Chapter 5

Information theory

In this chapter we develop some tools to describe language data for our models and to distinguish among different language models. These tools involve *information theory*, and so we must first consider the basic nature of this theory.

5.1 What is it?

Information theory (Shannon 1951) is concerned with how much information can be transmitted in how much time over thus-and-such channel given some noise condition. What is relevant for our purposes is that information theory provides a way of quantifying how much information is in some message or language and how well some model captures that information.

5.2 Entropy

Entropy is a measure of the amount of information in some variable. Consider the following hypothetical situation. You must transmit to someone else a series of numbers that range from zero to three: $\{0, 1, 2, 3\}$. Moreover, each number is equally likely, and for generality's sake, you must represent it using binary numbers, e.g. $0 = 00$, $1 = 01$, $2 = 10$, and $3 = 11$. Note that you must use two digits to represent 0 and 1 as well, since in a string of digits you must be able to distinguish where one number ends and another number begins. For example, if we encoded 1 and 0 as is, a string like 111 is three ways ambiguous: $1 - 1 - 1$, $1 - 3$, or $3 - 1$.

probabilities	encoding	bits
.25, .25, .25, .25	00, 01, 10, 11	2 bits
.25, .25, .25, .25	1, 01, 000, 001	2.25 bits
.5, .2, .2, .1	00, 01, 10, 11	2 bits
.5, .2, .2, .1	1, 01, 000, 001	1.8 bits

Figure 5.1: Entropy for various probabilities

Given that there are only four choices, and that you are encoding your digits with binary numbers, it follows that your messages will contain two digits for each number they contain. In the parlance of information theory, you need two *bits* to communicate each unit.

If we apply this to a text or language L, then what we would want to say is that the information content of some L is a measure of the number of words that occur (assuming all words are equally likely). Crudely, if a language contains 1000 words, then its entropy is 10 (since $2^{10} = 1000$).

If, however, the different words are not equally likely, then a more efficient binary coding is possible. For example, to represent the four options in the system above, we could instead use: 000, 001, 01, and 1. Here the number of bits used varies from one to three. This is not a problem in a message composed of more than one of these because of the encoding used. That is, this system will not lead to an ambiguities where the representation of one number ends and another number begins. For example, 001011 can only be $001 - 01 - 1$.

The logic behind using an encoding like this when the message units are not equally likely is to use the longer encodings for the less frequent message units and the shorter encodings for the more frequent message units. The average number of bits per unit will then be affected by the frequency of the different units. Thus, the entropy of L can be computed by multiplying the frequency of each message unit by the number of bits required to encode it and summing the results.

The chart in figure 5.1 compares different possible encodings and different possible frequency ranges. The simple two-bit encoding works fine when each outcome or number is equally probable. However, when the outcomes have different probabilities, then the second encoding above is more efficient, providing an average entropy of 1.8 bits, rather than 2 bits.

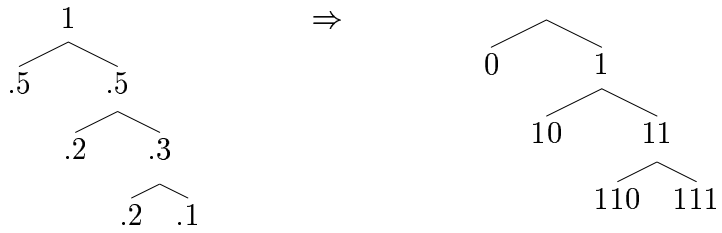

Figure 5.2: Asymmetric Huffman tree

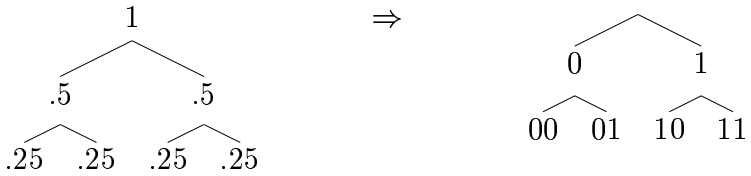

Figure 5.3: Symmetric Huffman tree

5.3 Huffman encoding

We can calculate the most efficient code for some distribution of outcomes using *Huffman encoding* (Hopcroft & Ullman 1979). The algorithm is as follows. One makes a binary tree from the two lowest-frequency elements. Their parent node is assigned the sum of their probabilities. Keep doing this until all elements are in the tree. Assign 0 and 1 to each pair of nodes. Read the code off the tree. Let's consider how this works for the two cases we considered above. For $\{.5, .2, .2, .1\}$, we get the tree in figure 5.2, which results in the code: 0, 10, 110, 111. For $\{.25, .25, .25, .25\}$, we get the tree in figure 5.3, which results in the code: 00, 01, 10, 11.

