Entropy

Consider the following problem. You have a alphabet of K different symbols and a collection C of strings of these symbols where each string has length N. To each string $S \in C$ you want to assign a unique bit string, called the *encoding* of S, and you want to do this efficiently, so that the average length of the encodings of the strings in C is as small as possible, in terms of number of bits.

First, an obvious but fundamental remark: The collection of M bit strings with the minimal average length is obviously the one with the 2 bit strings of length 2, the 4 of length 2, the 8 of length 3 etc. Let $K = \lfloor \log_2(M) \rfloor$. Then this collection has 2^I bit strings of length I for $I = 1 \ldots K$; it may also have some length $K+1$, but we will ignore these. The average length of the bit strings in the collections is thus at least $\sum_{I=1}^{K} I2^I/M = ((K-1)2^{K+1}+2)/M > K-1$. Therefore, if C is any collection of bit strings, the average length of a string in C is at least $\lfloor \log(|C|) \rfloor - 1$. (Throughout these notes, logarithms are to base 2, unless otherwise specified.)

If C is the set of all the bit strings of length N, then there is an obvious encoding scheme that nearly attains this; namely associate each symbol with a bit string of $\lceil \log(K) \rceil$ bits, and then string these together to represent the symbol string. For instance, suppose the alphabet is the four letters A,B,C,D. Then we can encode A→00, B→01, C→10, and D→11. A string like ABADCB then becomes 000100111001. We know that we can translate back from the bit string to the symbols by breaking the bit string into pairs of bits, and then translating each pair into a symbol. A string of N symbols requires here 2N bits; in general, if the alphabet has K symbols, a string of N symbols will require $N \cdot \lceil \log(K) \rceil$ bits, or $\lceil \log(K) \rceil$ bits per symbol.

However, if the strings in the collection C have a particular, uneven, distribution of the alphabetical symbols, then it may be possible to do better than that. Consider for example a collection C where every string S has the following feature: 3/4 of the letters in S are "A", 7/8 are "B", 1/16 are "C", and 1/16 are "D". Now consider the encoding A→0, B→10, C→110, D→111. Thus ABADCB gets encoded as 010011111010. Now, a string with N symbols will have consist of

(3/4)N occurrences of A with	(3/4)N bits
(1/8)N occurrences of B with	2·(1/8)N bits
(1/16)N occurrences of C with	3·(1/16)N bits
(1/16)N occurrences of D with	3·(1/16)N bits
Total	(11/8)N bits

Thus, this encoding requires 11/8 bits per symbol, much shorter than the 2 bits per symbol required by the naive encoding.

Making sure that this encoding gives a different code for every string is not as easy in this case as in the first case, since the codes for the symbols have different lengths. However, it can be done using the fact that this is a *prefix-free* encoding of the symbols; that is, it is never the case that the code for one symbol is a prefix of a code for another. That being the case, one can read off the symbols unambiguously by working from left to right in the string. For example, starting with the bit string 010011111010, the first "0" must be the code for A, since that is the only symbol whose code begins with 0. The next "10" must be the code for B since "1" is not a code for any symbol, and there is no other code that begins with "10". The next "0" must be the code for A again. The next "111" must be the code for D, since "1" and "11" are not codes, and there is no other code starting "111". And so on. Prefix-free codes are one type of *unambiguous* codes.

In general, suppose that C contains only strings with the following distribution: The Ith alphabetic symbol occurs with frequency p_I. Then there exists a prefix-free encoding where each symbol has a code of length at most $\lceil \log(1/p_k) \rceil$. Proof: The Huffman coding algorithm generates such a code. Suppose we have such a code; how long is the encoding of a string in C? Well, there are $N \cdot p_I$

occurrences of the Ith symbol, and each such occurrence requires at most $\lceil \log(1/p_I) \rceil$, so the total number of bits is at most $N \cdot \sum_I p_I \lceil log(1/(p_i)) \rceil$.

Moreover, if the collection C contains all strings with this distribution of symbols, then one can't do very much better than that. Proof: By simple combinatorics, the number of strings with the specified distribution of symbols,

$$|C| = \frac{N!}{(p_1 N)! \cdot (p_2 N)! \cdot \ldots \cdot (p_k N)!}$$

Since $\log N!$ is approximately $\log((N/e)^N) = N \log(N) - N \log(e)$, we get

$$\log(|C|) = (N \log(N) - N \log(e) - \sum_I (p_I N) \log(p_I N) - (p_I N \log e)$$

However, since $\log(p_I N) = \log(p_I) + \log(N)$ and $\sum_I p_I = 1$, the $N \log N$ and $N \log e$ terms cancel out, leaving the sum

$$\log(|C|) = -N \sum_I p_I \log(p_I)$$

that is, $-\sum_I p_I \log(p_I)$ bits per symbol. (Since the p_I's are all less than 1, $\log(p_I)$ is negative, so this sum is positive.) Since $\log(1/x) = -\log(x)$ this is very nearly the same as the results above for the Huffman coding, except for the rounding functions.

