Chapter 2

Coding and Entropy

Note. This topic is used for the second lecture to retain the attention of students with mathematical interests, and in class I jump quickly to the mathematics. At Berkeley, information theory is taught in a graduate course but not an undergraduate one, so I assume my students have not seen any of this material. The final section summary should be comprehensible even if all the math is skipped.

One of my pet peeves is academic topics which have acquired misleading names. Today’s topic grew from a 1948 Shannon paper with the title “a mathematical theory of communication” but has subsequently acquired the hugely misleading name Information Theory (W). The latter Wikipedia page indicates the (surprising large) scope of this particular topic, and the thought-provoking book Information: A Very Short Introduction by Floridi gives one view of how it fits into the much bigger picture of what “information” really is. In one lecture all I can do is present one or two of the central mathematical points, which I try to motivate by introducing two somewhat paradoxical statements (A,B) in the next section.

2.1 On opposites and similars

Recently my (non-mathematician) wife asked me how to make reduced size copies on our home printer. Having no memory for such things, I consulted the manual, and the following conversation ensued.

David: I see you’ve written down notes on how to make enlargements
Katy: Yes, I know how to enlarge
David: Well, reducing is just the same
Katy: (in that particular tone of voice wives use on dense husbands) No, dear, reducing is the opposite of enlarging.
What I meant was: instead of setting to 150%, set to 67%. So I was employing the phrase *the same* in a rather special way, to mean *if you know how to do one then you know how to do the other*. Incidentally this is the concept underlying *computational complexity theory* – there is a known large collection of “hard” algorithmic problems such that, if one could be solved, then they all could be solved. And a million dollar Clay prize for proving they can’t!

Moving on to the actual topic of this lecture, *code* and *coding* have several meanings, but we will be concerned with the following two.

- **Encryption**: coding for secrecy

  familiar from old spy novels and from modern concerns about security of information sent over the internet.

- **Compression**: coding to make a text shorter

  useful both in data storage and in data transmission, because there is some “cost” both to storage space and transmission time. These differ in the obvious way. Compressing files on your computer will produce, say, a `.zip` file, and the algorithms for compressing and decompressing are public. Encryption algorithms in widespread use are commonly like *public-key cryptography* (WK) in that the logical form of the algorithms for encryption and decryption are public, but a private key (like a password) is required to actually perform decryption. In contrast, intelligence agencies presumably use algorithms whose form is secret. For concreteness, in this lecture I talk in terms of coding English language text, but the issues are the same for any kind of data.

  Intuitively there seems no particular connection between encryption and compression – if anything, they seem opposites, involving secrecy and openness. But a consequence of the mathematical theory outlined in this lecture is that

  (A) finding good codes for encryption is the same as finding good codes for compression.

That’s the first of our “opposite but similar” observations, and here is the second. Writing an English language document – a lecture like this one, for instance – with conscious intent to have a particular meaning, seems a complete opposite of anything to do with chance or randomness. Now the theory of coding starts by supposing a particular form of randomness for the data; one can design and test algorithms based on this theory and they
work very well in practice, to the extent of testable quantitative predictions. But why?

(B) If you designed a vehicle to work well as an airplane, you wouldn’t expect it to work well as a submarine. So why do algorithms, designed to work well on random data, in fact work well in the completely opposite realm of meaningful English language?

Briefing: on “random”. In everyday language the word random often carries the connotation of “equally likely”, e.g. in the phrase “pick at random”. Mathematicians like me use it to mean “randomness enters in some way”. In particular, “algorithms are designed to work well on random data” does not refer to completely random letters (the monkeys on typewriters (W) metaphor) but to the notion of stationary in section 2.3.

2.2 A verbal argument for (A)

A code or cipher transforms plaintext into ciphertext. The simplest substitution cipher (W) transforms each letter into another letter. Such codes – often featured as puzzles in magazines – are easy to break using the fact that different letters and letter-pairs occur in English (and other natural languages) with different frequencies. A more abstract viewpoint is that there are 26! possible “codebooks” but that, given a moderately long ciphertext, only one codebook corresponds to a meaningful plaintext message.

Now imagine a hypothetical language in which every string of letters like QHISKUUUC... had a meaning. In such a language, a substitution cipher would be unbreakable, because an adversary seeing the ciphertext would know only that it came from of 26! possible plaintexts, and if all these are meaningful then there would be no way to pick out the true plaintext. Even though the context of secrecy would give hints about the general nature of a message – say it has military significance, and only one in a million messages has military significance – that still leaves $10^{-6} \times 26!$ possible plaintexts.

