

Information and Communication Course

24 May 2021

- Introductory stuff (bs)

Communication Theory

- Real world communication systems
- Structured (rigorous) mathematics

Math to system -> Synthesis

System to math -> Analysis (Needs rigour)

Basically you need math and intuition to make something constructive and interface with the real world.

Signals: A function that conveys information about a phenomenon. Any function that transmits information (useful, hence information) is a signal. Information can be considered as something we are “interested” in.

Signals **NEED NOT** be time dependent alone. Example:

- RGB values of a certain point if we consider colours to be time invariant.
 - A video is a combination of space and time variance.
 - Text/Documents are not time variant too, but are signals as they convey information. It is a discrete signal as it can be quantised into specific characters (discrete elements) that can be picked out of a countable set. An example of the set can be the set of characters on a keyboard, which form a countable set. DO NOT consider sets like the natural numbers, as they can be broken down into individual digits and characters, and will still be discrete, as we can interpret them in any way we want (As Aftab sir would say, “They are nothing but symbols in sand”).
-

26 May 2021

Communication: Can be broken into three steps:

- Transmission
- Reception
- Reconstruction of received information

Point to point communication system

Source -> Transmitter -> Channel -> Receiver -> Reconstruction/Processing

The channel is one part of this system that is not completely under our control and we will have to account for it in our design as it will be imperfect.

We may assume the channel as an ideal channel (No disturbance/noise) for analysis.

Transmitter side

- Source (Sending information)
- Transmitter (Modulation, amplification, and channel coding (For example, repeating the signal to minimise loss of information) to make the information appropriate for travelling in the channel)
- Transmission of data

Receiver end

- Received signal
- Detect, Estimate, demodulate and decode the transmitted signal (Because of noise in the channel, the signal will not be identical to the one generated at the transmitter)
- Extract information
- An estimate of the information that was transmitted (Because we don't know EXACTLY what was transmitted)

What does the end user want from the communication system?

- Low probability of error, High fidelity
- High rate of communication

In digital communication, these two happen via channel coding and modulation. In analog communication, this can be done via intelligent modulation and amplification.

- High range -> This is captured in the channel (Already done because of the choice of channel)
- Low latency in communication (Time delay between transmission and decoding)

Other communication systems

- Point to point
- Single transmitter to multiple receivers
- One transmitter, multiple receivers (Like a broadcast)
- Multiple transmitters, one receiver (Multiple phones connecting to one tower)

28 May 2021

- What is information? (Handwavy answers about usefulness)
- Information of any outcome has to do with the uncertainty of the outcome.
- Example : Say that $X \in \{0, 1\}$ & can take value 0 with probability p , while 1 has a probability of $1 - p$. X can be any quantity that represents a choice we care about. X is technically called a random variable. If probability of a certain event occurring is low or the occurrence of that event is highly unlikely, then X denoting the occurrence of that event is said to have large information content.

Information content in a particular event $\propto \frac{1}{\text{Probability of the event}}$

For two independent events with Probabilities P_1 and P_2 , we multiply them to find the probability of both occurring together, while the net information content of both at the same time is the algebraic sum of their individual information contents. So, if we define X , we can say,

The Information content in an event $X = x$ (where x is the event and $X = x$ denotes it happening, X being a random variable) is

Information content in an event $X = \log \frac{1}{P(X=x)}$

So the average information content (or the “expected information content”) would be,

$$H(X) = \sum_{i=1}^n P(X = x_i) \cdot \log \frac{1}{P(X = x_i)}$$

This expression is defined as the *ENTROPY* in random variable X , denoted by $H(X)$.

Lemma \rightarrow Suppose $X_1 \in x_1$ and $X_2 \in x_2$ are independent random variables, then

$$H(X_1, X_2) = H(X_1) + H(X_2)$$

Where, $H(X_1, X_2)$ is the joint entropy of X_1 and X_2 . We will be using \log_2 as default in the course.

