Lecture 11: Coding and Entropy.

David Aldous

October 10, 2017

The concept "information" arises in different ways in different fields of the mathematical sciences – see the book *Information: A Very Short Introduction*. This lecture relates to the field called **Information Theory** – course EE 229A – just one of these fields.

Start with discussing a concept of **entropy**. This word also has several different-but-related meanings in different fields of the mathematical sciences; we focus on the one particular meaning relevant to "Information Theory".

Note: in this lecture **coding** also has a special meaning – representing information in some standard digital way for storage or communication.

Here is our "anchor data".

[show xkcd]

Is the cartoon's message sensible? In the Lecture 1 survey I asked for common 5-letter words, and let us choose

happy apple beach yacht

and use a password strength checker.

[show]

[Volunteer to check your own password]

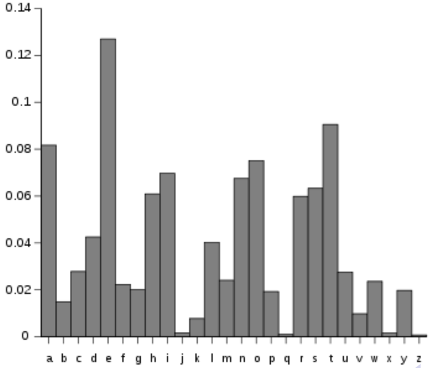
This lecture explains some of the background math.

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

$$\mathbf{p} = (p_s, s \in S)$$

Examples we have in mind for S are
Relative frequencies of letters in the English language



- Relative frequencies of letters in the English language
- Relative frequencies of words in the English language
- Relative frequencies of phrases or sentences in the English language [show Google Ngram]
- Relative frequencies of given names [show]

For such S *mean* does not make sense. But statistics such as

$$\sum_s p_s^2$$

and

$$-\sum_s p_s \log p_s$$

do make sense.

What do these particular statistics $\sum_s p_s^2$ and $-\sum_s p_s \log p_s$ measure?

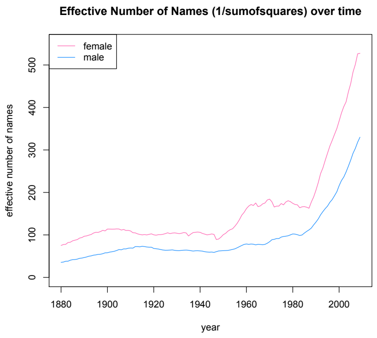
[board]: spectrum from uniform distribution to deterministic.
Interpret as "amount of randomness" or "amount of non-uniformity".

First statistic has no standard name.
Second statistic: everyone calls it the **entropy** of the probability distribution $\mathbf{p} = (p_s, s \in S)$.

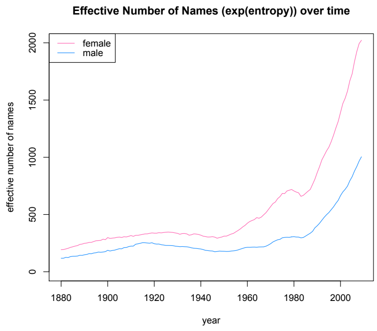
For either statistic, a good way to interpret the numerical value is as an "effective number" N_{eff} – the number such that the uniform distribution on N_{eff} categories has the same statistic.

[show effective-names.pdf – next slide]

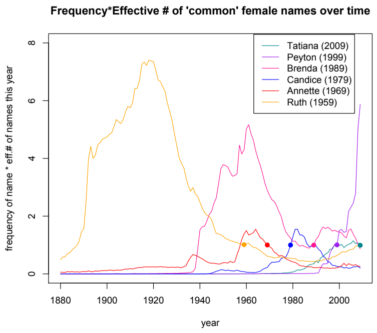
For many purposes the first statistic is most natural – e.g. the chance two random babies born in 2013 are given the same name. The rest of this lecture is about contexts where the **entropy** statistic is relevant.



