SOME EXAMPLES FROM INFORMATION THEORY (AFTER C. SHANNON).

1. Some easy problems.

1.1. **Guessing a number.** Someone chose a number x between 1 and N. You are allowed to ask questions: "Is this number larger than ....?". What is the smallest number of questions that will always identify x? Try it for $N = 2, 32, 1024$.

1.2. **Finding a fake coin.** You are given N coins, one of which is fake (it is lighter than the rest). You are allowed to compare weights of any two groups of coins (each such comparison gives you three possible answers: the first group is lighter, of the same weight, or heavier, than the second). How many weightings do you need to determine which coin is fake? Try it for $N = 3$ and $N = 81$.

1.3. **n-ary notation.** How many digits do you need to write the number N in binary (base 2), trinary (base 3), decimal (base 10) notation? What about n-ary notation?

1.4. **Guessing a letter.** Someone chooses a word at random from Webster's dictionary (all words are equally likely to be chosen), and takes its first letter (we'll denote it by x). You are allowed to present that person with any group of letters of your choosing (containing letters of your choice) and ask him whether x is in this group or not. The game is played many times. Can you devise a strategy using which you will be able to guess the letter asking the smallest number of questions *on average*? What is that number?

You might be tempted to answer $\log_2 26 \approx 4.7$. We'll show later that it is actually possible to do this (though it involves combining several games into one; but for now let us assume that we can guess the value of $1 \leq x \leq N$ by using $\log_2 N$ questions on average). Moreover, this would be the optimal answer if all letters were equally likely to occur. However, this is not the case (see Figure 1).

In reality not all values of x are equally likely. For example, in my dictionary of 22890 words, there will be very few words that start with X (154) or Z (564) and many more that start with N (941). In my electronic dictionary, most letters occurred as first letters of about 920 words, with the exception of X and Z. The frequency with which X occurs is thus only about 20% of the frequency of typical letters; the frequency of Z is about 60%. This means that the probability of seeing X *or* Y (80%) is thus about the same as seeing any other letter.

Let's put on some special glasses that make us see all letters fine, except that when we are shown X or Z we see the same letter, Ξ.

Now we effectively have a new alphabet of 25 letters: A, B, C, ..., W, Y, Ξ, *all of which are equally likely* (roughly; the frequency of Ξ is 718). Thus we'll need $4.64 = \log_2 25$ choices to narrow it down to one of these letters. In most cases, this tells us which letter was chosen, except when we get the letter Ξ (which happens 728/22890, or about 3% of the time). So 3% of the time we need to ask an extra question. Thus on average, we use $4.64 + 0.03$, or about 4.67 questions.

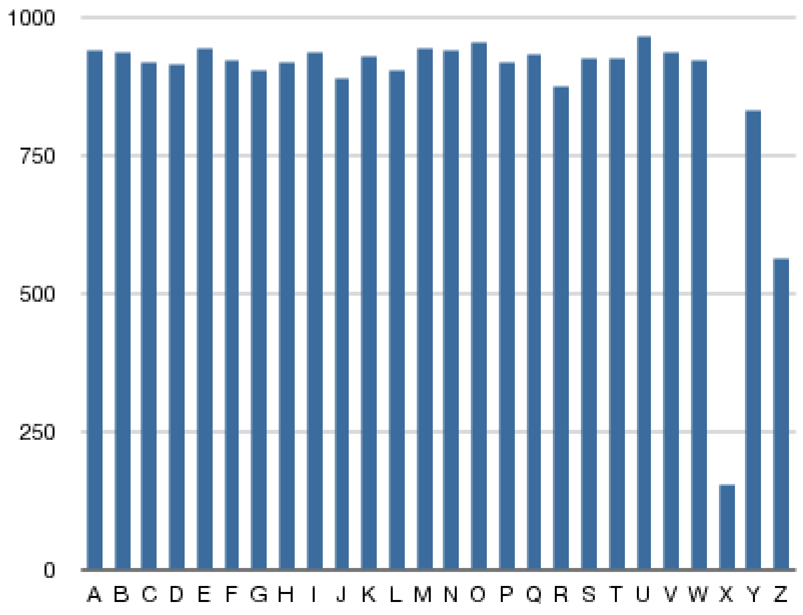

Figure 1. Frequencies of first letters in a dictionary of 22,890 English words.