31 May 2021

Proof for $H(X_1, X_2) = H(X_1) + H(X_2)$ where X_1, X_2 are **independent random variables**. Independent random variables can be defined as two variables that are always independent for all events.

$$H(X_1, X_2) = \sum_{x_1 \in \mathcal{X}_\infty, x_2 \in \mathcal{X}_\epsilon} P(X_1 = x_1, X_2 = x_2) \cdot \log(1/P(X_1 = x_1, X_2 = x_2))$$

by definition. Now, we can split that summation into a double summation (Like a nested for loop, one over X_1 and one over X_2) and we can split the log term too. We can split them into two separate terms and sum them over their respective summations. After manipulation and using total probability theorem, that gives us

$$= \sum_{x_1 \in \mathcal{X}_1} \log\left(\frac{1}{P(X_1 = x_1)}\right) \left[\sum_{x_2 \in \mathcal{X}_2} P(X_1 = x_1, X_2 = x_2) \right] \\ + \sum_{x_2 \in \mathcal{X}_2} \log\left(\frac{1}{P(X_2 = x_2)}\right) \left[\sum_{x_1 \in \mathcal{X}_1} P(X_1 = x_1, X_2 = x_2) \right]$$

After this, we can just use total probability theorem on the inner summation, and we have our result.

Now, if X_1, X_2 are NOT independent, we define a probability measure called conditional probability.

$$P(X_2 = x_2 | X_1 = x_1) = P(X_2 = x_2, X_1 = x_1) / P(X_1 = x_1) \text{ where } P(X_1 = x_1) \neq 0$$

Note that $P(X_2 = x_2 | X_1 = x_1) = P(X_2 = x_2)$ for independent events.

So, we can say that

$$\sum_{x_2 \in \mathcal{X}_\epsilon} P(X_2 = x_2 | X_1 = x_1) = 1$$

So, conditional probability can be considered a probability distribution in itself.

So conditional entropy could be defined as,

$$H(X_2 | X_1) = \sum_{x_1 \in \mathcal{X}_\infty} P(X_1 = x_1) \cdot H(X_2 | X_1 = x_1)$$

while, $H(X_2 | X_1 = x_1)$ is defined as,

$$H(X_2 | X_1 = x_1) = \sum_{x_2 \in \mathcal{X}_\epsilon} P(X_2 = x_2 | X_1 = x_1) \cdot \log(1/P(X_2 = x_2 | X_1 = x_1))$$

Intuitively makes sense.

A more general equation now would be,

$H(X_2, X_1) = H(X_1) + H(X_2|X_1)$ or shuffle the order accordingly, shouldn't make a difference. Oh lol confirmed in the next class.

2 June 2021

If f is a function, then $\text{Support}(f)$ or $\text{sup}(f)$ is all the inputs in its domain that give a non zero output.

