Notes on Information Theory

- Developed by Claude Shannon and a few others. The basic theory was outlined in his 1948 paper: *The Mathematical Theory of Information*: Here is a book based on it by Shannon and Weaver: https://pure.mpg.de/rest/items/item_2383164/component/file_2383163/content

- Shannon is concerned with an engineering problem: the efficient and error-free transmission of information from a sender to a receiver, despite noise that might corrupt the data in transit.

- We, on the other hand, are just concerned with the measure of information, which, intuitively, is the amount of surprise its apprehension causes in the receiver. Alternatively, and more mathematically, it's the measure of the reduction in uncertainty.

- If I tell you that the sun is shining during a day that you know is cloudless, you're not surprised, and so I've lowered your uncertainty about the brightness outside hardly at all. I've sent a message, but little or no information. On the other hand, if you cannot see outside, I may have lowered your uncertainty quite a bit, particularly answering the shouldI-take-an-umbrella question. And so, in this alternative case, I've succeeded in transmitting a good deal of information.

- The information content of a message or data depends upon the probability of an event occurring. We'll try to assume a measure of such a probability that's as objective as possible, although, as you can see in the sun-shining case above, it can depend the recipient's current knowledge and beliefs.

- Let **E** be an event. For instance, the result of a coin flip, namely which side of the coin came up. Or, the event could be the next word that I utter.

- Let **P(E)** be the objectively calculated probability of that event occurring. For instance, that the result will be "heads", or that the word will be "daffodil". Of course, this probability will often be dependent upon may things: whether we believe it's a fair coin, or what I've already said just before uttering the word in question.

- The Shannon information content of transmitting a particular event -- such as telling you that the result was "heads", or the word with "daffodil" – is:

$$S(E) = \frac{1}{log_2(P(E))} = -log_2(P(E))$$

- Why is it Log base 2, instead of the natural Log, or some other base? In theory, it doesn't matter much. But since our digital communications are based on transmitting binary digits

(0 and 1), it's more convenient to use base 2. That way, the numerical measure of information in a message and the number of bits used to transit that message can be easily compared.

Henceforth, we'll assume log(x) to mean log() in base 2.

- For a fair coin, if the event H is that the flip comes up heads, then P(H) = ½ and S(E=H) = log(2) = 1. So one bit of information is transmitted when I tell you that the flip came up heads. Note that we're *not* talking about how I've transmitted the information to you, just that I've given you one bit of information.

- The Shannon information content of two independent events, E1 and E2, coming up one after the other is simply S(E1) + S(E2). This means that it will take a minimum of 2 bits to transmit the results of two flips of a fair coin.

- If I flip 100 times, then about half the time it will be heads, and the other half tails. I will have to send 50 + 50 bits to transmit all the information. We can look at this probabilistically. There's a ½ chance that the flip will be heads and ½ chance that it will be tails, and so, on the average I will send ½ * S(E=H) + ½ * S(E=T) = 1. In general, if there are many possible results of an event, each with its own probability p_i, then

$$S(E) = -\sum_{i=1}^{n} p_i log_2(p_i)$$

- What this tells us is that **S(E) is the minimum number of bits required, on average, to send the results of an event E.**

- Suppose we want to send a printable character. Let's pretend that there are 128 possible printable characters and also pretend that the different characters are equally probable. Why? Because, well, 128 is a power of 2, and powers of 2 are nice and equal probability is nice and pretending is also nice. So, sending the fact that the next letter is "Q", namely S(letter="Q") to a listener would transmit log(128) = 7 bits of information. The average amount of information sent when one letter is sent is

$$S(letter) = \frac{S(letter = A)}{128} + \frac{S(letter = B)}{128} + \cdots = 128 * \frac{S(letter = A)}{128} = 7$$

- All of these calculations are relatively easy when the number of possibilities N of an event E is some power of 2 ($N = 2^R$) and when all the possibilities are equally probable. Then the log(N) will be an integer and S(E) = R. Then it's easy to code the value of the outgoing letter in R bits.

- Let’s change the probabilities for the coin flip. It’s a loaded coin, with the probability P(H) = ¾ and P(T) = ¼ . Let’s sink into Python:

-
  ```
from math import log
   # Shannon info for one possibility given its probability def
   S1(prob):
     return log(1/prob,2)

   # Shannon info for an event, given list of probabilities:
   def S(event):
     return sum([x * S1(x) for x in event])
```

- So, for the loaded coin, S = 0.81 (to 2 decimal places). That means that optimally, we should need less than 1 bit to transmit, on average, the result of a flip. This makes sense, since if the coin were really, really loaded (almost always heads), then very little information is actually transmitted to the receiver: she knows, with high probability, then the result was heads.

- But if we use the simple transmission coding for heads and tails, sending a 1 if heads and a 0 if tails, then we are using 1 bit of transmission capacity to send about 0.81 bits of information. We’re wasting 0.19 bits of capacity every time we do this. What we’d like to do is somehow use a different coding scheme to get to the state where we are using about 0.81 bits <u>on the average</u> to transmit the result of a typical coin flip.