2. Data transmission.

We wish to transmit data via a telegraph. The telegraph is a device that can transmit, every second, one of two symbols: • (dot) and — (dash). In other words, every second we can transmit a **bi**nary digi**t**, or **bit.**

2.1. Transmitting text.

We now want to transmit English text by telegraph. One way to do this is to enumerate each symbol and transmit its binary representation. For example, if English had only 4 letters, we would then need two bits per symbol, which results in a transmission rate of 0.5 characters per second.

We'll see, however, that this is not the most efficient way to go.

We first make a connection between the transmission problem and the games we discussed earlier. Let us suppose that each symbol in our language is assigned a certain *codeword*, i.e., a combination of dots and dashes that is transmitted whenever we want to send a given symbol. For example, we may agree to that a codeword for the symbol A is • — •. Thus if we wish to transmit the symbol A, we send this particular codeword.

On the receiving end, the receiver does not know which letter we had in mind to send; to him it is an unknown symbol x. He knows, however, the list of our codewords. At first, he knows nothing about x. The moment the first bit arrives, the receiver can eliminate all codewords that do not start with that bit. Each new bit narrows the possibilities for x even further.

This is exactly parallel to the game we played in problem 1.4.

If we have received some bits $B_1 \cdots B_k$ so far, we know that x must be one of the letters whose codewords start with $B_1 \cdots B_k$. Thus you can think of the next bit as the yes/no answer to the question: does x belong to the the set of letters with codewords starting with $B_1 \cdots B_k 1$?

So you can think of the set of codewords as a strategy for playing the game. The first question you ask is: "Is x among the letters whose codewords start with $\bullet$"? If you get a "yes" for the first question, the second question is: "Is x among the letters whose codewords start with $\bullet\ \bullet$?". If the answer to the first question is a "no", the second question is "Is x among the letters whose codewords start with $—\ \bullet$?".

Note that the strategy (=list of questions, i.e., the set of codewords) does not depend on x at all; what depends on x are the specific answers that we get to each question. Thus if you know the strategy and the answers, you know x!

Hence our transmission problem amounts to: (a) pick a strategy (i.e., the codewords); (b) to transmit a symbol S, we send its codeword, i.e., the unique set of answers to the game questions that characterize x. The receiver then just plays the guess-the-x game.

2.2. **Optimizing transmission rate.** The question now is: how does one choose the strategy so as to maximize the average transmission rate (i..e, the average number of questions that need to be asked to identify x).

2.2.1. *Two symbols.* If we had only two symbols, Ψ_1 and Ψ_2, then it would be clear what to do: our strategy would be to ask "is $x = \Psi_1$?" and to transmit $\bullet$ or $—$ accordingly.

2.2.2. *Three symbols.* If we had three symbols, Φ_1, Φ_2, Φ_3, we proceed as in problem 1.4. We choose the two lowest-frequency symbols (let's say Φ_2, Φ_3) and invent a new symbol, Φ_{23} (which stands for Φ_2 or Φ_3). We then choose the most efficient way to transmit Φ_1 and Φ_{23}: we send a $\bullet$ for Φ_1 and a $—$ for Φ_{23}. However, in the latter case we didn't supply enough information for the receiver to determine whether we intended Φ_2 or Φ_3 (we only told him that t we *didn't* send Φ_1). So if we send a $—$, we must follow it by a transmission that determines if we intended a Φ_2 or a Φ_3, i.e., a $\bullet$ if we intended to send Φ_2, and a $—$ for Φ_3. To summarize, here is our "code table":

Symbol	Φ_1	Φ_2	Φ_3
Transmitted code	$\bullet$	$—\ \bullet$	$—\ —$

Let us see what happens in the specific case that Φ_1 occurs 50% of the time and Φ_2, Φ_3 occur 25% of the time. In this case, we need to transmit 1 bit 50% of the time, and 2 bits the remaining 50% of the time. Thus on average, each character costs us 1.5 bits, so that we transmit at $1/1.5 \approx 0.67$ characters per second.

Note that this is clearly the most efficient way to go. To beat 1.5 bits per symbol, we would need to assign a 1-bit code to at least one symbol. The other symbols must get two bits (or else the transmission could be ambiguous: if we e.g. assign $\bullet$ to Φ_1, $—$ to Φ_2 and some two-bit symbol (say $—\ \bullet$) to Φ_3, then one cannot unambiguously decode $—\ \bullet$. It may indicate either a transmission of $\Phi_2\Phi_1$ or Φ_3). But clearly we achieve the best transmission speed if we assign the shortest transmission code to the most frequent symbol.