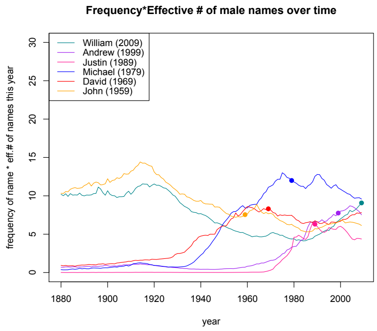
Effective Number of Names (1/sumofsquares) over time
female
male
effective number of names
0
100
200
300
400
500
1880
1900
1920
1940
1960
1980
2000
year



Effective Number of Names (exp(entropy)) over time
female
male
effective number of names
0
500
1000
1500
2000
1880
1900
1920
1940
1960
1980
2000
year



Frequency*Effective # of 'common' female names over time
Tatiana (2009)
Peyton (1999)
Brenda (1989)
Candice (1979)
Annette (1969)
Ruth (1959)
frequency of name * eff.# of names this year
0
2
4
6
8
1880
1900
1920
1940
1960
1980
2000
year



Frequency*Effective # of male names over time
William (2009)
Andrew (1999)
Justin (1989)
Michael (1979)
David (1969)
John (1959)
frequency of name * eff.# of names this year
0
5
10
15
20
25
30
1880
1900
1920
1940
1960
1980
2000
year

In our context it is natural to take logs to base 2. So if we pick a word uniform at random from the 2000 most common English words, this random process has entropy $\log_2 2000 \approx 11$, and we say this as "11 *bits* of entropy".

[show xkcd again]

Of course we can't actually pick uniformly "out of our head" but the notion of "effective population size" holds.

You may have seen the *second law of thermodynamics* [show]

The prominence of entropy in this “physical systems” context has led to widespread use and misuse of the concept in other fields. This lecture is about a context where it is genuinely a central concept.

A simple **coding** scheme is ASCII [show]

In choosing how to code a particular type of data there are three main issues to consider.

- May want coded data to be short, for cheaper storage or communication: **data compression**
- May want secrecy: **encryption**
- May want to be robust under errors in data transmission: **error-correcting code**

[comment on board]

At first sight these are quite different issues, but . . . . . .

Here is a non-obvious conceptual point.

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

Here "the same as" means "if you can do one then you can do the other".

In this and the next 3 slides I first give a verbal argument for this assertion, and this argument motivates subsequent mathematics.

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. Suppose we have a public **ideal code for compression**, which encodes each message as a different 1000-bit string, 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.

Math theory

The basis of the mathematical theory is that we model the source of plaintext as random “characters” $X_1, X_2, X_3, \dots$ 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*

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

The *stationarity* assumption is that for each "time" t (really this is "position in the sequence")

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

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 (Shannon-McMillan-Breiman theorem: STAT205B) is

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

$$-\log L_n \approx n \times \mathcal{E}nt.$$

It is conventional to use base 2 logarithms in this context, to fit nicely with the idea of coding into bits.

I will illustrate by simple calculations in the IID case, but it's important that the AEP is true very generally. We will see the connection with coding later.

For n tosses of a hypothetical biased coin with $\mathbb{P}(H) = 2/3, \mathbb{P}(T) = 1/3$, the *most likely* sequence is $HHHHHH \ldots HHH$, 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

$$\mathcal{E}(\mathbf{q}) = -\sum_j q_j \log_2 q_j. \qquad (2)$$

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 $\mathbf{x} = (x_1, \ldots, x_n)$ 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 $n_a(X_1, \ldots, X_n) \approx np_a$ we find

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

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

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.

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 $(X_1, \ldots, X_k)$ 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.

Entropy as 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 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. 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. I outline an easy to describe, but not the textbook, scheme.

Suppose we want to transmit the massage

$$010110111010|011001000\ldots\ldots$$

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. Here $(k,\ell)=(10,4)$. 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:

$$010110111\underline{0100}110|\underline{0100}0\ldots\ldots$$

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 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.

What part of this theory can we check ourselves?

The Unix `compress` command implements one version of the Lempel-Ziv algorithm. A simple theoretical prediction is that if you take two "similar" long pieces of text, and compress them separately, then the ratios compressed length/uncompressed length should be almost the same.

It takes only a few minutes to check an example. Let me use a text of *History of the Decline and Fall of the Roman Empire*, downloaded from Project Gutenberg.

[do demo]

Note a conceptual point: theory assumes a certain notion of *randomness* (stationarity) but the algorithms actually work well in the completely opposite realm of meaningful language.

So what is the connection with passwords/cryptography?

[show xkcd again and discuss]

Further reading

The standard textbook is Cover and Thomas *Elements of Information Theory*.

Floridi *Information: A Very Short Introduction* gives an overview of the breadth of the concept of "information".

A best-seller popular account is Glick *The Information*.