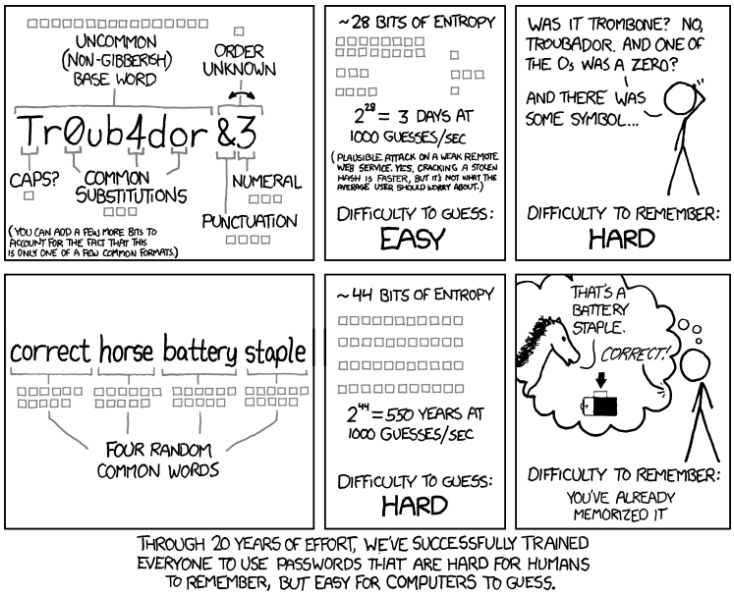
Chapter 7

Coding and entropy

Note. *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.*

Figure 7.1: `xkcd.com/936`



7.1 Introduction

In an earlier survey I asked students to write down a common five-letter English word. I start this lecture by showing the `xkcd` cartoon in Figure 7.1 and then demonstrate the cartoon's essential truth by using a password strength checker to assess the strengths of

- a concatenation of 4 of the students' words
- a volunteer student's password.

Invariably the former is judged "strong" or "very strong" and the latter "weak" or "medium".

This lecture introduces a few topics from a big field known misleadingly as *Information Theory* – see the "further reading" section 7.10. Each of the words *coding* and *entropy* have rather specific meanings in this lecture, so I first must explain these meanings.

7.2 Entropy as a measure of unpredictability

For a probability distribution over numbers – Binomial or Poisson, Normal or Exponential – the mean or standard distribution are examples of "statistics" – numbers that provide partial information about the distribution. Consider instead a probability distribution over an arbitrary finite set S. Simple concrete examples we have in mind for S are

(i) Relative frequencies of given names (Table 7.1)[1].

(ii) Relative frequencies of letters in the English language (Figure 7.2)

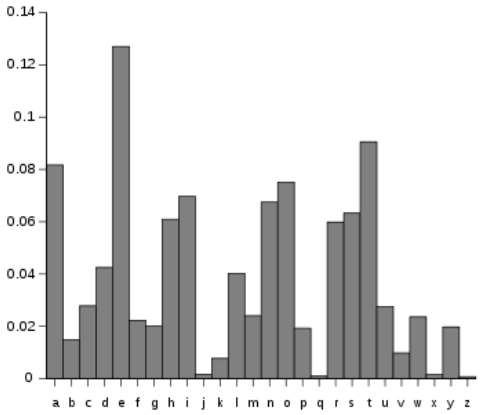
Table 7.1: 2013 U.S. births given names.

Rank	Male name	Percent of total males	Female name	Percent of total females
1	Noah	0.9043%	Sophia	1.1039%
2	Liam	0.8999%	Emma	1.0888%
3	Jacob	0.8986%	Olivia	0.9562%
4	Mason	0.8793%	Isabella	0.9161%
5	William	0.8246%	Ava	0.7924%
6	Ethan	0.8062%	Mia	0.6844%

(iii) Relative frequencies of words in the English language.

[1] The extensive such data from the Social security site is an interesting source for student projects.

Figure 7.2: Relative frequencies of letters in the English language (from Wikipedia)



(iv) Relative frequencies of phrases in the English language[2].

