5.8.3 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 (W) 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 practice 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.

5.8.4 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 (W)) 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.

5.8.5 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 style⁷ algorithms, though the specific version described below, chosen as easy to describe, is not the textbook form.

Suppose we want to transmit the message

\[010110111010|01100100\ldots\]

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 01100100\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.

\[010110111010|0110010000\ldots\]

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 \(\ell\) 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:

\[0101101110100110|0100\ldots\]

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

⁷The current (October 2008) Lempel-Ziv-Welch (W) article needs improvement.
log \ n \text{ 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} \text{ and so the distance } k \text{ we need to look back to find the same segment is order } 2^{+\ell \times \mathcal{E}nt}. \text{ So to transmit the pair } (k, \ell) \text{ we need } \log_2 \ell + \log_2 k \approx \ell \times \mathcal{E}nt \text{ bits. Because this is transmitting } \ell \text{ letters of the text, we are transmitting at rate } \mathcal{E}nt \text{ bits per letter, which is the optimal rate.}