12 Entropy and Data Compression

12.1 Surprisal and Entropy

Suppose we have a biased coin that comes up heads with probability p that is very small. If we toss the coin and it comes up tails, we do not learn much because it is almost what we expect. But if it comes up heads, we learn a lot more. In particular, less likely events are more informative because they are more surprising. Let us try to define a function which capture the amount of surprise (or surprisal) in an event A. What are the properties such a function should satisfy? The following are natural axioms that make intuitive sense:

1. S should be a function of the probability of the event. So, we can write $S(p)$ where $p \in (0,1)$ is the probability of the event A.
2. $S(p)$ should decrease as p increases since more likely events are less surprising.
3. $S(p)$ should be a continuous function of p since we do not expect the surprise value to suddenly jump
4. For two events A and B, the total surprise should be sum of individual surprises $S(\mathbb{P}(A,B)) = S(\mathbb{P}[A]) + S(\mathbb{P}[B])$. We can also equivalently write the above as $S(p_1 \cdot p_2) = S(p_1) + S(p_2)$ for $p_1, p_2 \in (0,1)$.

It turns out that there is essentially a unique function that satisfies the above axioms. See Figure 1.

Theorem. *The only function satisfying the above properties is* $S(p) = \log \frac{1}{p}$

Technically the base of the logarithm above can be chosen arbitrarily but we will choose it to be 2 and measure the surprisal in bits. We skip the proof of the above theorem.

Entropy

Suppose we have a random variable X. The surprise associated with the event $X = x$ is

$$S(\mathbb{P}(X = x)) = \log \frac{1}{\mathbb{P}[X = x]}.$$

The **entropy** of X is the average surprise on learning the value of X and defined as

$$H(X) \triangleq \sum_x \mathbb{P}[X = x] \cdot \log \frac{1}{\mathbb{P}[X = x]}.$$

Basic Properties of Entropy

1. $H(X) \geq 0$ and $H(X) = 0$ iff X is deterministic. This is easy to see since the surprisal is always non-negative.

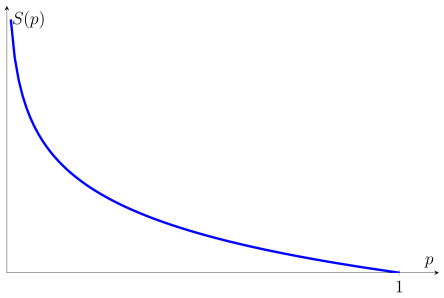

Figure 1. The surprisal function $S(p) = \log(1/p)$ measures the information content of an event. Less probable events carry more information and are more surprising. The function is strictly decreasing and approaches infinity as $p \to 0$.

2. Suppose $X = \mathbb{1}_A$, i.e., X is an indicator variable for an event A. Let us denote $p = \mathbb{P}[X = 1]$ and $1 - p = \mathbb{P}[X = 0]$. Then, the entropy of X is

$$p \cdot \log \frac{1}{p} + (1-p) \log \frac{1}{1-p} \text{ where } p \in [0,1].$$

   This is called the **binary entropy function** $H(p)$ and has maximum average surprise for a random bit at $p = 1/2$.

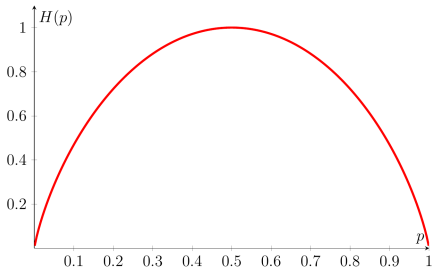

Figure 2. The binary entropy function $H(p)$ represents the average information content of a binary random variable with probability p. It achieves its maximum value of 1 bit at $p = 1/2$, corresponding to a fair coin flip with maximum uncertainty.

3. Suppose X is uniform over $\{1, \ldots, m\}$, i.e., $\mathbb{P}[X = i] = \frac{1}{m}$ for every i. Then, the entropy $H(X) = \sum_{i=1}^{m} \frac{1}{m} \log m = \log m$. It turns out that for any random variable over $\{1, \ldots, m\}$, the entropy $H(X) \leq \log m$. Thus, the uniform distribution over $\{1, \ldots, m\}$ has the largest entropy, which matches with our intuition that entropy captures the randomness inherent in a random variable.

Joint Entropy

Given two random variables X and Y, their joint entropy is:

$$H(X,Y) \triangleq \sum_{x,y} \mathbb{P}[X = x, Y = y] \log \frac{1}{\mathbb{P}[X = x, Y = y]}.$$

As an example, if X is uniform over $\{1, \ldots, m\}$ and Y is uniform over $\{1, \ldots, n\}$, then we have $\mathbb{P}[X = x, Y = y] = \frac{1}{mn}$, and thus, $H(X,Y) = \sum_{x,y} \frac{1}{mn} \log mn = \log mn$. Observe that

$H(X) = \log m$ and $H(Y) = \log n$ and that $H(X,Y) = H(X) + H(Y)$. In fact, for any pair of random variables, we have the following relation

$$H(X,Y) \leq H(X) + H(Y)$$

Equality holds above if X and Y are independent.

