Shannon's noiseless coding theorem

Prof. Peter Shor

While I talked about the binomial and multinomial distribution at the beginning of Wednesday's lecture, in the interest of speed I'm going to put the notes up without this, since I have these notes (modified from last year). I'll put up some short notes on the binomial distribution this weekend.

1 Introduction

Today's lecture is on Shannon's *noiseless coding theorem,* which in modern terminology is about data compression. One of the earliest instances of widespread use of data compression came with telegraph code books, which were in widespread use at the beginning of the 20th Century. At this time, telegrams were quite expensive; the cost of a transatlantic telegram was around $1 per word, which would be equivalent to something like $30 today. This led to the development of telegraph code books, some of which can be found in Google Books. These books gave a long list of words which encoded phrases. Some of the codewords in these books convey quite a bit of information; in the Fourth edition of the ABC Code, for example, "Mirmidon" means "Lord High Chancellor has resigned" and "saturation" means "Are recovering salvage, but should bad weather set in the hull will not hold out long."[1]

In 1948, Claude Shannon published a seminal paper which founded the field of information theory[2]. In this paper, among other things, he set data compression on a firm mathematical ground. How did he do this? Shannon consider an alphabet A of symbols, which we will call letters. To prove anything mathematically formal about data compression, you need a mathematical model of the data you are going to compress. Shannon considered a random *source* of letters, and asked how well on average could one compress n letters emitted from his source. What was his mathematical definition of a source? It's a probabilistic structure called a Markov chain. We won't go into the definition now, but a subclass of the sources he considered were sources where the probability that the source emits a given letter depends only on the k letters that immediately precede it, for some fixed value of k.

Why are these sources relevant to English? Suppose you are trying to compress English text. We might consider that we have some sample corpus of English text on hand (say, everything in the Library of Congress). Shannon considered a series of sources, each of which is a better approximation to English. The first-order source which emits a letter a with probability p_a which is proportional to its frequency in the text. The probability distribution of a sequence of n letters from this source is just n independent random variables where the letter a_j appears with some probability p_j. The second-order source is that where a letter is emitted with probability that depends only on the previous letter, and these probabilities are just the conditional probabilities that appear in the corpus (that is, the conditional probability of getting a 'u', given that the

[1] *The ABC Universal Commercial Telegraph Code, Specially Adapted for the Use of Financiers Merchants, Shipowners, Brokers, Agents, Etc.* by W. Clauson-Thue, American Code Publishing Co., Fourth Edition (1899)

[2] Available on-line at http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf

previous letter was a 'q', is derived by looking at the frequency of all letters that follow a 'q' in the corpus). In the third-order source, the probability of a letter depends only on the two previous letters, and so on. High-order sources would seem to give a pretty good approximation of English, and so it would seem that a compression method that works well on this class of sources would also work well on English text.

Now, suppose we look just at a first order source, where all the letters emitted by the source are independent. We will ask: how well can we compress n letters from this source? If we look at k'th order sources, where each letter only depends on the $k-1$ previous ones, the theorem works in much the same way conceptually, but the details get quite a bit more complicated, so we will only look at first order sources in this lecture. What we will show is both a lower bound on the the number of bits needed and also a compression scheme that essentially achieves this bound.

Suppose we look at a sequence of n letters from a source of independent, identical letters (we will call this an i.i.d. source). We will let the probability of letter a_i be p_i. The expected number of letter a_i is np_i. What does the distribution of the number of each letter look like? We saw this in the previous lecture. If

$$n_1 + n_2 + \ldots + n_{|A|} = n$$

then the probability that we see exactly n_i of letter a_i is given by the multinomial distribution:

$$\binom{n}{n_1, n_2, \ldots, n_{|A|}} p_1^{n_1} p_2^{n_2} \ldots p_{|A|}^{n_{|A|}}$$

We saw in the previous lecture that the expectation of n_i is $p_i n$, and that the probability is small that n_i is very far from $p_i n$. We will use this fact in this lecture. In fact, what we need is a somewhat stronger bound than Chebyshev's bound will give us. Recall that Chebyshev's bound said:

$$P\left(|n_i - np_i| \geq y\sqrt{p_i(1-p_i)}\sqrt{n}\right) \leq \frac{1}{y^2}$$

If we let $y \approx \sqrt{n}$, then Chebyshev's bound tells us that the probability that $|n_i - np_i| \geq \epsilon n$ is at most $O(1/n)$. The actual value is a lot smaller than this, but we won't treat it in this course. [Exercise: if you consider the fourth moment of the variable, you can get a better bound than Chebyshev's bound. Do this.]

First, let's try to show that there is a limit beyond which we cannot compress this source. For the multinomial distribution, if we fix an $\epsilon > 0$, then the bounds above show that for large enough n, we will have with high probability that

$$(p_i - \epsilon)n \leq n_i \leq (p_i + \epsilon)n$$

For the time being, we'll ignore ϵ. Now, let's assume that we know the exact number of each letter the source has sent out of the first n letters. We will show how well we can compress our source, given this knowledge. Since it's clear that extra knowledge can only help us compress the source, this gives a lower bound on how much compression we can do.

Suppose we know that we have n_i of letter a_i. We saw last time that the number of different ways of arranging this set of letters was

$$\binom{n}{n_1, n_2, \ldots, n_{|A|}.}$$

Each of these rearrangements will be equally likely.

