1000011 at the 2nd and 3rd bits, so reject a. It differs from $b = 0100101$ at 1st and 3rd, so drop b. It differs from $c = 0010110$ at 1st and 2nd, so drop c. It differs from $d = 0001111$ at 1st and 2nd, so drop d. It differs *only* at 3rd, from $e = 1100110$ so has Hamming distance 1 from e. We decode 1110110 as $e = 1100110$ and **stop**, not even measuring the distance of 1110110 to $f = 1010101$ and $g = 1001100$.

Similar computations apply to the second and third batches of 7 bits from the received message.

Summarizing,

1110110 closest (only 2nd differs) 1100110 = e
1000110 closest (only 1st differs) 1100110 = e
1001101 closest (only 6th differs) 1001100 = g

Thus, the decoding of the message '111011010001101001101' is **'eeg'**.

Channel capacity

Part of Shannon's theorem about error-correction is a precise meaning for **channel capacity** (to carry information).

Let C be a memoryless discrete channel with input alphabet Σ_{in} and output alphabet Σ_{out} and for $x_i \in \Sigma_{\text{in}}$ and $y_j \in \Sigma_{\text{out}}$ transition probabilities

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

Let source X emit elements of Σ_{in} and

$$p_i = P(X \text{ emits } x_i)$$

The **output** of the channel C with X connected to its input is a memoryless source Y emitting Σ_{out} with probabilities

$$p'_j = \sum_{i=1}^{m} P(y_j \text{ received } | x_i \text{ sent })$$

$$P(X \text{ sent } x_i) = \sum_{i=1}^{m} p_{ij} p_i$$

The **information about X given Y** is the decrease in entropy

$$I(X|Y) = H(X) - H(X|Y)$$

$$= H(X) + H(Y) - H(X,Y)$$

Remark: The expression for $I(X|Y)$ is symmetrical

$$I(X|Y) = I(Y|X)$$

so the amount of information about X imparted by Y is equal to the amount of information about Y imparted by X.

The **channel capacity** is

$$\text{capacity } (C) = \max_X I(X|Y)$$

with max over all probability distributions for sources emitting the given alphabet accepted as inputs by the channel.

Remark: This is not a *computationally useful* definition.

Remark: Capacity is a continuous function on the closed and bounded set of probabilities $p_1, \ldots, p_m$, so the maximum exists. From calculus the max of a continuous function on a closed and bounded set in $\mathbf{R}^m$ is achieved.

Remark: *Units* for channel capacity are **bits per symbol**.

Theorem: *(Shannon)* Channel capacity of a binary symmetric channel with bit error probability p is

$$1 - H(p, 1-p) = 1 + p \log_2 p + (1-p) \log_2(1-p)$$

Remark: This makes channel capacity computable!

Remark: Sensibly, when $p = \frac{1}{2}$ channel capacity is 0, since what we get over the channel is worthless. We can *detect* errors (by parity-check bits) but cannot *correct* them. Similarly, reasonably-enough:

Proposition: Let C be a memoryless channel with capacity c. Then for any positive integer n the n^{th} *extension* $C^{(n)}$ of C has capacity nc.

Examples: Values of channel capacity for varying bit-error probability p:

bit-err prob	channel cap
.01	0.92
.02	0.86
.04	0.76
.05	0.71
.06	0.67
.07	0.63
.08	0.60
0.1	0.53
0.2	0.28
0.3	0.12
0.4	0.03
.45	0.007
0.5	0.00

Remark: This function is not linear.

Remark: For bit-error rate 1/2 or anything close to it, the channel capacity approaches 0.000 quite rapidly. Not linearly.

Shannon's noisy coding theorem

Shannon's 1948 theorem proves that there *exists* an **error-correcting** encoding so that information can be sent through a noisy channel at a rate arbitrarily close to the *capacity* of the channel.

Word error probability of encoding f is *average* probability of error in decoding, weighted-averaging over source words $w_1, \ldots, w_N$ This is not a good model, since an assumption of equal probability is invariably stupid, and we might not know the probabilities.

A better measure to minimize is

maximum word error probability

$$= \max_i P(\text{error}|w_i \text{ sent})$$

If *max* prob error prob is small, then *avg* word error prob is small, since

$$\begin{aligned} &\text{maximum word error probability of } f \\ \geq\ &\text{average word error probability of } f \end{aligned}$$

We now emphasize **binary** codes, so everything is 0's and 1's.