5.4 Entropy formalized

The formula for entropy follows directly from what we've said above. One calculates the simplest base-two representation of a message and multiplies it by its overall probability. Then sum over all possible messages.

$$H(\mathrm{p}) = -\sum \mathrm{p}(x) \log \mathrm{p}(x) \tag{5.1}$$

One can also calculate the per-word entropy, given messages of some length n.

Text	Entropy	Perplexity
a b a b a b a b	1	2
a b c d e a b c d e	2.32	5
a b a b a a a a a a	0.72	1.65
a a a a a b b b b b	1	2

Figure 5.4: Entropy/Perplexity of four texts

$$H_{\text{rate}} = -\frac{1}{n}\sum \text{p}(x_{1n}) \log \text{p}(x_{1n}) \tag{5.2}$$

The per-word entropy captures some of the relationships that might occur in larger sequences.

There is, in fact, the *Shannon-McMillan-Breiman theorem* that allows us to estimate the entropy of a language from a big enough sample (Algoet & Cover 1988):

$$H(L) = \lim_{n\to\infty} \frac{1}{n} \log \text{p}(w_1 w_2 \ldots w_n) \tag{5.3}$$

Entropy is also occasionally expressed as *perplexity*:

$$\text{perplexity}(x_{1n}, m) = 2^{H(x_{1n}, m)} \tag{5.4}$$

Perplexity is defined as 2 raised to the power indicated by the entropy.

We can now measure the entropy of languages and texts.[1] What this tells us is a measure of the size of the vocabulary used in a text measured against the relative probabilities of the items in the text. If there are a lot of words in a text and they are relatively equivalent in terms of how often they occur, entropy will be high. If, on the other hand, the number of words is relatively low, or there is a large asymmetry in terms of the probabilities of those words, then entropy will be low.

Consider the entropy of the texts given in figure 5.4. Here, single letters represent words. Each text thus contains exactly ten words. In the first text, there are only two words, each of which occurs five times. This thus produces an entropy of 1. On the other hand, the second text contains *five* words each of which occurs two times. This results in a higher entropy of 2.32. The

[1] This can be done with the `entropy.pl` program.

Text	Entropy	Perplexity
a b a b a b a b a b	0.128	1.093
a a a a a b b b b b	0.489	1.404

Figure 5.5: Entropy/Perplexity computed with bigrams

third text contains only two words, but they have a remarkably different distribution. This thus results in a lower entropy than the first text: 0.72. Finally, the fourth text simply reorders the words of the first text; notice that their entropy is the same.

We can also calculate the entropy of a text using bigrams.[2] In this case, the order of words will certainly matter. The chart in figure 5.5 compares the first and fourth cases from figure 5.4 with respect to entropy computed with bigrams.

5.5 Cross-entropy

Entropy can be used to assess the success of a model. The basic idea is that we choose the entropy for all messages predicted by a model—multiply them by the observed probabilities—and sum.

$$H(\mathrm{p}, \mathrm{q}) = -\sum \mathrm{p}(x) \log \mathrm{q}(x) \tag{5.5}$$

This number will always be greater than or equal to the actual entropy of the system.

By the Shannon-McMillan-Breiman theorem, this can be simplified to:

$$H(\mathrm{p}, \mathrm{q}) = \lim_{n \to \infty} \frac{1}{n} \log \mathrm{q}(w_1 w_2 \ldots w_n) \tag{5.6}$$

Let's now consider how this can be applied to some hypothetical texts (figure 5.6). Here the same four languages as in figure 5.4 are used as training texts and then all used as testing texts. This results in sixteen possibilities when all are combined. The training texts are given in each of the rows and the testing texts are given in each of the columns.

Notice how there is *infinite* entropy when the probabilities computed from one text are tested against a text with new words. This is because the new

[2] This can be done with the `entropy2.pl` program.

Text	1	2	3	4
1 (a b a b a b a b a b)	1	∞	1	1
2 (a b c d e a b c d e)	2.32	2.32	2.32	2.32
3 (a b a b a a a a a a)	1.32	∞	0.72	1.32
4 (a a a a a b b b b b)	1	∞	1	1

Figure 5.6: Cross-entropy of four texts

words have zero probability and the logarithm of zero is infinity.[3] Notice too that in each case the cross-entropy that results when a model trained on a text is tested against the same text is the same or lower than the entropy that results when it is tested against a different text. There is a general theorem that has this effect: the cross-entropy revealed by some model is no smaller than the actual entropy of a text:

$$H(\mathrm{p}) \leq H(\mathrm{p}, \mathrm{q}) \tag{5.7}$$

This turns out to be quite useful. It means that cross-entropy can be used to *test* models.

[3] Check this!