LECTURE 12 (October 7^{th})

Entropy & Data Compression

Surprisal 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.

Less likely events are more informative because they are more surprising.

Let's try to define a function S which captures the amount of surprise in an event A.

What are the properties such a function should satisfy?

$\boxed{1}$ S should be a function of the probability of the event. So, we can write $S(p)$ where $p \in [0,1]$ is $\mathbb{P}[A]$

$\boxed{2}$ $S(p)$ should decrease as p increases since more likely events are less surprising

$\boxed{3}$ $S(p)$ is a continuous function of p since we don't expect the surprise value to suddenly jump

$\boxed{4}$ For two events A, B, total surprise should be sum of individual surprises:

$$S(\mathbb{P}(A,B)) = S(\mathbb{P}(A)) + S(\mathbb{P}(B))$$

$$\text{OR} \qquad S(p_1 p_2) = S(p_1) + S(p_2) \qquad \text{for } p_1, p_2 \in [0,1]$$

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

Remark Technically the base of the log can be chosen arbitrarily but we will choose it to be 2 and measure the surprisal in bits

Entropy

Suppose we have a random variable X

surprise associated with the event $X=x$ is

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

Entropy of X is the average surprise on learning the value of X

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

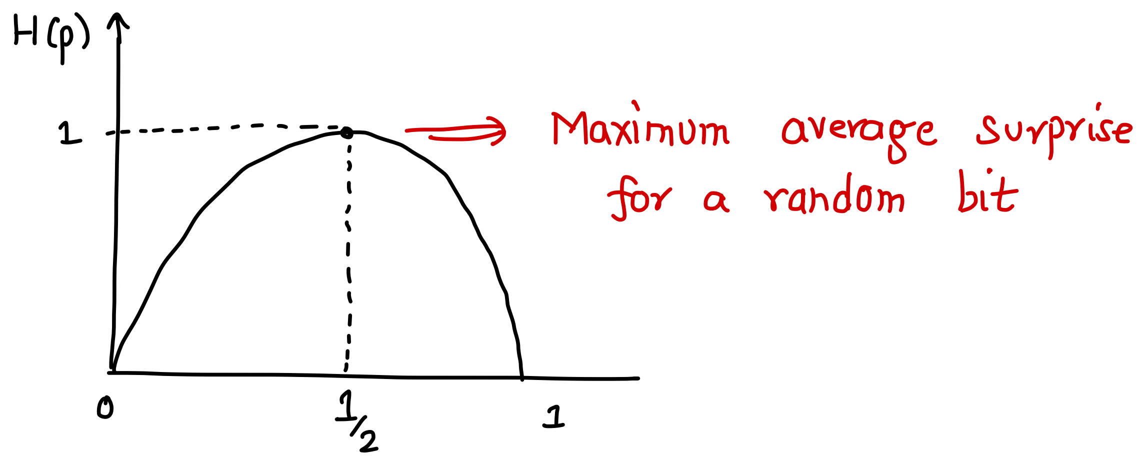
Basic properties of Entropy

$\boxed{1}$ $H(X) \geq 0$ and $H(X) = 0$ iff X is deterministic

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

Then, $H(X) = p \cdot \log \frac{1}{p} + (1-p) \log \frac{1}{1-p}$ where $p \in [0,1]$

This is called the binary entropy function $H(p)$ and looks like



$\boxed{2}$ Suppose X is uniform over $\{1, \dots m\}$, i.e. $\mathbb{P}[X=i] = \frac{1}{m}$ $\forall i$

Then,

$$H(X) = \sum_{i=1}^{m} \frac{1}{m} \log m = \log m$$

FACT For any random variable over $\{1, \dots m\}$

$$H(X) \leq \log m$$

So uniform distribution over $\{1, \dots m\}$ has the largest entropy.

3 **Joint Entropy** Given two random variables X & Y their joint entropy

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

E.g. if X is uniform over $\{1, \dots m\}$ and Y is uniform over $\{1, \dots n\}$

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

So, $H(X,Y) = \sum_{x,y} \frac{1}{mn} \log mn = \log mn$

Observe that $H(X) = \log m$ & $H(Y) = \log n$

$$H(XY) = H(X) + H(Y)$$

$\boxed{\text{FACT}}$ for any pair of random variables

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

↳ Equality holds if X & Y are independent

If X & Y are independent,

$$\mathbb{P}[X=x, Y=y] = \mathbb{P}[X=x] \cdot \mathbb{P}[Y=y]$$

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

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

$$= \sum_{x} \mathbb{P}[X=x] \Big(\overset{1}{\cancel{\sum_y \mathbb{P}[Y=y]}} \Big) \log \frac{1}{\mathbb{P}[X=x]}$$

$$+ \sum_{y} \mathbb{P}[Y=y] \cdot \Big(\overset{1}{\cancel{\sum_x \mathbb{P}[X=x]}} \Big) \log \frac{1}{\mathbb{P}[Y=y]}$$

$$= H(X) + H(Y)$$

In general, for n independent random variables

$$H(X_1, \dots X_n) = H(X_1) + \dots + H(X_n)$$

Prefix-free Encoding

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 $A = \underbrace{00}$, $G = \underbrace{01}$, $C = \underbrace{10}$, $T = \underbrace{11}$ 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,

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

So, we assigned shorter codes to more frequent symbols
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 left to right.