Returning to English language plaintext, let us think about what makes a compression code good. It is intuitively clear that for an ideal coding we want each possible sequence of ciphertext to arise from some meaningful plaintext (otherwise we are wasting an opportunity); and it is also intuitively plausible that we want the possible ciphertexts to be approximately equally likely (this is the key issue that the mathematics deals with).

Suppose there are $2^{1000}$ possible messages, and we’re equally likely to want to communicate each of them. Then an ideal code would encode each as
CHAPTER 2. CODING AND ENTROPY

a different 1000-bit (binary digit) string, and this could be a public algorithm for encoding and decoding. Now consider a substitution code based on the 32 word “alphabet” of 5-bit strings. Then we could encrypt a message by (i) apply the public algorithm to get a 1000-bit string; (ii) then use the substitution code, separately on each 5-bit block. An adversary would know we had used one of the 32! possible codebooks and hence know that the message was one of a certain set of 32! plaintext messages. But, by the “approximately equally likely” part of the ideal coding scheme, these would be approximately equally likely, and again the adversary has no practical way to pick out the true plaintext.

Conclusion: given a good public code for compression, one can easily convert it to a good code for encryption.

2.3 The asymptotic equipartition property

We must jump into math theory to state a non-elementary result, and accompany it with some discussion. The basis of the mathematical theory is that we model the source of plaintext as random “characters” \(X_1, X_2, X_3, \ldots\) in some “alphabet.” It is important to note that we do not model them as independent (even though I use independence as the simplest case for mathematical calculation later) since real English plaintext obviously lacks independence. Instead we model the sequence \((X_i)\) as a stationary process, which basically means that there is some probability that three consecutive characters are CHE, but this probability does not depend on position in the sequence, and we don’t make any assumptions about what the probability is,

For any sequence of characters \((x_1, \ldots, x_n)\) there is a likelihood

\[
\ell(x_1, \ldots, x_n) = P(X_1 = x_1, \ldots, X_n = x_n).
\]

The stationarity assumption is that for each time \(t\)

\[
P(X_{t+1} = x_1, \ldots, X_{t+n} = x_n) = P(X_1 = x_1, \ldots, X_n = x_n).
\]

Consider the empirical likelihood

\[
L_n = \ell(X_1, \ldots, X_n)
\]

which is the prior chance of seeing the sequence that actually turned up. The central result (non-elementary; I teach it in a graduate course as the Shannon-McMillan-Breiman theorem) is
2.3. THE ASYMPTOTIC EQUIPARTITION PROPERTY

The asymptotic equipartition property (AEP) \( W \). For a stationary ergodic\(^1\) source, there is a number \( \mathcal{E}nt \), called the entropy rate\(^2\) of the source, such that for large \( n \), with high probability

\[
-\log_2 L_n \approx n \times \mathcal{E}nt.
\]

The rest of this section is the mathematical discussion of the theorem that I say in class. I’m not going to attempt to translate it for the general reader, who should skip to the next section to see the relevance to coding. It is conventional to use base 2 logarithms in this context, to fit nicely with the idea of coding into bits.

For \( n \) tosses of a hypothetical biased coin with \( P(H) = 2/3, P(T) = 1/3 \), the most likely sequence is \( \text{HHHHHH...HHH} \), which has likelihood \((2/3)^n\), but a typical sequence will have about \( 2n/3 \) H’s and about \( n/3 \) T’s, and such a sequence has likelihood \( \approx (2/3)^{2n/3}(1/3)^{n/3} \). So

\[
\log_2 L_n \approx n\left(\frac{2}{3}\log_2 \frac{2}{3} + \frac{1}{3}\log_2 \frac{1}{3}\right).
\]

Note in particular that log-likelihood behaves differently from the behavior of sums, where the CLT implies that a “typical value” of a sum is close to the most likely individual value.