The quantity $-\sum_I p_I \log(p_I)$ is known as the *entropy* of the distribution $p_1 \ldots p_K$; it is often denoted $H(p_1 \ldots p_K)$. (The physicists use the natural log in this definition.) The following are fundamental theorem of information theory, due to Claude Shannon:

> Theorem 1: Let $p_1 \ldots p_k$ be a distribution over K symbols. If C is the collection of all strings of length N with distribution $p_1 \ldots p_k$ then any unambiguous coding scheme for P requires at least $H(p_1 \ldots p_k) - 2$ bits per string on average.
>
> Theorem 2: Let $p_1 \ldots p_k$ be a distribution over K symbols, and let $N > 0$. Then there exists in unambiguous coding scheme such that for any $M > N$, if C is the set of all strings of length M with distribution $p_1 \ldots p_k$ the encoding of C no more than $M \cdot (H(p_1 \ldots p_k) + O(1/N))$ bits per string on average.

These bounds are very close to the lower bound we calculated above and substantially better than the upper bound we calculated. To achieve this, you need to use codes that encode more than one symbol at a time. This is most clearly seen when the entropy is less than 1. Suppose that you have an alphabet with 2 symbols A and B, and the collection C contains strings in which 127/128 of the symbols are A and 1/128 of the symbols are B. Then the entropy is equal to (1/128)log(1/128) + (127/128)log(127/128) = 0.066. How can a coding scheme have less than one bit per symbol?

The answer is that the coding scheme encodes more than one symbol at a time. Here is an instance. Divide the string S into chunks of 128 consecutive symbols. These can be divided into the following categories:

1. Chunks with no occurrences of B.
2. Chunks with exactly one occurrence of B.
3. Chunks with two occurrences of B.
4. Chunks with more than two occurrences of B.

One can show that, for large enough N, for almost all strings S in C, the fraction of chunks in category 1 is 1/e = 0.368; the fraction in category 2 is also 1/e; the fraction in category 3 is 1/2e = 0.184; and the fraction in category 4 is (1-5/2e) = 0.080. Let us use the following encoding:

- A string in category 1 is encoded as 00 (2 bits).
- A string in category 2 with B in the Ith place is encoded as 01 followed by the 7 bit binary code for I. (9 bits)
- A string in category 3 with B in the Ith and Jth places is encoded as 10 followed by the 7 bit binary codes for I and J. (16 bits)
- A string in category 4 is encoded as 11 followed by 128 with 0 for A and 1 for B. (130 bits)

Then the average number of bits for each block of 128 symbols is 0.368*2 + 0.368 * 9 + 0.184 * 16 + 0.08 * 130 = 17.39 bits per block or 0.13 bits per symbol. So this isn't optimal, but it's getting there.

At this point we've done all the calculation we will need. Now we make a couple of conceptual leaps. First, instead of talking about strings in a collection, we generally want to talk about strings that are generated by a random process. The calculations are pretty much all the same, though the argumentation is more subtle.

Theorem 3: Suppose that one generates strings of symbols by a random process that, at each step, outputs symbol I with probability p_I. Then there is no encoding scheme for the strings produced by this process for which the expected bit length of a string of N symbols is less than $N \cdot H(p_1 \dots p_k)$.

Theorem 4: Suppose that one generates strings of symbols by a process — random, partially random, or deterministic – such that, in the long run, the frequency of symbol I is p_I. Then for any $\epsilon > 0$ there is an encoding scheme for strings produced by the process such that, for sufficiently large N, the expected bit length of a string of length N is less than $N \cdot (H(p_1 \dots p_k) + \epsilon)$

This leads to the idea of entropy as a measure of *ignorance.* Suppose initially all you know about a string S is that its length is N and it was produced by a process of the kind in theorems 3 and 4. Then you find out exactly what S is. This could have been specified for you in an encoding of S in $N \cdot H(p_1 \dots p_k)$ bits; thus the information you have gained is $N \cdot H(p_1 \dots p_k)$ bits; thus your ignorance in the initial state was $N \cdot H(p_1 \dots p_k)$ bits. We can also say that finding out just one symbol generated by the process would supply $H(p_1 \dots p_k)$ bits on average; thus this is your ignorance about the next symbol to be generated. Thus $H(p_1 \dots p_k)$ is the measure of your ignorance about the next symbol to be produced, if you know that the process producing symbols follows the distribution $p_1 \dots p_k$.

Now, suppose that you have a large space of instances Ω, each of which has a classification attribute $X.C$, and you have an imperfect classifier ϕ, which predicts $X.C$ from the predictive attributed. Let $v_1 \dots v_k$ be the possible values of $X.C$. You have generated a large table T of labelled instances and you have a new instance Q for which you know the predictive attribute. Assume that T and Q are both random samples of Ω and that T is a representative sample of Ω. Then before you run the classifier on I, your ignorance of the value of $Q.C$ is
$H(Freq_{X \in T}(X.C = v_1) ... Freq_{X \in T}(X.C = V_k))$ = ENTROPY(C,T)
in the notation of the ID3 algorithm.

Then you run the classifier on Q and get the value $\phi(Q)$.
Let T_I be the subtable $\{X \in T | \phi(X) = v_I\}$. If $\phi(Q) = I$ that Q is in the table T_I so your ignorance is now ENTROPY(C, T_I). However, not all subtables are equally likely; in fact the likelihood that $\phi(Q) = I$ is just $|T_I|/|T|$. Therefore, if you ask before running the classifier what

will be your expected ignorance after the classifier, it is just $\sum_I |T_I|/|T|$ ENTROPY(C, T_I) = AVG_ENTROPY(ϕ, C, T). The average information about the value of $Q.C$ gained by running the classifier is ENTROPY(C, T) − AVG_ENTROPY(ϕ, C, T).