We think of a **binary symmetric channel** (and, without explicit mention, its extensions to process a stream of bits), whose nature is completely described by the single parameter p, the bit-error probability.

Always use **maximum-likelihood** (equivalently, **minimum-distance**) decoding.

From Shannon, a symmetric binary channel C with bit error probability p has capacity

$$c = 1 + p \log_2 p + (1-p) \log_2 (1-p)$$

Definition: The **rate** of a binary code with maximum word length n with t codewords is defined to be

$$\text{rate} \;=\frac{\log_2 t}{n}=\frac{\log_2(\text{number codewords})}{\text{max word length}}$$

Remark: The maximum possible rate is 1, which can occur only for a binary code with maximum word length n where *all* the 2^n binary codewords of length n are used in the code. This represents the fullest possible transmission of information through a channel.

Remark: In a **noisy** channel where the bit error probability is >0 it is unreasonable to use a code with info rate too close to 1, because such a code will not have enough *redundancy* to either **detect** or **correct** errors.

Example: For binary code 001, 110, 010, 101

$$\text{info rate} = \frac{\log_2(\text{no. codewords})}{\text{max length}} = \frac{\log_2 4}{3} = \frac{2}{3}$$

Example: For binary code 001, 110, 010

$$\text{info rate} = \frac{\log_2(\text{no. codewords})}{\text{max length}}$$

$$= \frac{\log_2 3}{3} \approx 0.585$$

Examples:

For three-fold binary **repetition code** 111, 000

$$\text{info rate} = \frac{\log_2(\text{no. codewords})}{\text{max length}} = \frac{\log_2 2}{3} = \frac{1}{3}$$

For 5-fold binary **repetition code** 11111, 00000

$$\text{info rate} = \frac{\log_2(\text{no. codewords})}{\text{max length}} = \frac{\log_2 2}{5} = \frac{1}{5}$$

Remarks: Repetition codes can correct errors by **majority vote/logic**, meaning assume that the majority of bits are correct.

But repetition codes are very inefficient, since they have a very low information rate.

Theorem: *(Noisy Coding)* For symmetric binary channel C with bit error probability $p < \frac{1}{2}$, let R be an info rate

$$0 < R < 1 + p \log_2 p + (1-p) \log_2 (1-p)$$

There is a sequence $C_1, C_2, \ldots$ of codes of lengths n_i with rates R_i approaching R such that

$$\lim_i \text{ word length } (C_i) = \infty$$

$$\lim_i \text{ max word error probability } (C_i) = 0$$

More specifically, given $\varepsilon > 0$, for sufficiently large n there is a code C of length n with rate $R_0 \leq R$ such that

$$|R_0 - R| \leq \frac{1}{n}$$

and

$$\text{max word error probability } (C) < \varepsilon$$

Remark: The unusual nature of the proof gives no explanation of how to *find* or *create* the codes, nor how rapidly the maximum word error probability decreases to 0.

Shannon's amazing insight was that whatever the *average* value P_{avg} of P_C, averaged over all length n codes C with t codewords, there must be at least one code C_0 which has

$$P_{C_0} \leq P_{\text{avg}}$$

This is elementary: let $a_1, \ldots, a_N$ be real numbers, with average

$$A = \frac{a_1 + \ldots + a_N}{N}$$

We claim that there is at least one a_i (though we do not know which) with $a_i \leq A$. If $a_i > A$ for all a_i, then

$$a_1 + \ldots + a_N > A + \ldots + A = N \cdot A$$

and

$$\frac{a_1 + \ldots + a_N}{N} > A$$

contradicting the fact that equality holds (since A is the average).

Remarks:

Only in the last decade or two has there been much systematic success in finding codes that approach the Shannon bound.

Length 7 Hamming codes were the first good codes found, about 1950. But these do not scale up well, giving only good *small* codes.

Reed-Solomon (RS) and Bose-Hocquengham-Chaudhuri (BCH) codes were and are reasonably good medium-small codes, and are still in use.

It turns out that making good error-correcting codes seems to be a much harder problem than compression issues.