Now, if we have some number M of equally likely messages that must be sent, then in order to send them we need to use $\log M$ bits on average. (We will see this in more detail next time.) So we need to send at least

$$\log_2 \binom{n}{n_1, n_2, \ldots, n_{|A|}}$$

bits. To approximate this, we need Stirling's formula

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n.$$

In fact, because we're taking the log of the answer, we can ignore the first term and just use

$$n! \approx \left(\frac{n}{e}\right)^n.$$

We won't explain in these notes why this is a good enough approximation. If you want to, work it out yourself.

Now, let's assume that $n_i = np_i$. Then we have

$$\begin{aligned} \binom{n}{n_1, n_2, \ldots, n_{|A|}} &= \frac{n!}{n_1! n_2! \ldots n_{|A|}!} \\ &\approx \frac{\left(\frac{n}{e}\right)^n}{\left(\frac{np_1}{e}\right)^{np_1} \left(\frac{np_2}{e}\right)^{np_2} \cdots \left(\frac{np_{|A|}}{e}\right)^{np_{|A|}}} \end{aligned}$$

Let's look at this last expression. In the numerator, we have e^n on the bottom. In the denominator, we have $e^{np_1} e^{np_2} \ldots e^{np_{|A|}}$. However,

$$e^{np_1} e^{np_2} \ldots e^{np_{|A|}} = e^{n \sum_i p_i} = e^n$$

because the p_i sum to 1, and these terms all cancel out. Similarly, the n terms all cancel out, and we get

$$\begin{aligned} \binom{n}{n_1, n_2, \ldots, n_{|A|}} &\approx {p_1}^{-np_1} {p_2}^{-np_2} \ldots {p_{|A|}}^{-np_{|A|}} \\ &= 2^{(-p_1 \log_2 p_1 - p_2 \log_2 p_2 - p_3 \log_2 p_3 \ldots - p_{|A|} \log_2 p_{|A|})n}. \end{aligned}$$

Taking the log, we see we cannot compress the source to fewer than

$$-n \sum_i p_i \log_2 p_i$$

bits. Of course, our assumption that $n_i = np_i$ is not exactly true, with high probability $n_i = n(p_i + \epsilon_i)$ for some very small ϵ_i, and the ϵ_i do not affect the lower bound much. This sum, $\sum_i -p_i \log_2 p_i$ is called the *entropy* of a probability distribution P, and is denoted $H(P)$ or $H(p_1, p_2, \ldots, p_{|A|})$. Summarizing, the first part of the theorem says that no matter how we compress we will need to send at least $H(p)n$ bits for a message of n symbols.

We now will show that one can do compression and send only slightly more than $H(P)n$ bits. For this upper bound, what we do is divide all the possible outputs of our source into two classes:

typical and atypical outputs. We define a *typical output* to be one where the numbers of each letter are approximately correct, so $|n_i - np_i| \leq \epsilon n$ for all i. Everything else is an *atypical output*. We now use the linearity of expectation to bound the length of the coded message:

$$\begin{aligned} E(\text{length}) &= E(\text{length}|\text{typical output})P(\text{typical}) \\ &= + E(\text{length}|\text{atypical output})P(\text{atypical}). \end{aligned}$$

Now, we need to analyze this expression. $P(\text{atypical}) \leq \frac{c}{n}$, so as long as we don't make the output in the atypical case more than length Cn, we can ignore the second term, as this one will be constant while the first term will be linear. What we could do is use one bit to tell the receiver whether the output was typical and atypical. If it is atypical, we can send it without compression (thus sending $\log_2(|A|^n) = n\log_2(|A|)$), and if it typical, we can then compress it. The dominant contribution to the expected length then occurs from typical outputs, because of the rarity of atypical ones.

How do we compress the source output if it's typical? One of the simplest ways theoretically (but this is not practical) is to calculate the number of typical outputs, and then assign a number (in binary repesentation) to each output. This compresses the source to $\log_2$ of the number of typical outputs. We will do this and get an upper bound of $nH(p) + o(n)$ bits, where the "little o" notation means that the expression divided by n goes to zero as n goes to infinity.

How do we calculate the number of typical outputs? For each typical vector of numbers n_i, we have that the number of outputs is

$$\binom{n}{n_1, n_2, \ldots, n_{|A|},}$$

and there are at most $n^{|A|}$ such vectors (this is a crude upper bound; we could for example improve it to $(2\epsilon n)^{|A|}$). Multiplying, we get that the number of typical outputs is

$$n^{|A|}\binom{n}{(p_1 \pm \epsilon)n, (p_2 \pm \epsilon)n, \ldots, (p_{|A|} \pm \epsilon)n},$$

where we can choose the $p_i \pm \epsilon$ to maximize the expression. Taking logs, we get that the number of bits required to send a typical output is at most

$$|A| \log_2 n + nH(p) + cn\epsilon,$$

for some constant c. The first term is neglibile for large n, and we can let ϵ go to zero as n goes to ∞ so as to get compression to $nH(p) + o(n)$ bits.

In summary, Shannon's noiseless theorem says that we need to transmit $H(P)n$ bits and this is can be essentially achieved. We'll see next a much more practical way (Huffman codes) to do the compression and although it often does not quite achieve the Shannon bound, it gets fairly close to it.