Recall that the entropy of a probability distribution \( \mathbf{q} = (q_j) \) is defined as the number

\[
\mathcal{E}(\mathbf{q}) = -\sum_j q_j \log_2 q_j.
\]

The AEP provides one of the nicer motivations for the definition, as follows. If the sequence \( (X_i) \) is IID with marginal distribution \( (p_a) \) then for \( \mathbf{x} = (x_1, \ldots, x_n) \) we have

\[
\ell(\mathbf{x}) = \prod_a p_{n_a}(\mathbf{x})
\]

where \( n_a(\mathbf{x}) \) is the number of appearances of \( a \) in \( \mathbf{x} \). Because \( n_a(X_1, \ldots, X_n) \approx np_a \) we find

\[
L_n \approx \prod_a p_{np_a}^{n_{p_a}}
\]

\[
-\log_2 L_n \approx n \left(-\sum_a p_a \log_2 p_a\right).
\]

\(^1\)The formal definition (Ergodic (adjective) \( W \)) is hard to understand; basically we exclude a source that flips a coin to choose between “all English” and “all Russian”.

\(^2\)Confusingly, entropy rate is often abbreviated to entropy.
So the AEP identifies the entropy rate of the IID sequence with the entropy 
\(E = -\sum_a p_a \log_2 p_a\) of the marginal distributions \(X\).

Let me mention three technical facts.

**Fact 1.** (easy). For a 1-1 function \(C\) (that is, a code that can be be decoded precisely), the distributions of a random item \(X\) and the coded item \(C(X)\) have equal entropy.

**Fact 2.** (easy). Amongst probability distributions on an alphabet of size \(B\), entropy is maximized by the uniform distribution, whose entropy is \(\log_2 B\). So for any distribution on binary strings of length \(m\), the entropy is at most \(\log_2 2^m = m\).

**Fact 3.** (less easy). Think of a string \((X_1, \ldots, X_n)\) as a single random object. It has some entropy \(E_k\). In the setting of the AEP,

\[k^{-1} E_k \to \text{Ent} \text{ as } k \to \infty.\]

Finally a conceptual comment. Identifying the entropy rate of an IID sequence with the entropy of its marginal distribution indicates that entropy is the relevant summary statistic for the non-uniformness of a distribution when we are in some kind of multiplicative context. This is loosely analogous to the topic of Lecture 4, the Kelly criterion, which is tied to “multiplicative” investment.

### 2.4 Entropy as minimum code length

**Briefing:** IID. The common technical abbreviation IID means “independent identically distributed”, meaning roughly the outcomes of a repeated chance experiment. To me, the most helpful example is one person repeatedly throwing darts at a target. The position of the hits can be modelled as IID, but (unlike dice throws) the actual distribution depends on the skill of the particular thrower.

Here we will outline in words the statement and proof of the fundamental result in the whole field. The case of an IID source is *Shannon’s source coding theorem* \((W)\) from 1948. The “approximation” is as \(n \to \infty\).

A string of length \(n\) from a source with entropy rate \(\text{Ent}\) can be coded as a binary string of length \(\approx n \times \text{Ent}\) but not of shorter length.

More briefly, the optimal coding rate is \(\text{Ent}\) bits per letter.
2.5. MORSE CODE AND ASCII

Why not shorter? Think of the entire message \((X_1, \ldots, X_n)\) as a single random object. The AEP says the entropy of its distribution is approximately \(n \times \mathcal{E}nt\). Suppose we can code it as a binary string \((Y_1, \ldots, Y_m)\) of some length \(m\). By Fact 1, the entropy of the distribution of \((Y_1, \ldots, Y_m)\) also \(\approx n \times \mathcal{E}nt\), whereas by Fact 2 the entropy is at most \(m\). Thus \(m\) is approximately \(\geq n \times \mathcal{E}nt\) as asserted.

How to code this short. We give an easy to describe but completely impractical scheme. Saying that a typical plaintext string has chance about 1 in a million implies there must be around 1 million such strings (if more then the total probability would be > 1; if less then with some non-negligible chance a string has likelihood not near 1 in a million). So the AEP implies that a typical length-\(n\) string is one of the set of about \(2^{n \times \mathcal{E}nt}\) strings which have likelihood about \(2^{-n \times \mathcal{E}nt}\) (and this is the origin of the phrase asymptotic equipartition property). So in principle we could devise a codebook which first lists all these strings as integers \(1, 2, \ldots, 2^{n \times \mathcal{E}nt}\), and then the compressed message is just the binary expansion of this integer, whose length is \(\log_2 2^{n \times \mathcal{E}nt} = n \times \mathcal{E}nt\). So a typical message can be compressed to length about \(n \times \mathcal{E}nt\); atypical messages (which could be coded in some non-efficient way) don’t affect the limit assertion.