For a probability distribution $\mathbf{p} = (p_s, s \in S)$ on such sets S it does not make sense to talk about *mean* or *standard deviation*. But it does make sense to devise statistics that involve only the *unordered* set of values $\{p_s\}$, and the particular statistic relevant to this lecture is

$$\mathcal{E}(\mathbf{p}) := -\sum_s p_s \log p_s$$

which is called the *entropy* of the probability distribution $\mathbf{p}$. This terminology is confusing, patly because "entropy" is often used for what is properly called *entropy rate* (section 7.4), and partly because of the only indirectly related notion of *entropy* in statistical physics.

A basic fact is that the uniform distribution on an n-element set has entropy $= \log n$ whereas the "degenerate" distribution concentrated at a single element has entropy zero. The entropy statistic serves to place a distribution on the spectrum from degenerate to uniform; entropy is of described as "amount of randomness" but for our purposes is better regarded as a measure of *unpredictability*. Note that many other statistics serve the same general purpose, as discussed further in Lecture xxx under the phrase diversity statistic.

[2]See the Google Books Ngram Viewer, which has various interesting uses. To see usage of *data* as singular or plural, compare frequencies of "the data is" and "the data are".

A good way to interpret the numerical value of $\mathcal{E}(\mathbf{p})$ is via the "effective number" N_{eff} – the number[3] such that the uniform distribution on N_{eff} elements has the same statistic. See section xxx for an illustration concerning the changes over time in the diversity of given names (Table 7.1).

Entropy in physics. The reader has likely seen a statement of the second law of thermodynamics in a verbal form such as

> the total entropy of any isolated thermodynamic system increases over time, approaching a maximum value

and the informal description of entropy as a measure of *disorder*. When expressed in mathematical terms one can indeed see connections between this physics formulation of *entropy* and our definition of $\mathcal{E}(\mathbf{p})$, but this connection is not particularly helpful for an introductory treatment of the topic of this lecture.

7.3 Coding, compression and encryption

The word *coding* nowadays primarily means "writing computer code" but here we are concerned with representing data in some convenient form. A simple example is the original ASCII scheme (section 7.6) for representing letters and typewriter symbols in binary. In choosing how to code a particular type of data there are several issues one might consider.

- *Compression*: coding to make a text shorter

is useful both in data storage and in data transmission, because there is some "cost" both to storage space and transmission time.

- *Encryption*: coding for secrecy

is familiar from old spy novels and from modern concerns about security of information sent over the internet. These differ in an 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 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

[3] The solution of $\mathcal{E}(\mathbf{p}) = \log N_{\text{eff}}$, typically not actually an integer.

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.

A third issue I will not discuss is

- robustness under errors in data transmission: **error-correcting code**

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

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

Here is a verbal argument for (*). A code or cipher transforms *plaintext* into *ciphertext*. The simplest substitution cipher 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 QHSKUUC ... 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 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.

7.4 The asymptotic equipartition property

We now 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 implies 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.

To say the setup more carefully, for any sequence of characters $(x_1, \ldots, x_n)$ there is a *likelihood*

$$\ell(x_1, \ldots, x_n) = \mathbb{P}(X_1 = x_1, \ldots, X_n = x_n).$$

The *stationarity* assumption is that for each time t and each sequence $(x_1, \ldots, x_n)$

$$\mathbb{P}(X_{t+1} = x_1, \ldots, X_{t+n} = x_n) = \mathbb{P}(X_1 = x_1, \ldots, X_n = x_n). \tag{7.1}$$

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

The asymptotic equipartition property (AEP) . For a stationary ergodic[4] source, there is a number $\mathcal{E}nt$, called the *entropy rate* 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 $\mathbb{P}(H) = 2/3, \mathbb{P}(T) = 1/3$, the *most likely* sequence is $HHHHHH \dots HHH$, 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

$$\log_2 L_n \approx n(\tfrac{2}{3}\log_2 \tfrac{2}{3} + \tfrac{1}{3}\log_2 \tfrac{1}{3}).$$

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

$$\mathcal{E}(\mathbf{q}) = -\sum_j q_j \log_2 q_j. \tag{7.2}$$

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 $\mathbf{x} = (x_1, \dots, x_n)$ we have

$$\ell(\mathbf{x}) = \prod_a p_a^{n_a(\mathbf{x})}$$

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

$$L_n \approx \prod_a p_a^{np_a}$$

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

[4]The formal definition of ergodic is hard to understand; basically we exclude a source that flips a coin to choose between "all English" and "all Russian".