Let us give a proof that the entropy of a pair of independent random variables X and Y is the sum of the entropies. Recall that if X and Y are independent, then $\mathbb{P}[X = x, Y = y] = \mathbb{P}[X = x] \cdot \mathbb{P}[Y = y]$. Therefore, we have

$$\log \frac{1}{\mathbb{P}[X = x, Y = y]} = \log \frac{1}{\mathbb{P}[X = x]} + \log \frac{1}{\mathbb{P}[Y = y]}.$$

Since the entropy is the average value of the above quantity, we obtain

$$\begin{aligned} H(X,Y) &= \sum_{x,y} \mathbb{P}[X = x, Y = y] \left(\log \frac{1}{\mathbb{P}[X = x]} + \log \frac{1}{\mathbb{P}[Y = y]} \right) \\ &= \sum_{x} \mathbb{P}[X = x] \left(\sum_{y} \mathbb{P}[Y = y] \right) \log \frac{1}{\mathbb{P}[X = x]} + \sum_{y} \mathbb{P}[Y = y] \left(\sum_{x} \mathbb{P}[X = x] \right) \log \frac{1}{\mathbb{P}[Y = y]} \\ &= H(X) + H(Y) \end{aligned}$$

One can generalize the above for n independent random variables in the following way

$$H(X_1, \ldots, X_n) = H(X_1) + \cdots + H(X_n).$$

12.2 Data Compression and Codes

To understand why we care about entropy and how it relates to data compression, let us consider the following problem: Suppose we have a long segment of DNA that looks like AGCCATTAC...CCGTA. How many bits do we need to represent it? We can encode $A = 00$, $G = 01$, $C = 10$, $T = 11$ which uses 2 bits per character. These are called **codewords**.

But if we knew something about the statistics of the sequence, we can do better. For example, we can use the following encoding which assigns shorter codes to more frequent symbols.

character	probability	codeword	encoding length
A	1/2	0	1
G	1/4	10	2
C	1/8	110	3
T	1/8	111	3

The above is a **prefix-free code**, i.e., no codeword is a prefix of another codeword. These are very easy to decode by reading the sequence of bits from left to right. For example, 01011011100 corresponds to the sequence AGCCA.

To see that the above encoding is indeed more efficient than the trivial encoding which takes 2 bits per symbol, let us compute the average code length $\bar{\ell}$ of the above encoding, i.e., the expected number of bits we need to encode each character.

$$\bar{\ell} = \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot 3 = 1 + \frac{3}{4} = \frac{7}{4}$$

which is better than 2 bits per character.

Let us also compute the entropy of the above distribution

$$\begin{aligned} H(X) &= \frac{1}{2}\log 2 + \frac{1}{4}\log 4 + \frac{1}{8}\log 8 + \frac{1}{8}\log 8 \\ &= \frac{1}{2}\cdot 1 + \frac{1}{4}\cdot 2 + \frac{1}{8}\cdot 3 + \frac{1}{8}\cdot 3 = \frac{7}{4}. \end{aligned}$$

The entropy of this distribution equals the average code length! As we will see later, this is not a coincidence.

One can wonder: can we get a smaller code? For example, consider the following encoding which uses shorter codewords compared to the previous one:

character	probability	codeword	length
A	1/2	0	1
G	1/4	1	1
C	1/8	01	2
T	1/8	11	2

The expected length of this encoding is

$$\frac{1}{2}\cdot 1 + \frac{1}{4}\cdot 1 + \frac{1}{8}\cdot 2 + \frac{1}{8}\cdot 2 = \frac{5}{4}$$

which is a lot better than the previous scheme.

But there is a catch! We do not want to encode just one sumbol but sequences of symbols. This code is not good for sequences as information is lost. For example, the sequences AGT and CGG are both encoded to 0111. A code where every sequence is mapped to a distinct bit representation is called a **uniquely decodable** code. Every prefix-free code is uniquely decodable using the algorithm above.

12.3 Huffman Coding

Suppose we have probability distribution over characters. Such a distribution is called a source. For example, consider the same source as before:

character	probability
A	1/2
G	1/4
C	1/8
T	1/8

How can we compress a long sequence of characters sampled from the source? One can also think of this as a transmission problem: Alice has the source. She samples a long sequence from it and wants to send Bob a message to convey the sequence while sending as few bits as possible. We will assume for simplicity that the probability of each character is a power of 2, i.e., of the form 2^{-k} for some integer $k \geq 0$, but we remark that the algorithm we are going to see will work more generally.