The second argument is really exploiting a loophole in the statement. Viewing the procedure as transmission, we imagine that transmitter and receiver are using some codebook, but we placed no restriction on the size of the codebook, and the code described above uses a ridiculously large and impractical codebook.

The classical way to get more practical codes is by fixing some small \(k\) and coding blocks of length \(k\), Thus requires a codebook of size \(A^k\), where \(A\) is the underlying alphabet size. However, making an optimal codebook of this type requires knowing the frequencies of blocks that will be produced by the source. Rather than explain further, we shall jump (after a brief historical digression) to more modern codes that don’t assume such knowledge.

2.5 Morse code and ASCII

Invented around 1840, *Morse code* \((W)\) codes each letter and numeral as a sequence of dots and dashes: for instance T is – and Z is —••. Logically this is like coding into a *three*-letter alphabet, because one also needs to indicate (by a pause) the spaces between letters. As is intuitively natural,
common letters (like T) are coded as short sequences and uncommon letters (like Z) are coded as longer sequences. Given frequencies of letters, there is a theoretical optimal way (Huffman coding) to implement such a variable length code, and this has the same intuitive feature. But it’s important to note that Huffman coding is optimal only amongst codes applied to individual letters, and depends on known fixed frequencies for letters.

Developed in the 1960s, ASCII (W) codes letters, numerals and other symbols into 128 7-bit strings: for instance T is 101 0100 and Z is 101 1010. At first sight it may seem surprising that ASCII, and its current extension unicode (W), don’t use variable length codes as did Morse code. But the modern idea is that with any kind of original data one can first digitize into binary in some simple way, and then compress later if needed.

2.6 Lempel-Ziv algorithms

In the 1970s it was realized that with computing power you don’t need a fixed codebook at all – there are schemes that are (asymptotically) optimal for any source. Such schemes are known as Lempel-Ziv style algorithms, though the specific version described below, chosen as easy to describe, is not the textbook form.

Suppose we want to transmit the massage

010110111010|011001000......

and that we have transmitted the part up to |, and this has been decoded by the receiver. We will next code some initial segment of the subsequent text 011001000...... To do this, first find the longest initial segment that has appeared in the already-transmitted text. In this example it is 0110 which appeared in the position shown.

010110111010|011001000......

Writing \( n \) for the position of the current (first not transmitted) bit, let \( n - k \) be the position of the start of the closest previous appearance of this segment, and \( \ell \) for the length of the segment. Here \((k, \ell) = (10, 4)\). We transmit the pair \((k, \ell)\); the receiver knows where to look to find the desired segment and append it to the previously decoded text. Now we just repeat the procedure:

\( n \)The current (September 2011) Lempel-Ziv-Welch (W) article is not helpful for the general reader.
the next maximal segment is 0100 and we transmit this as (7, 4).

How efficient is this scheme? We argue informally as follows. When we’re a long way into the text—position $n$ say—we will be transmitting segments of some typical length $\ell = \ell(n)$ which grows with $n$ (in fact it grows as order $\log n$ but that isn’t needed for this argument). By the AEP the likelihood of a particular typical such segment is about $2^{-\ell \times \mathcal{E}nt}$ and so the distance $k$ we need to look back to find the same segment is order $2^{+\ell \times \mathcal{E}nt}$. So to transmit the pair $(k, \ell)$ we need $\log_2 \ell + \log_2 k \approx \ell \times \mathcal{E}nt$ bits. Because this is transmitting $\ell$ letters of the text, we are transmitting at rate $\mathcal{E}nt$ bits per letter, which is the optimal rate.

2.7 Checking for yourself

On my Mac I can use the Unix \texttt{compress} command, which implements one version of the Lempel-Ziv algorithm. A simple theoretical prediction is that if you take a long piece of text, split it into two halves of equal uncompressed length, and compress each half separately, then the two compressed halves will be approximately the same length. It takes only a few minutes to check an example. I used a text of \textit{Don Quixote}, in English translation, downloaded from Project Gutenberg.

<table>
<thead>
<tr>
<th></th>
<th>uncompressed</th>
<th>compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>first half</td>
<td>1109963</td>
<td>444456</td>
</tr>
<tr>
<td>second half</td>
<td>1109901</td>
<td>451336</td>
</tr>
<tr>
<td>whole</td>
<td>2219864</td>
<td>895223</td>
</tr>
</tbody>
</table>

The prediction works pretty well. Further predictions can be made based on the notion that the algorithm incurs some “start-up cost” before the coding becomes efficient, implying

- The compressed size of a complete text will be shorter than the sums of compressed sizes of its parts. (We see this in the example above, though the difference is very small).

