Basics of Information Theory

David S. Touretzky [*]

24 November 2004

Preliminaries

Although information is sometimes measured in characters, as when describing the length of an email message, or in digits (as in the length of a phone number), the convention in information theory is to measure information in *bits*. A "bit" (the term is a contraction of binary digit) is either a zero or a one. Because there are 8 possible configurations of three bits (000, 001, 010, 011, 100, 101, 110, and 111), we can use three bits to encode any integer from 1 to 8. So when we refer to a "3-bit number", what we mean is an integer in the range 1 through 8. All logarithms used in this paper will be to the base two, so $\log 8$ is 3. Similarly, $\log 1000$ is slightly less than 10, and $\log 1,000,000$ is slightly less than 20.

Efficient Encodings

Suppose you flip a coin one million times and write down the sequence of results. If you want to communicate this sequence to another person, how many bits will it take? If it's a fair coin, the two possible outcomes, heads and tails, occur with equal probability. Therefore each flip requires 1 bit of information to transmit. To send the entire sequence will require one million bits.

But suppose the coin is biased so that heads occur only 1/4 of the time, and tails occur 3/4. Then the entire sequence can be sent in 811,300 bits, on average. (The formula for computing this will be explained below.) This would seem to imply that each flip of the coin requires just 0.8113 bits to transmit. How can you transmit a coin flip in less than one bit, when the only language available is that of zeros and ones? Obviously, you can't. But if the goal is to transmit an entire sequence of flips, and the distribution is biased in some way, then you can use your knowledge of the distribution to select a more efficient code. Another way to look at it is: a sequence of biased coin flips contains "less information" than a sequence of unbiased flips, so it should take fewer bits to transmit.

[*]Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, *dst@cs.cmu.edu*

Let's look at an example. Suppose the coin is very heavily biased, so that the probability of getting heads is only 1/1000, and tails is 999/1000. In a million tosses of this coin we would expect to see only about 1,000 heads. Rather than transmitting the results of each toss, we could just transmit the numbers of the tosses that came up heads; the rest of the tosses can be assumed to be tails. Each toss has a position in the sequence: a number between 1 and 1,000,000. A number in that range can be encoded using just 20 bits. So, if we transmit 1,000 20-bit numbers, we will have transmitted all the information content of the original one million toss sequence, using only around 20,000 bits. (Some sequences will contain more than 1,000 heads, and some will contain fewer, so to be perfectly correct we should say that we expect to need 20,000 bits on average to transmit a sequence this way.)

We can do even better. Encoding the absolute positions of the heads in the sequence takes 20 bits per head, but this allows us to transmit the heads in any order. If we agree to transmit the heads systematically, by going through the sequence from beginning to end, then instead of encoding their absolute positions we can just encode the distance to the next head, which takes fewer bits. For example, if the first four heads occurred at positions 502, 1609, 2454, and 2607, then their encoding as "distance to the next head" would be 502, 1107, 845, 153. On average, the distance between two heads will be around 1,000 flips; only rarely will the distance exceed 4,000 flips. Numbers in the range 1 to 4,000 can be encoded in 12 bits. (We can use a special trick to handle the rare cases where heads are more than 4,000 flips apart, but we won't go into the details here.) So, using this more sophisticated encoding convention, a sequence of one million coin tosses containing about 1,000 heads can be transmitted in just 12,000 bits, on average. Thus a single coin toss takes just 0.012 bits to transmit. Again, this claim only makes sense because we're actually transmitting a whole sequence of tosses.

What if we invented an even cleverer encoding? What is the limit on how efficient any encoding can be? The limit works out to about 0.0114 bits per flip, so we're already very close to the optimal encoding.

The information content of a sequence is defined as the number of bits required to transmit that sequence using an optimal encoding. We are always free to use a less efficient coding, which will require more bits, but that does not increase the amount of information transmitted.

Variable Length Codes

The preceding examples were based on fixed-length codes, such as 12-bit numbers encoding values between 1 and 4,000. We can often do better by adopting a variable length code. Here is an example. Suppose that instead of flipping a coin we are throwing an eight-sided die. Label the sides A-H. To encode a number between 1 and 8 (or between 0 and 7, if you're a computer scientist) takes 3 bits, so a thousand throws of a fair die will take 3,000 bits to transmit. Now suppose the die is not fair, but biased in a specific way: the chances of throwing an A are 1/2, the chances of throwing a B are 1/4, C is 1/8, D is 1/16, E is 1/32, F is 1/64, and G and H are each 1/128. Let us verify that the sum of these probabilities is 1, as it must be for any proper probability distribution:

$$\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\frac{1}{16}+\frac{1}{32}+\frac{1}{64}+\frac{1}{128}+\frac{1}{128}=1$$

Now let's consider an encoding ideally suited to this probability distribution. If we throw the die and get an A, we will transmit a single 0. If we throw a B we will transmit a 1 followed by a 0, which we'll write 10. If we throw a C the code will be 11 followed by 0, or 110. Similarly we'll use 1110 for D, 11110 for E, 111110 for F, 1111110 for G, and 1111111 for H. Notice that the code for A is very concise, requiring a single bit to transmit. The codes for G and H require 7 bits each, which is way more than the 3 bits needed to transmit one throw if the die were fair. But Gs and Hs occur with low probability, so we will rarely need to use that many bits to transmit a single throw. On average we will need fewer than 3 bits. We can easily calculate the average number of bits required to transmit a throw: it's the sum of the number of bits required to transmit each of the eight possible outcomes, weighted by the probability of that outcome:

$$\frac{1}{2}+\frac{2}{4}+\frac{3}{8}+\frac{4}{16}+\frac{5}{32}+\frac{6}{64}+\frac{7}{128}+\frac{7}{128}\approx 1.984$$

So 1,000 throws of the die can be transmitted in just 1,984 bits rather than 3,000. This simple variable length code is the optimal encoding for the probability distribution above. In general, though, probability distributions are not so cleanly structured, and optimal encodings are a lot more complicated.