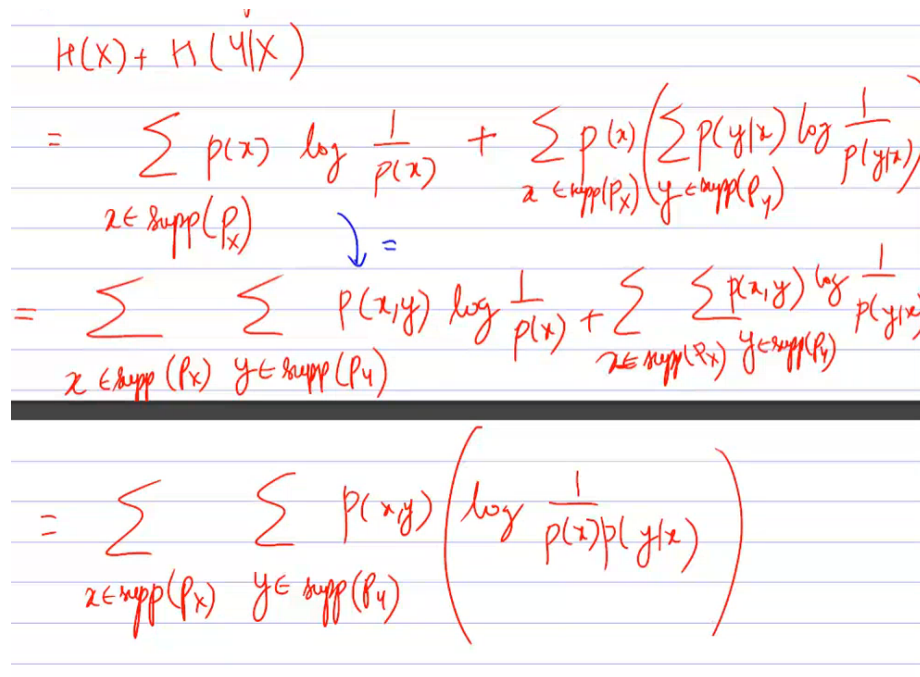
Proof for $H(X, Y) = H(X) + H(Y|X)$

$$H(X|Y) = \sum_{y \in \text{sup}(P_Y)} P_Y(y) \cdot H(X|Y = y)$$

where

$$H(X|Y = y) = \sum_{x \in \text{sup}(P_X)} P(x|y) \cdot \log(1/P(x|y))$$

Now, if we expand the RHS, in the main eqn, we get



$$\begin{aligned}
 & H(X) + H(Y|X) \\
 &= \sum_{x \in \text{sup}(P_X)} p(x) \log \frac{1}{p(x)} + \sum_{x \in \text{sup}(P_X)} p(x) \left(\sum_{y \in \text{sup}(P_Y)} p(y|x) \log \frac{1}{p(y|x)} \right) \\
 & \quad \downarrow = \\
 &= \sum_{x \in \text{sup}(P_X)} \sum_{y \in \text{sup}(P_Y)} p(x,y) \log \frac{1}{p(x,y)} + \sum_{x \in \text{sup}(P_X)} \sum_{y \in \text{sup}(P_Y)} p(x,y) \log \frac{1}{p(y|x)} \\
 &= \sum_{x \in \text{sup}(P_X)} \sum_{y \in \text{sup}(P_Y)} p(x,y) \left(\log \frac{1}{p(x)p(y|x)} \right)
 \end{aligned}$$

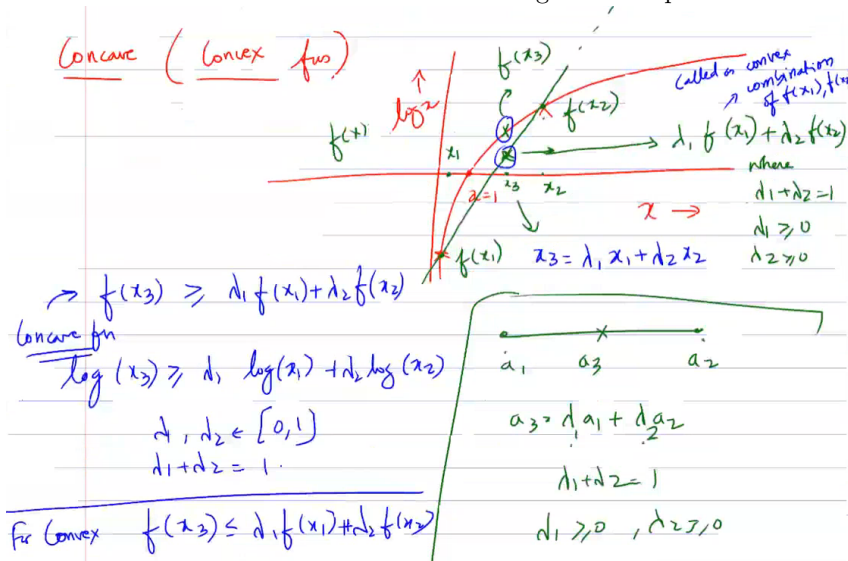
This is what we need, as the final expression is $H(X, Y)$.

Some properties of entropy

Can we bind $H(X)$ from above and below?

Claim : $0 \leq H(X) \leq \log|\mathcal{X}|$

- Proof for $H(X) \geq 0$ Trivial, as $P(x) > 0 \forall x \in \text{sup}(P_x)$ Exactly 0 when $|\text{sup}(P_x)| = 1$.
- Proof for $H(X) \leq \log|\mathcal{X}|$ As we know, \log is a concave function. So, $\log(\lambda_1 a + \lambda_2 b) \geq \lambda_1 \log(a) + \lambda_2 \log(b)$. This is **Jensen's inequality** in short. $\lambda_1 + \lambda_2 = 1, \text{both} \geq 0$. This combination is called a convex combination. It's the reverse for a convex function. Image below explains it in detail.



Now, we can write $H(X)$ as

$$H(X) = \sum_{x \in \text{sup}(P_x)} P(x) \log(1/P(x))$$

This resembles a convex combination. So we can say that,

$$H(X) \leq \log\left(\sum_{x \in \text{sup}(P_x)} P(x) \cdot (1/P(x))\right)$$

Which gives us $H(X) \leq \log(|\text{sup}(P_x)|)$ which also implies,

$$H(X) \leq \log|\mathcal{X}| \text{ Hence, proved.}$$

4 June 2021

Jensen's inequality is satisfied with the equality condition when the function is a straight line.