So the AEP identifies the *entropy rate* of the IID sequence with the *entropy* $\mathcal{E} = -\sum_a p_a \log_2 p_a$ 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 $\mathcal{E}_k$. In the setting of the AEP,

$$k^{-1}\mathcal{E}_k \to \mathcal{E}nt \text{ as } k \to \infty.$$

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 2, the Kelly criterion, which is tied to "multiplicative" investment.

7.5 Entropy rate and minimum code length

Here we will outline in words the statement and proof of the fundamental result in the whole field. The case of an IID source (recall section 2.2) is Shannon's source coding theorem from 1948. The "approximation" is as $n \to \infty$.

> A string of length n from a source with entropy rate $\mathcal{E}nt$ can be coded as a binary string of length $\approx n \times \mathcal{E}nt$ but not of shorter length.

More briefly, the optimal coding rate is $\mathcal{E}nt$ bits per letter.

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..

7.6 Morse code and ASCII

Invented around 1840, Morse code codes each letter and numeral as a sequence of dots and dashes: for instance T is $-$ and Z is $--\bullet\bullet$. 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 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, 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.

7.7 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[5] algorithms, though the specific version described below, chosen as easy to describe, is not the textbook form.

Suppose we want to transmit the message

$$010110111010|011001000\ldots\ldots$$

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 $011001000\ldots\ldots$. 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.

$$01\underline{0110}111010|\underline{0110}01000\ldots\ldots$$

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 ℓ for the length of the segment. In the example, $(k,\ell) = (10,4)$. We transmit the pair (k,ℓ); 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:

$$0101101110\underline{0100}110|\underline{0100}0\ldots\ldots$$

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

[5] The current Wikipedia article is not so helpful for the general reader.

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, ℓ) we need $\log_2 \ell + \log_2 k \approx \ell \times \mathcal{E}nt$ bits. Because this is transmitting ℓ letters of the text, we are transmitting at rate $\mathcal{E}nt$ bits per letter, which is the optimal rate.

7.8 Checking for yourself

On my Mac I can use the Unix `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 *Don Quixote*, in English translation, downloaded from Project Gutenberg.

Table 7.2: Bytes in Don Quixote

	uncompressed	compressed
first half	1109963	444456
second half	1109901	451336
whole	2219864	895223

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 LaTeX text of the Grinsted-Snell textbook *Introduction to Probability*.

Table 7.3: Bytes in Grinsted-Snell

chapter	uncompressed	compressed	ratio
1	101082	46029	.465
2	73966	32130	.434
3	139490	61571	.441
4	123784	53962	.436
5	100155	43076	.430
6	134256	57577	.429
7	39975	18021	.451
8	39955	18759	.470
9	90019	39853	.443
10	79560	35058	.441
11	166626	69181	.415
12	56463	25299	.448

7.9 . . . but English text is not random

So one could just demonstrate that compression algorithms work in practice on natural English text, and stop. But this doesn't address a conceptual issue.

> (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?

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 such physical systems, *ergodic theory* 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 (7.1). Granted this as a model for English text, we get both "stabilization of frequencies" and the theory for coding that we described 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.

7.10 Wrap-up and further reading

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.

In Lecture 4 we saw a context (prediction markets and strategies for fair games) where one can make numerical predictions without needing a very specific model but only assuming a structural property (martingale). This lecture shows the same for the context of data compression, the structural property being stationarity. A third such context is spatial networks (section 9.4 later), the structural property being scale-invariance.

Further reading. This lecture's topic grew from a 1948 Shannon paper with the title "a mathematical theory of communication" and the broad academic field subsequently acquired the name *Information Theory*. This is an unfortunate name – the thought-provoking book *Information: A Very Short Introduction* by Floridi gives one view of how this field fits into the much bigger picture of what "information" really is. The Wikipedia article Information Theory outlines the scope of this academic field, and the Cover - Thomas textbook *Elements of information theory* is a standard first mathematical treatment.