Where Do the Probabilities Come From?

So far, we've assumed that we "just know" the probabilities of the symbols, $p_1, \ldots, p_I$. Note: The transmitter and the receiver must both know the *same* probabilities.

This isn't realistic. For instance, if we're compressing black-and-white images, there's no reason to think we know beforehand the fraction of pixels in the transmitted image that are black.

But could we make a good guess? That might be better than just assuming equal probabilities. Most fax images are largely white, for instance. Guessing $P(\text{White}) = 0.9$ may usually be better than $P(\text{White}) = 0.5$.

The Penalty for Guessing Wrong.

Suppose we use a code that would be optimal if the symbol probabilities were $q_1, \ldots, q_I$, but the real probabilities are $p_1, \ldots, p_I$. How much does this cost us?

Assume we use large blocks or use arithmetic coding — so that the code gets down to the entropy, given the assumed probabilities.

We can compute the difference in expected code length between an optimal code based on $q_1, \ldots, q_I$ and an optimal code based on the real probabilities, $p_1, \ldots, p_I$, as follows:

$$\sum_{i=1}^{I} p_i \log(1/q_i) \;-\; \sum_{i=1}^{I} p_i \log(1/p_i) \;=\; \sum_{i=1}^{I} p_i \log(p_i/q_i)$$

This is the *relative entropy* of $\{p_i\}$ and $\{q_i\}$. It can never be negative. (See Section 2.6 of MacKay's book.)

Why Not Estimate the Probabilities, Then Send Them With the Data?

One way to handle unknown probabilities is to have the transmitter *estimate* them, and then send these probabilities along with the compressed data, so that the receiver can uncompress the data correctly.

Example: We might estimate the probabilitiy that a pixel in a black-and-white image is black by the *fraction* of pixels in the image we're sending that are black.

One problem: We need some code for sending the estimated probabilities. How do we decide on that? We need to guess the probabilities for the different probabilities...

Why This Can't be Optimal

This scheme may sometimes be a pragmatic solution, but it can't possibly be optimal, because the resulting code isn't *complete*.

In a complete code, all sequences of code bits are possible (up to when the end of message is reached). A prefix code will not be complete if some nodes in its tree have only one child.

Suppose we send a 3-by-5 black-and-white image by first sending the number of black pixels (0 to 15) and then the 15 pixels themselves, as one block, using probabilities estimated from the count sent.

Some messages will not be possible, eg:

```
4   ○ ● ● ○ ○
    ● ● ● ● ○
    ○ ○ ● ● ●
```

This can't happen, since the count of 4 is inconsistent with the image that follows.

Adaptive Models

We can do better using an *adaptive* model, which continually re-estimates probabilities using counts of symbols in the *earlier* part of the message.

We need to avoid giving any symbol zero probability, since its "optimal" codeword length would then be $\log(1/0) = \infty$. One "kludge": Just add one to all the counts.

Example: We might encode the 107th pixel in a black-and-white image using the count of how many of the previous 106 pixels are black.

If 13 of these 106 pixels were black, we encode the 107th pixel using

$$P(\text{Black}) = (13+1)/(106+2) = 0.1308$$

Changing probabilities like this is easy with arithmetic coding, harder with Huffman codes, especially if we encode blocks of symbols.

Why This Isn't Just a Kludge

Adding one to all the counts may seem like a horrible kludge for avoiding probabilities of zero. But it is actually one of the methods that can be justified by the statistical theory of *Bayesian inference*.

Bayesian inference uses probability to represent uncertainty about anything — not just which symbol will be sent next, but also what the *probabilities* of the various symbols are.

For our black-and-white image example, only one probability is unknown: $p_1 = P(\text{Black})$. (Since $P(\text{White}) = 1 - p_1$.)

We start by selecting a *prior distribution* for p_1, that expresses what we know about p_1 *before* we've seen the image we're encoding.

The Posterior Distribution

Suppose we've seen n_B black pixels and n_W white pixels so far. What is the probability that the next pixel is black?

To find this, we first need to find the *posterior distribution* for the unknown probability, p_1.

This is found using Bayes' Rule:

$$P(p_1 \,|\, \text{pixels observed}) = \frac{P(\text{pixels observed} \,|\, p_1) P(p_1)}{\int_0^1 P(\text{pixels observed} \,|\, p_1) P(p_1) dp_1}$$

$P(p_1)$ is our prior probability density for p_1.

$P(\text{pixels observed} \,|\, p_1)$ is called the *likelihood*. It captures what we've learned about p_1 from the data. For this example:

$$P(\text{pixels observed} \,|\, p_1) \;=\; p_1^{n_B} (1-p_1)^{n_W}$$

The Predictive Distribution

To make a prediction, we find the *average* probability of a symbol with respect to its posterior distribution.

Suppose we've seen n_B black pixels and n_W white pixels, and suppose we've choosen a *uniform* prior for p_1, for which $P(p_1) = 1$.

We predict that the next pixel has the following probability of being black:

$$P(\text{Black}) \;=\; \int_0^1 p_1 \frac{p_1^{n_B} (1-p_1)^{n_W}}{\int_0^1 p_1^{n_B} (1-p_1)^{n_W} dp_1} dp_1$$

A useful fact:

$$\int_0^1 p^a (1-p)^b dp \;=\; a!\,b!/(a+b+1)!$$

Using this, we find that

$$\begin{aligned} P(\text{Black}) &= \frac{(n_B+n_W+1)!}{n_B!\, n_W!} \frac{(n_B+1)!\, n_W!}{(n_B+n_W+2)!} \\ &= (n_B+1)/(n_B+n_W+2) \end{aligned}$$