For strictly concave functions like logarithm,

$$f(\lambda_1 x_1 + \lambda_2 x_2) = \lambda_1 f(x_1) + \lambda_2 f(x_2)$$

when $x_1 = x_2$.

In general, if $\lambda_i \neq 0$ and $\sum_{i=1}^n \lambda_i = 1$ and Jensen's holds with equality, then

$$x_1 = x_2 \dots = x_n$$

Now, if we apply this condition to $H(X) \leq \log|\mathcal{X}|$,

$$\sum_{x \in \text{supp}(P_X)} \lambda_x \log(1/P(x)) \leq \log|\text{supp}(P_X)|$$

then for equality, by above claim, we have

$$\frac{1}{P(x)} = \text{Constant}$$

But we also know,

$$\sum_{x \in \text{supp}(P_X)} P(x) = 1.$$

So, we can infer that

$$P(x) = \frac{1}{|\text{supp}(P_X)|} \forall x \in \text{supp}(P_X)$$

This distribution is called as a uniform distribution.

Lemma: So, $H(X) = \log|\mathcal{X}|$ iff P_X is uniform and $|\text{supp}(P_X)| = |\mathcal{X}|$. This is a necessary and sufficient condition.

Relative Entropy (or) Information Divergence (or) Kullback-Leibler Divergence

Suppose there is a random variable X that has two different probability distributions p_X and q_X , then the Relative Entropy/Information Divergence/K-L Divergence can be defined as

$$D(p_X || q_X) \triangleq \sum_{x \in \text{supp}(p_X)} p(x) \log(p(x)/q(x))$$

This expression clearly depends on the order of the arguments.

Divergence can be **intuitively** thought of as a distance measure between different probability distributions.

Claim: $D(p||q) \geq 0$

Proof:

We first manipulate the expression to use Jensen's inequality.

$$D(p||q) = - \sum_{x \in \text{supp}(p_X)} p(x) \log(q(x)/p(x))$$

Now, we can say that

$$- \sum_{x \in \text{supp}(p_X)} p(x) \log(q(x)/p(x)) \geq -\log\left(\sum_{x \in \text{supp}(p_X)} q(x)\right)$$

using Jensen's inequality. Now, we can say that $\sum_{x \in \text{supp}(p_X)} q(x) \leq 1$, and hence, the minimum value the RHS can take is zero. So,

$$- \sum_{x \in \text{supp}(p_X)} p(x) \log(q(x)/p(x)) \geq 0$$

Hence, proved.

When we apply equality condition for Jensen's, we get

$$\frac{p(x)}{q(x)} = \text{Constant} \implies p(x) = C \cdot q(x) \quad \forall x \in \text{supp}(p_X)$$

Now, if we take summation on both sides, we get 1 on the LHS and $\sum_{x \in \text{supp}(p_X)} C \cdot q(x)$

Now, as $p(x)/q(x) = \text{Constant} \quad \forall x \in \text{supp}(p_X)$, we can say that $|\text{supp}(q_X)| \geq |\text{supp}(p_X)|$. If $|\text{supp}(q_X)| > |\text{supp}(p_X)|$, then C will have to be greater than 1, and this is impossible as

$$\text{WRONG } C = \left(\sum_{x \in \text{supp}(p_X)} p(x) / \sum_{x \in \text{supp}(p_X)} q(x) \right) \implies C = 1 / \sum_{x \in \text{supp}(p_X)} q(x).$$

But the max value that $\sum_{x \in \text{supp}(p_X)} q(x)$ can take is 1.

So, this is a contradiction, and hence, $|\text{supp}(q_X)| = |\text{supp}(p_X)|$

So, from this, we can say that $C = 1$ and hence, the two probability distributions are equal for divergence equal to 0.

$\therefore D(p||q) = 0$ iff $p_X = q_X$.

Going back to conditional entropy,

$$\text{Claim: } 0 \leq H(X|Y) \leq H(X)$$

For $H(X|Y) \geq 0$, we can consider the conditional probability to be another distribution for X, and hence, the proof is the same as it was for $H(X)$.