The following algorithm due to Huffman generates a prefix-free code for any source:

- Create a tree with the leaves as the characters. Label the nodes by probabilities.
- Take the two smallest probabilities and add a parent with label the sum of the two probabilities of its children.
- Repeat until we are left with a node with probability 1. This will be the root.
- Give the two edges from each node a 0/1 label.

The codeword for a symbol corresponds to the 0/1 labels from root to the leaf corresponding to that symbol. This code is always prefix-free!

The following figure shows the Huffman code for the above example source.

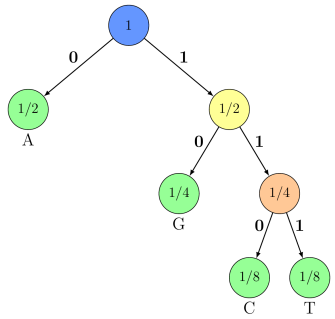

Figure 3. Huffman tree for the example source assigns $A \to 0, G \to 10, C \to 110, T \to 111$.

Let us compute the expected length of the code. To see this, we notice two things:

1. If every probability is a power of two, then there must be at least two elements with the minimum probability, if minimum probability is strictly smaller than one. The easiest way to see this is to write the probabilities in binary:

$$1 = 1.000000$$
$$1/2 = 0.100000$$
$$1/4 = 0.010000$$
$$1/8 = 0.001000 \text{ and so on.}$$

   If there is only one element with minimum probability that is at most one, then when we add up all the probabilities, the total probability will not be 1 as the least significant bit will remain 1.

2. The above means that when we create a new node by adding the nodes with the minimum probability as children, the new labels for the merged node will still be a power of two.

The number of times we merge on the way from a leaf node until we hit the root is exactly k if the probability of the symbol x that corresponds to the leaf node is 2^{-k}. In other words, $k = \log \frac{1}{\mathbb{P}[X=x]}$. Therefore, the expected length of the Huffman code is exactly:

$$\sum_x \mathbb{P}[X = x] \cdot \log \frac{1}{\mathbb{P}[X = x]},$$

which is the entropy of X. If the probabilities are not a power of two, then the length of Huffman code is at most $H(X) + 1$, due to rounding effects. It turns out that this is almost optimal as we will see in the next section.

12.4 Shannon's Source Coding Theorem

A seminal result in information theory is Shannon's Source Coding Theorem. One part of the theorem proves the following result.

Theorem. *For any uniquely decodable code, the expected length is at least $H(X)$.*

From the above it follows that Huffman coding is optimal up to the 1 bit additive factor. This 1 bit additive factor may not look significant but if one has to encode billions of characters this starts to matter. For example, suppose X is the source $\{H, T\}$ where $\mathbb{P}[X = H] = 1/4$ and $\mathbb{P}[X = T] = 3/4$. The Huffman code needs 1 bit. But the entropy is:

$$\frac{1}{4} \log 4 + \frac{3}{4} \log \frac{4}{3} = 0.811 \text{ bits/symbol}$$

This is 20% smaller and starts to add up for long sequences. How do we get around this? Recall that we are interested in encoding sequences rather than individual characters. Consider pairs of characters, e.g., $\{HH, HT, TH, TT\}$ as a source:

	HH	HT	TH	TT
Probabilities	1/16	3/16	3/16	9/16

The Huffman codeword for each pair above has length 1.69 bits per two symbols. So, per symbol we need 0.845 bits which is closer to the entropy.

In fact, pushing this idea further, we can encode a block of n symbols together and it turns out that we can get as close to entropy as we want. Let us formalize this: Suppose we have $X_1, \ldots, X_n$ which are all identically distributed as X and independent.

Therefore, from the properties of entropy

$$H(X_1, \ldots, X_n) = \sum_{i=1}^{n} H(X_i) = nH(X)$$

We generate a Huffman code for the block of n symbols $X_1, \ldots, X_n$ by treating it as a super symbol. The expected length of encoding this super symbol satisfies

$$H(X_1, \ldots, X_n) \leq \mathbb{E}[\text{length}] \leq H(X_1, \ldots, X_n) + 1$$

So, the expected number of bits per symbol is

$$\frac{H(X_1,\ldots,X_n)}{n} \leq \frac{\mathbb{E}[\text{length}]}{n} \leq \frac{H(X_1,\ldots,X_n)}{n} + \frac{1}{n}$$

Therefore, we obtain

$$H(X) \leq \frac{\mathbb{E}[\text{length}]}{n} \leq H(X) + \frac{1}{n}$$

As $n \to \infty$, the expected length per symbol goes to $H(X)$.

Thus, entropy of X is the minimum number of bits per source symbol on average necessary to encode a sequence of independent and identically distributed symbols from the source. This is also a part of Shannon's source coding theorem.