- Instead of sending a single value (1 or 0) to encode the result of a single flip, we’ll try to encode the values to send for pairs of results. In other words, the transmitter will wait for the results of 2 flips before transmitting anything, and will then encode the pair of results into a string of bits, as follows:

  for HH, whose probability is ¾ * ¾ = 9/16, send a single 0 bit
  for HT, prob = 3/16, send 2 bits: 10
  for TH, prob = 3/16, send 3 bits: 110
  for TT, prob = 1/16, send 3 bits: 111

  First of all, you must convince yourself, that the receiver, getting a stream of correctly coded bits, will always be able to decipher the results of the flips encoded in the stream. There is no ambiguity in the coding results. Take for example the stream:

(0 0 1 1 0 1 0 0 1 1 1 0

Reading from left to right, we can correctly divide this up in the following way:

0 – 0 – 1 1 0 – 1 0 – 0 – 1 1 1 – 0 giving us the pair-results:
HH – HH – TH – HT – HH - TT – HH

Now, calculating the average number of bits to send a pair of results:
(9/16 * 1) + (3/16 * 2) + (3/16 * 3) + (1/16 * 3) = 27/16. That means for a single flip's results, we need only half of this = 27/32 bits = 0.84 bits. That's much closer to the ideal minimum of 0.81 bits, though we're not quite there yet. It also suggests that **we can extend this trick of grouping together more than one result, and use variable-length coding to encode all the possible group results. In that way, we might be able to get closer to the ideal minimum.**

- Let's take a different situation. Suppose we flip a perfect 3-sided coin (I'm sure there's one somewhere), with faces: A, B and C. Each face has a 1/3 chance of coming up. How much information is there in the result E?

$$S(E) = \frac{log_2(3)}{3} + \frac{log_2(3)}{3} + \frac{log_2(3)}{3} \approx 1.58$$

Now, to paraphrase and old saying, we cannot send one bit to do a 1.58-bit's job. So we have to resort to some clever coding to get close to that amount per flip result.

We notice the following. If we group together 2 flips, there are 9 possibilities, with 3 flips, there are 27 possibilities, with n results, there are 3^n possibilities. We also notice that sending 2 bits allows for 4 different values to be sent, with 3 bits: 8 values, and with n bits: 2^n values. We know that to send one flip, we need to be able to encode 3 different values, and can only do so with at least 2 bits.

So, encoding for just one flip: we can send the result A by sending 00, B=01 and C=10. Note however, that 11 is never sent, and is useless. This is a waste. This is an average of 2 bits/flip.

How about coding pairs of results? We'll need to encode 9 different possibilities, and so we can't use 3 bits (only 8 possibilities), and must use 4 bits. That again relegates us to 4 bits/pair-of-flips = 2 bits/flip. No improvement.

How about coding triplets of results? We'll need 27 different possibilities, and can get them with 5 bits (32 possibilities). That's 5 bits/triplet-of-results = 1.67 bits/flip. Now we're getting somewhere! Much closer to the theoretical minimum of $log_2(3)$ = 1.58.

In fact, if we think this through, and try to encode N flips, we'll need 3^N possibilities. And they must be provided by R bits, which give us 2^R actual possibilities. Well 3^N can never equal exactly 2^R with integers N and R, but they can get arbitrarily close. So let's pretend that for some large value of N (how many flips we group together to encode), there's a value of R (number of bits to encode them in), such that 3^N ≈ 2^R. Then the number of bits/flip would be R/N. But look: taking logs of both sides: **R/N = $\log_2(3)$** ! In other words, we can get to the theoretical minimum (most efficient coding) by saving N results, and encoding that set into R output bits.

- Let's recap.
    1. If the number of possibilities T happens to be a power of 2 (say, 2^R), and they're all equally probable, then we have the simplest code: we encode them as successive binary numbers in R binary digits. For instance, if we have 4 possibilities (values) to encode: A, B, C and D, then we can encode them in 2 bits as 00, 01, 10, 11.
    2. If the number of possibilities T is not a power of 2, but they're still equally probable, then we can find a group of size N such that T^N is close to but less than some power of 2, say 2^R. Then we'll encode groups of N symbols together.
    3. If the possibilities have different probabilities, then we'll work on assigning different encoding lengths to the different symbols, with the symbol with the highest probability given the shorted encoding.

- **Exercises**:
    1. Suppose you want to send the results of an event E with 5 possible outcomes (say, the flip of your usual 5-sided coin). Assume that all the possibilities are equally probable. S(E) = $\log_2(5) \approx 2.32$ so that's the minimum number of bits needed to send a result. Find a "pretty good" coding that's close to that optimal value.
    2. A somewhat harder problem: Suppose the event E is the result of flipping a 3-sided coin, with faces: H (heads), T (tails) and M (midriffs). Suppose M has probability of ½ and the other 2 have a probability of ¼ each. S(E) = 0.5*log(2) + 0.25*log(4) + 0.25*log(4) = 1.5. Find better encoding than simply using 2 bits to encode each result.
    3. An even harder problem (in honor of Amit Narang): You have a 5 sided coin, one favored side has probability of 1/3, each of the other sides has probability of 1/6. Provide the Shannon limit of the number of bits/flip, and a "pretty good" coding scheme.