For example, 0101101100 corresponds to the sequence ________

What is the average code length, i.e. the expected number of bits we need to encode each character?

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

$$= 1 + \frac{3}{4} = \frac{7}{4}$$ which is better than 2 bits per character

Let's also compute the entropy of the distribution

$$H(X) = \frac{1}{2}\cdot \log_2 2 + \frac{1}{4}\log_2 4 + \frac{1}{8}\log_2 8 + \frac{1}{8}\log_2 8$$

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

This is not a coincidence as we will see in a second.

One can wonder can we get a smaller code. For example, consider

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

$$\mathbb{E}[\text{length}] = \frac{1}{2}\cdot 1 + \frac{1}{4}\cdot 1 + \frac{1}{8}\cdot 2 + \frac{1}{8}\cdot 2 = \frac{5}{4}$$

which is a lot better than the previous scheme.

But the catch is that this code is not good for sequences as information is lost

$$AGT \longrightarrow 0111$$

$$CGG \longrightarrow 0111$$

A code is called uniquely decodable if every sequence is mapped to a distinct bit representation

Every prefix-free code is uniquely decodable using the algorithm above.

Huffman Coding

Suppose we have a source that outputs a character from a probability distribution. For example,

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

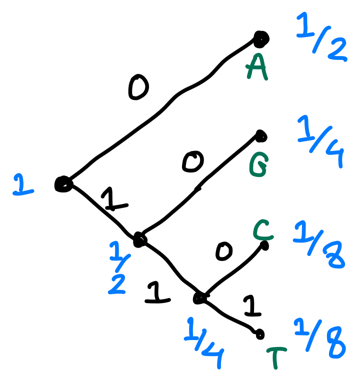
How can we compress a long sequence of characters sampled from the source ?

One can also think of it 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 $k \geq 0$.

Note: This is just for simplicity. The algorithm will work more generally.

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



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

- 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 that symbol. For the above,

$$
\begin{aligned}
A &\to 0 \\
G &\to 10 \\
C &\to 110 \\
T &\to 111
\end{aligned}
$$

This code is always prefix-free [Why?]

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

$\boxed{1}$ If every probability is a power of two, then there must be at least two elements with the minimum probability (if minimum probability <1)

The easiest way to see this is to write the probabilities in binary

$$
\begin{aligned}
1 &= 1.000000 \\
1/2 &= 0.100000 \\
1/4 &= 0.010000 \\
1/8 &= 0.001000
\end{aligned}
$$

If there is only one element with minimum probability (<1), then when we add them up, we can not get 1 as the least significant bit will remain 1.

$\boxed{2}$ 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 there will be merge from a leaf node until we hit the root is exactly k if $\mathbb{P}[\text{leaf node}] = 2^{-k}$. Note that here

$$k = \log_2 \frac{1}{\mathbb{P}[\text{leaf node}]}$$

Therefore, the expected length of "Huffman" code is exactly

$$\sum_x \mathbb{P}[X=x] \cdot \log \frac{1}{\mathbb{P}[X=x]} = H(X) = \text{entropy of } X$$

If the probabilities are not a power of two, then the length is atmost $H(X)+1$ (due to rounding effects)

It turns out that this is almost optimal.

> **Shannon's Source Coding Theorem**
>
> For any uniquely decodable code, expected length is $\geqslant H(X)$.

So, Huffman coding is almost optimal upto the 1 bit additive factor. That 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 $\mathbb{P}[X=H] = 1/4$
$\mathbb{P}[X=T] = 3/4$

The Huffman code needs 1 bit, But the entropy is

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

This is 20% smaller and adds up for long sequences

How to get around this?

We are interested in encoding sequences rather than individual characters. Consider codewords for pairs of characters, e.g.

$$\{HH, HT, TH, TT\}$$

Probabilities $\frac{1}{16}$ $\frac{3}{16}$ $\frac{3}{16}$ $\frac{1}{16}$

Huffman codeword for each pair has length

1.69 bits per two symbols

So, per symbol we need 0.845 bits which is closer to the entropy.

In fact, we can encode a block of n- symbols together and get as close to entropy as we want.

Let's see a proof. Suppose we have $X_1 \ldots X_n$ identically distributed as X and independent.

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

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

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

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

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.

Note: This is also a part of Shannon's source coding theorem.