Exercise: suppose you are given a five-sided biased die that has a probability of 1/8 of coming up A, 1/8 for B, and 1/4 for each of C, D, and E. Design an optimal code for transmitting throws of this die. (Answer at end.)

Measuring Information Content

In the preceding example we used a die with eight faces. Since eight is a power of two, the optimal code for a uniform probability distribution is easy to caclulate: $\log 8 = 3$ bits. For the variable length code, we wrote out the specific bit pattern to be transmitted for each face A-H, and were thus able to directly count the number of bits required.

Information theory provides us with a formula for determining the number of bits required in an optimal code even when we don't know the code. Let's first consider uniform probability distributions where the number of possible outcomes is not a power of two. Suppose we had a conventional die with six faces. The number of bits required to transmit one throw of a fair six-sided die is: $\log 6 = 2.58$. Once again, we can't really transmit a single throw in less than 3 bits, but a sequence of such throws can be transmitted using 2.58 bits on average. The optimal code in this case is complicated, but here's an approach that's fairly simple and yet does better than 3 bits/throw. Instead of treating throws individually, consider them three at a time. The number of possible three-throw sequences is $6^3 = 216$. Using 8 bits we can encode a number between 0 and 255, so a three-throw sequence can be encoded in 8 bits with a little to spare; this is better than the 9 bits we'd need if we encoded each of the three throws separately.

In probability terms, each possible value of the six-sided die occurs with equal probability $P = 1/6$. Information theory tells us that the minimum number of bits required to encode a throw is $-\log P = 2.58$. If you look back at the eight-sided die example, you'll see that in the optimal code that was described, every message had a length exactly equal to $-\log P$ bits. Now let's look at how to apply the formula to biased (non-uniform) probability distributions. Let the variable x range over the values to be encoded, and let $P(x)$ denote the probability of that value occurring. The expected number of bits required to encode one value is the weighted average of the number of bits required to encode each possible value, where the weight is the probability of that value:

$$\sum_x P(x) \times (-\log P(x))$$

Now we can revisit the case of the biased coin. Here the variable ranges over two outcomes: heads and tails. If heads occur only 1/4 of the time and tails 3/4 of the time, then the number of bits required to transmit the outcome of one coin toss is:

$$\frac{1}{4} \times \left(-\log \frac{1}{4}\right) + \frac{3}{4} \times \left(-\log \frac{3}{4}\right) = 0.8113 \text{ bits}$$

A fair coin is said to produce more "information" because it takes an entire bit to transmit the result of the toss:

$$\frac{1}{2} \times \left(-\log \frac{1}{2}\right) + \frac{1}{2} \times \left(-\log \frac{1}{2}\right) = 1 \text{ bit}$$

The Intuition Behind the $-P \log P$ Formula

The key to gaining an intuitive understanding of the $-P \log P$ formula for calculating information content is to see the duality between the number of messages to be encoded and their probabilities. If we want to encode any of eight possible messages, we need 3 bits, because $\log 8 = 3$. We are implicitly assuming that the messages are drawn from a uniform distribution.

The alternate way to express this is: the probability of a particular message occurring is 1/8, and $-\log(1/8) = 3$, so we need 3 bits to transmit any of these messages. Algebraically, $\log n = -\log(1/n)$, so the two approaches are equivalent when the probability distribution is uniform. The advantage of using the probability approach is that when the distribution is non-uniform, and we can't simply count the number of messages, the information content can still be expressed in terms of probabilities.

Sometimes we write about rare events as carrying a high number of bits of information. For example, in the case where a coin comes up heads only once in every 1,000 tosses, the signal that a heads has occurred is said to carry 10 bits of information. How is that possible, since the result of any particular coin toss takes 1 bit to describe? Transmitting *when* a rare event occurs, if it happens only about once in a thousand trials, will take 10 bits. Using our message counting approach, if a value occurs only 1/1000 of the time in a uniform distribution, there will be 999 other possible values, all equally likely, so transmitting any one value would indeed take 10 bits.

With a coin there are only two possible values. What information theory says we can do is consider each value separately. If a particular value occurs with probability P, we assume that it is drawn from a uniformly distributed set of values when calculating its information content. The size of this set would be $1/P$ elements. Thus, the number of bits required to encode one value from this hypothetical set is $-\log P$. Since the actual distribution we're trying to encode is not uniform, we take the weighted average of the estimated information content of each value (heads or tails, in the case of a coin), weighted by the probability P of that value occurring. Information theory tells us that an optimal encoding can do no better than this. Thus, with the heavily biased coin we have the following:

$$P(\text{heads}) = 1/1000, \text{ so heads takes } \ -\log(1/1000) \approx 9.96578 \text{ bits to encode}$$
$$P(\text{tails}) = 999/1000, \text{ so tails takes } \ -\log(999/1000) \approx 0.00144 \text{ bits to encode}$$
$$\text{Avg. bits required} = \sum_x -P(x)\log P(x) =$$
$$(1/1000) \times 9.96578 + (999/1000) \times 0.00144 = 0.01141 \ \text{ bits per coin toss}$$

Answer to Exercise

The optimal code is:

Message	Code
A	000
B	001
C	01
D	10
E	11