Codes for the positive integers

There are many situations in which the alphabet is a set of positive numbers *e.g.* $\{1, 2, 3, \ldots, \}$. For example, we might have an array of items and we wish to encode the index of one of these items; or we might have a linked list of items and we wish to encode the position of an element in the list. Or, we may wish to encode an image or audio file and we wish to encode the intensities.

In discussing optimal prefix codes, we proved a number of results in which the symbols were ordered by decreasing (more specifically, non-increasing) probability. This ordering was, in part, a convenience to help our proofs. But it is more than that. There are many types of data in which the symbols are positive integers, and such that the probability of a lower number is greater than or equal to the probability of a higher number.

$$p(1) \geq p(2) \geq p(3) \cdots \geq \ldots$$

When we are in a situation such as this, we should choose a code whose codelengths respect the probability ordering, i.e.

$$\lambda_1 \leq \lambda_2 \leq \lambda_3 \leq \ldots$$

We saw that this was a necessary condition for an optimal prefix code.

In this lecture and the next one, we will look at methods for coding positive integers. We will *not* place a bound N on the maximum size of the integer that we will consider.

Unary code

Our first example of a code for the positive integers is the *unary code.* The unary code for i is $i - 1$ 0's, followed by a 1. Note, $\lambda_i = i$.

i	$C(i)$	λ_i
1	1	1
2	01	2
3	001	3
4	0001	4
5	00001	5
6	000001	6
$\vdots$		

A few observations can be made immediately. First, if $p(i)$ were such that

$$\lambda_i = \log \frac{1}{p(i)}$$

that is, if

$$p(i) = 2^{-i}$$

then the average code length would be equal to the entropy.[1] Typically $p(i)$ does not have this form, however.

[1] In making this statement, I am ignoring the fact that the set of positive integers is infinitely large, whereas previously I assumed the alphabet is of size N which is finite.

Second, it would be impossible to have an optimal prefix code whose codeword lengths grow faster than $\lambda(i) = i$, since growing faster than this would require that some (non-leaf) nodes don't have siblings.

Bernoulli trials and run length coding

Suppose we are encoding the outcomes of an experiment with repeated, *independent* trials. On each trial, suppose we have a failure or a success. We represent failures with a 0 and successes with a 1. The probability of a failure is p_0 and the probability of a success is thus $1 - p_0$. These are called *Bernoulli trials.*

(Bernouilli trials can be used in very general situations. For example, you roll a pair of dice and you say that a success is the event that the sum on the two dice is greater than or equal to some number, say 9.)

Given a sequence of Bernouilli trials, we can define a new sequence of events, by parsing the sequence of Bernouilli trials into "runs" of length i, defined as $i - 1$ failures (0's) followed by a success (1). Let's not worry for now about the case that the sequence terminates with a failure.

For example

$$000001011000010001$$

would be parsed

$$000001, 01, 1, 00001, 0001$$

and the run lengths are $i = 6, 2, 1, 5, 4$.

The unary code $C(i)$ serves trivially as a *run length code.* $C(i)$ is the codeword for $i-1$ failures followed by a success, i.e. a run of $i - 1$ failures. Later in the lecture we will consider a better way to encode run lengths.

Before we do, let's calculate the *average run length.* Because we are assuming independent Bernouilli trials, the event of $i - 1$ failures followed by a success has probability:

$$p(i) = p_0^{i-1}(1 - p_0),$$

which is called the *geometric distribution.* Then,

$$\overline{\lambda} = \sum_{i=1}^{\infty} i\ p_0^{i-1}(1 - p_0) = (1 - p_0) \sum_{i=1}^{\infty} i\ p_0^{i-1}$$

We recognize each term inside the sum:

$$i p_0^{i-1} = \frac{d}{dp_0} p_0^i\ .$$

But

$$\sum_{i=1}^{\infty} \frac{dp_0^i}{dp_0} = \frac{d}{dp_0} \sum_{i=0}^{\infty} p_0^i = \frac{d}{dp_0}(\frac{1}{1 - p_0}) = \frac{1}{(1 - p_0)^2}$$

Thus,

$$\overline{\lambda} = \frac{1 - p_0}{(1 - p_0)^2} = \frac{1}{1 - p_0}$$

When p_0 is close to 0, the average run length i is slightly larger than 1. Whe p_0 is close to 1, the average run length is a very large number. This makes intuitive sense. When p_0 is close to 0, most of the events are successes, and so most of the symbols are 1, and so the run length tends to be close to 1. When p_0 is close to 1, most of the events are failures, and so most of the symbols are 0, and so the run length tends to be large.