2.2.3. *Four symbols.* Let us now analyze the situation with 4 symbols, $\Phi_1, \Phi_2, \Phi_3, \Phi_4$. We proceed as before, inventing a new symbol for the two least frequent characters, e.g., Φ_3 and Φ_4. Let us call this new symbol Φ_{34}. Next, choose a code for $\Phi_1, \Phi_2, \Phi_{34}$ as we did before. (Here one should take care to take the two least frequent symbols and replace them by a new symbol. It could be e.g. that the two least frequent symbols are Φ_2 and Φ_{34}, in which case our two new symbols are Φ_1 and Φ_{234}. Or it could happen that the two least frequent symbols are Φ_1 and Φ_2, whence we would take Φ_{23} and Φ_{34}, and so on.)

We claim that this is the most efficient way to go. Indeed, our scheme assigns the two longest code words to the least frequent symbols, which clearly is necessary for any coding scheme. Here are two examples, with their respective transmission codes:

Symbol	Φ_1	Φ_2	Φ_3	Φ_4
Frequency	81%	9%	9%	1%

Symbol	Φ_1	Φ_2	Φ_3	Φ_4
Frequency	30%	25%	25%	20%

Symbol	Φ_1	Φ_2	Φ_3	Φ_4
Transmission code	•	— •	— — •	— — —

Symbol	Φ_1	Φ_2	Φ_3	Φ_4
Transmission code	— •	— —	• •	• —

These encodings require $0.81 + 2 \times 0.09 + 3 \times 0.10 = 1.29$ bits per symbol in the first case, and 2 bits per symbol in the second case. The corresponding transmission rates are 0.78 an 0.5 symbols per second. Note that in the first case our encoding gives a 56% improvement over the "obvious" encoding in which every symbol receives 2 bits.

2.2.4. *Problem.* (a) Explain why in the first encoding above you couldn't encode Φ_3 or Φ_4 by two bits.
(b) Work out a code for transmitting the English language; see Table 1 and Figure 2 for the relative frequencies of letters in English.

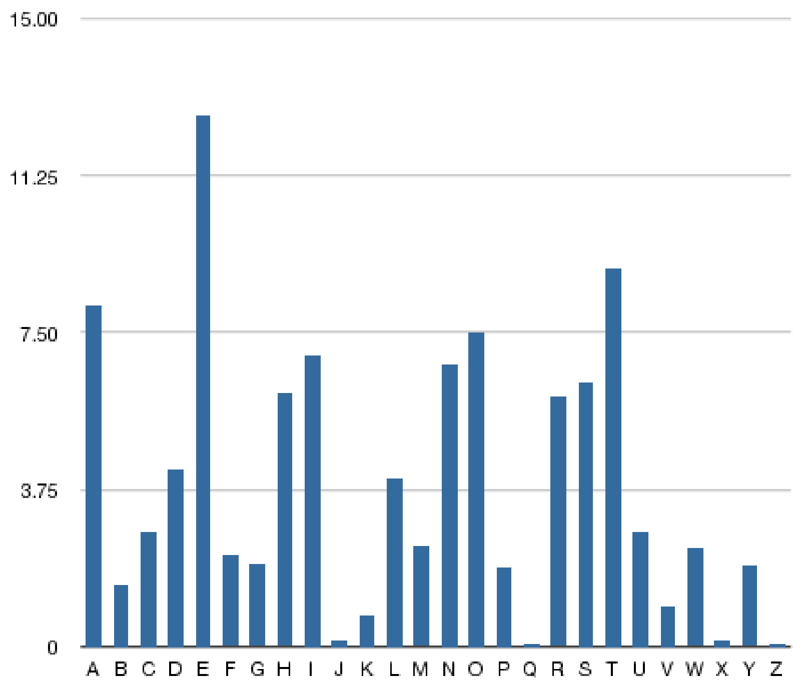

FIGURE 2. Relative frequencies of English letters, %.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
8.2	1.5	2.8	4.3	12.7	2.2	2.0	6	7	0.2	0..8	4	2.4	6.7	7.5	1.9	0.1	6	6.3	9	2.8	1	2.4	0.2	2	0.1

TABLE 1. Frequency of occurrence of English letters (relative frequency, %).

3. Data compression.

3.1. **Grouping.** You may have noted that the transmission codes that we have invented seem to not be the most efficient. Consider for example the situation that we have two symbols, Ψ_1 and Ψ_2. Our scheme will produce a symbol rate of 1 symbol per second, no matter what the probabilities of Ψ_1 and Ψ_2 are. This looks wasteful, however, in the case that one of the symbols is very infrequent. Imagine, for example, that Ψ_1 occurs 90% of the time and Ψ_2 occurs only 10% of the time. Why should we allocate each one of them a whole bit? Clearly, Ψ_1 should get a shorter codeword than Ψ_2. Of course, the problem is that we don't know how to make codewords shorter than 1 bit.