- For a text broken into pieces of different sizes, the compression ratio for the pieces will be roughly constant but also will tend to decrease slightly as size increases.

To illustrate the latter, I used the \texttt{LaTeX} text of the Grinstead-Snell textbook \textit{Introduction to Probability}. 

2.8 ... but English text is not random

So one could just demonstrate that compression algorithms work in practice on natural English text, and stop. But our conceptual question (B) remains – why do algorithms, designed to work well on random data, in fact work well in the completely opposite realm of meaningful English language?

A standard explanation goes as follows. Do we expect that the frequency of any common word (e.g. “the”) in the second half of a book should be about the same as in the first half? Such “stabilization of frequencies” seems plausible – we are not looking at meaning, just syntax, which doesn’t change through the book. This idea of “the rules are not changing” suggests the analogy between written text and a deterministic physical system. An iconic mental picture of the latter is “frictionless billiard balls” which, once set in motion, continue bouncing off each other and the table sides forever. For (certain kinds of) physical systems, *ergodic theory* (alas the (W) article is too abstract to be helpful) predicts “stabilization of frequencies” – e.g. the proportion of time a ball spends near a corner should be about the same in the first hour as in the second hour. One can introduce randomness into the story by taking, for the physical system, a random time as “time 0”, or a random page as “page 0” in a text, and then counting time relative to this start. And the notion of “stabilization of frequencies” turns out to be mathematically equivalent to saying that by a special choice of a random initial state (e.g. what we would see at a time chosen at random from a very long time interval) one sees a stationary random process in the sense (2.1). Granted this as a model for English text, we get both “stabilization of frequencies” and the theory for coding that we described

<table>
<thead>
<tr>
<th>chapter</th>
<th>uncompressed</th>
<th>compressed</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>101082</td>
<td>46029</td>
<td>.465</td>
</tr>
<tr>
<td>2</td>
<td>73966</td>
<td>32130</td>
<td>.434</td>
</tr>
<tr>
<td>3</td>
<td>139490</td>
<td>61571</td>
<td>.441</td>
</tr>
<tr>
<td>4</td>
<td>123784</td>
<td>53962</td>
<td>.436</td>
</tr>
<tr>
<td>5</td>
<td>100155</td>
<td>43076</td>
<td>.430</td>
</tr>
<tr>
<td>6</td>
<td>134256</td>
<td>57577</td>
<td>.429</td>
</tr>
<tr>
<td>7</td>
<td>39975</td>
<td>18021</td>
<td>.451</td>
</tr>
<tr>
<td>8</td>
<td>39955</td>
<td>18759</td>
<td>.470</td>
</tr>
<tr>
<td>9</td>
<td>90019</td>
<td>39853</td>
<td>.443</td>
</tr>
<tr>
<td>10</td>
<td>79560</td>
<td>35058</td>
<td>.441</td>
</tr>
<tr>
<td>11</td>
<td>166626</td>
<td>69181</td>
<td>.415</td>
</tr>
<tr>
<td>12</td>
<td>56463</td>
<td>25299</td>
<td>.448</td>
</tr>
</tbody>
</table>
earlier, as mathematical consequences.

What is unsatisfactory about that explanation? Well, we are asked to accept, in this particular setting of writing text, the analogy between conscious decisions and a physical system. But it is hard to think of another setting where conscious decisions of a single individual can reasonably be modeled probabilistically, so it begs the question of what is so special about writing text.

2.9 Wrap-up

For the topic of this lecture

- There is extensive mathematical theory, and algorithms based on the theory are used widely.

- Some consequences of theory are readily checkable.

- The use of probability is conceptually subtle. We don’t think of speech or writing as random in everyday life, not does it fit naturally into neat philosophical categories like “intrinsic randomness” or “opinion/lack of knowledge randomness”.

- But there is no explanation of why algorithms work except via a model of randomness.

At the end of Lecture 1 I expressed reservations about the “suppose …….” methodology, and an unusually perceptive reader may have wondered in what way our “suppose English text is like a stationary random sequence” is any different. One answer is that we are not inventing specific rules, just saying it is some unspecified process whose structure is not changing in time. A different topic with the same “not a specific rule” feature will appear in Lecture 3 – modeling a “fair game” as a martingale.