Now that we understand how the average run length depends on p_0, we can ask what would be a better way to encode the run lengths.

Our target is to choose a code with codeword lengths $\lambda_i \approx \log \frac{1}{p(i)}$ since this would achieve an average code length near the entropy. Thus, we seek a code for which

$$\lambda_i \approx \log \frac{1}{p(i)} = \log \frac{1}{p_0^{i-1}(1-p_0)}, \quad \text{for all } i \ .$$

Expanding, we get:

$$\lambda_i \ \approx \ i \ \log \frac{1}{p_0} + \log \frac{p_0}{1-p_0}$$

Thus, we would like λ_i to be a linear function of i, where the slope depends on p_0.

Cases of interest:

- If $p_0 = \frac{1}{2}$, we have $\lambda_i = i$ which is the unary code. In this case, the unary code is the optimal prefix code.
- When $p_0 > \frac{1}{2}$, $\log \frac{1}{p_0} < 1$ and so λ_i grows linearly with i but a slope less than 1. We will see an example of a code $C(i)$ that achieves this (the Golomb code) next.
- When $p_0 < \frac{1}{2}$ and so λ_i grows linearly with i but a slope greater than 1. A code such that λ_i grows faster than the unary code cannot be optimal, however, since it would require that some (non-root) nodes don't have siblings.

The last case is somewhat strange. When p_0 is small (much less than $\frac{1}{2}$), the average run length (and hence average code length for unary code) is close to 1. The means that the sequence of run length events will have very low entropy. Most runs are of length 1, a minority are of length 2, and only a very small minority are of length more than 2. Thus, for very small p_0, the entropy on the run lengths will be much less than 1 bit. As a result, the average run length (which is greater than 1 bit) will be almost 1 bit greater than the entropy. Thus, the unary code for run length will have an average codelength that is about 1 bit greater than the entropy. This is not very impressive considering the entropy in this case is close to 0 bits.

Note that in the third case, it would be better to compress the sequence by encoding runs of successes followed by a failure, rather than failures followed by a success.

Golomb code

Consider a small number b that is a power of 2, for example $b = 1, 2, 4, 8, \ldots,$. Suppose we group the positive integers $i \geq 1$ into groups of size b. For example, if $b = 4$, the groups are $\{1, 2, 3, 4\}$, $\{5, 6, 7, 8\}$, etc. To code an integer $i \geq 1$, use a unary code to specify the group number to which i belongs, and then append a fixed length code ($\log b$ bits) that specifies one of the b numbers within that group. The group number is $\lceil \frac{i}{b} \rceil$. The position within the group is "$(i-1) \bmod b$". This is called a Golomb[2] code. Here is how the code starts out for $b = 4$:

i	$C(i)$	λ_i
1	1 00	3
2	1 01	3
3	1 10	3
4	1 11	3
5	01 00	4
6	01 01	4
7	01 10	4
8	01 11	4
9	001 00	5
10	001 01	5
$\vdots$		

Observe that

$$\lambda_i = \lceil \frac{i}{b} \rceil + \log b$$

For the average code length of the Golomb code to be close to the entropy, it is sufficient that $\lambda_i \approx \log \frac{1}{p(i)}$, or

$$p(i) \approx 2^{-\lambda_i} = 2^{-\lceil i/b \rceil}/b$$

Note that the Golomb code is identical to the unary code for $b = 1$. For $b > 1$, the probability function falls off more slowly than for a unary code ($b = 1$), but the falloff is still exponential.

In class, I drew a binary tree representations of Golomb code for $b = 4$.

[2]S.W. Golomb. "Run-length encodings". *IEEE Transactions on Information Theory*, IT–12(3):399–401, July 1966.