It is clear that this happens because we have too few symbols.

The solution is to accumulate symbols and encode blocks of several symbols at once.

In the case above, let us consider the various frequencies of *pairs* of symbols (we'll assume that symbols occur independently, more on this later):

Pair of symbols	$\Psi_1\Psi_1$	$\Psi_1\Psi_2$	$\Psi_2\Psi_1$	$\Psi_2\Psi_2$
Frequency	1%	9%	9%	81%

Let us denote by Φ_1 the pair $\Psi_1\Psi_1$, by Φ_2 the pair $\Psi_1\Psi_2$, by Φ_3 the pair $\Psi_2\Psi_1$ and by Φ_4 the pair $\Psi_2\Psi_2$.

We've seen that we can transmit 4 symbols with such frequencies at the rate of 0.78 symbols per second. Since transmitting a single symbol Φ_i amounts to transmitting a pair of symbols $\Psi_j\Psi_k$, the transmission rate for the original symbols Ψ_1, Ψ_2 is $2 \times 0.78 = 1.56$ characters per second, a 56% improvement over what we could do before.

One can further improve the situation by accumulating more symbols. How well can you do? The answer is given by the following theorem of Shannon:

Theorem 1. *Assume that symbols $\Sigma_1, \ldots, \Sigma_n$ occur with frequencies $f_1, \ldots, f_n$. Let $H = -\sum f_i \log_2 f_i$. Then:*
(a) It is not possible to transmit these symbols at at average rate of more that $1/H$ symbols per second.
(b) For any $\varepsilon > 0$, there is a collection of codewords that permits you to transmit at the rate of $1/H - \varepsilon$ symbols per second.

In our example, $H = 0.1 \log_2 0.1 + 0.9 \log_2 0.9 \approx 0.47$, so the optimal transmission rate predicted by Shannon's theorem is approximately 2.1 characters per second.

3.2. **Problem.** (a) What rate do you get if you accumulate 3 characters?
(b) Compute the maximal transmission rate for the English alphabet.

3.3. **The letters are not independent.** Once we decide to aggregate symbols, we might as well note that the frequency of a given pair need not be the product of frequencies of the constituent symbols. For example, even the frequency of the pair 'QU' in English is much higher than the product of the frequencies of 'Q' and 'U'. This is due to the fact that various letters in English are correlated.

Thus in making the table of frequencies of pairs, we might as well take the real frequencies, as they occur in English.

3.4. **Application: Data compression.** The procedure we described is actually quite close to how many lossless compression programs work (such as ZIP). Computer data is naturally stored in bits; however, not all bit patterns are equally likely to occur in a given file (for example, the bit patterns 0000000000000000 and 1000000010000000 never occur in plain English text files). Thus our situation is akin to that of §3.1. We group bits into blocks of some length (let's say, 16 bits, which results in 65,536 blocks). We then think of each block as a separate symbol. We go on to analyze the frequencies with which the various blocks occur in original file. Next, we find an encoding scheme like in §2.2. In other words, we determine the codewords for each of our 65,536 symbols.

We now write the compressed file. First, we write all the codewords at the beginning of the file. Then we read data from the original file and encode it using our encoding. If the transmission rate for our encoding is sufficiently big, the resulting file will be smaller than the original one (even counting the extra space we need to save the table of codewords). (*Why?*)

"Unzipping" the file amounts to decoding the contents of the file using the symbol table.

4. Notes and further reading.

The remarkable quantity $H = -\sum f_i \log f_i$ is called *entropy*. The notion of entropy arose in physics in Boltzman's treatment of thermodynamics. Amazingly enough, ideas from thermodynamics can be applied in information theory (and elsewhere in mathematics); this is in essence the basis of Shannon's work.

For further reading on this topic, consider the following books:

R. Ash, *Information Theory*, Dover Publications, New York (reprint of 1965 Interscience Publishers Edition). The first several chapters of this book require very little background, other than perhaps some basic understanding of elementary probability theory.

C. Shannon and W. Weaver, *The mathematical theory of communication*, University of Illinois Press, 1963. This book contains both an expository introduction by W. Weaver and the book form of Shannon's original paper from 1948.

Dimitri Shlyakhtenko, Department of Mathematics, UCLA.