For the upper bound,

$$H(X) - H(X|Y)$$

We can write this as,

$$\begin{aligned} & \sum_{x \in \text{supp}(p_X), y \in \text{supp}(p_Y)} p(x, y) \log(1/p(x)) - \sum_{x \in \text{supp}(p_X), y \in \text{supp}(p_Y)} p(x, y) \log(1/p(x|y)) \\ &= \sum_{x \in \text{supp}(p_X), y \in \text{supp}(p_Y)} p(x, y) \log(p(x|y)/p(x)) \end{aligned}$$

On expanding the conditional probability term, we get,

$$= \sum_{x \in \text{supp}(p_X), y \in \text{supp}(p_Y)} p(x, y) \log(p(x, y)/(p(x) \cdot p(y)))$$

Here, both the terms in division in the log term can be proven to be valid joint probability distributions.

So, we can now say that

$$H(X) - H(X|Y) = D(p(x, y) || p_X(x) \cdot p_Y(y))$$

which is always greater than or equal to zero.

Hence, we have proved that $H(X|Y) \leq H(X)$.

7 June 2021

No class today =P

9 June 2021

Joint entropy in two variables,

$$H(X, Y) = \sum_{x, y \in \text{supp}(P_{X, Y})} P(x, y) \log(1/P(x, y))$$

$P_{X, Y} : \mathcal{X} * \mathcal{Y} \rightarrow [0, 1]$, indicating CARTESIAN PRODUCT, including the set of all possible ordered pairs, forming a distribution.

This can be extended to n variables easily,

$$H(X_1, \dots, X_n) = \sum_{x_1, \dots, x_n \in \text{supp}(P_{X_1, \dots, X_n})} P(x_1, \dots, x_n) \log(1/P(x_1, \dots, x_n))$$

If $X = Y$,

$H(X, Y) = H(X) = H(Y)$, this is complete dependence.

If they are independent on the other hand, we get

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

Chain rule of Joint entropy

$$H(X_1, \dots, X_n) = H(X_1) + \sum_{i=2}^n H(X_i | X_1, \dots, X_{i-1})$$

where,

$$H(X, Y | V, U) = \sum_{u, v \in \text{sup}(P_{U, V})} P_{U, V}(u, v) H(X, Y | U = u, V = v)$$

where,

$$H(X, Y | U = u, V = v) = \sum_{x, y \in \text{sup}(P_{X, Y | U=u, V=v})} P(x, y | u, v) \log(1/P(x, y | u, v))$$

Now, this can be extended to any number of variables, before and after the conditioning.

Note:

$$P(x_1, x_2, x_3 | y_1, y_2) = P(x_1, x_2, x_3, y_1, y_2) / P(y_1, y_2)$$

Some other way of writing it?

IMPORTANT:

$$P(x, y | z) = P(x | z) \cdot P(y | x, z)$$

Mainly because it forms its own probability distribution ig? This can be used to write the above expression in another way. It can be written as,

$$P(x_1, x_2, x_3 | y_1, y_2) = P(x_1 | y_1, y_2) \cdot P(x_2 | x_1, y_1, y_2) \cdot P(x_3 | x_2, x_1, y_1, y_2)$$

or,

$$P(x_1, x_2, x_3 | y_1, y_2) = P(x_1, x_2 | y_1, y_2) \cdot P(x_3 | x_2, x_1, y_1, y_2)$$

or this beauty,

x_1, x_2, x_3 can be thought of as a random VECTOR here.

Proof for chain rule:

$$P(x_1, \dots, x_n) = P(x_1) \cdot P(x_2, x_3, \dots, x_n | x_1) = P(x_1) \cdot P(x_2 | x_1) \cdot P(x_3, \dots, x_n | x_1, x_2)$$

and keep splitting, same as chain rule in PRP. We finally get,

$$P(x_1, \dots, x_n) = P(x_1) \cdot \prod_{i=2}^n P(x_i | x_1, \dots, x_{i-1})$$

Use definition of joint entropy to finish entire proof.

$$\begin{aligned}
&= \frac{p(x_3, y_2 | y_1) p(x_1, x_2 | x_3, y_1, y_2)}{p(y_2 | y_1)} \\
&= \frac{p(x_3, y_2, x_1, x_2 | y_1)}{p(y_2 | y_1)} \\
&= p(x_1, x_2, x_3 | y_2, y_1)
\end{aligned}$$

Figure 1: amaze

11 June 2021

Entropy $H(X)$ \rightarrow Avg uncertainty about X. So,

$H(X|X)$ denotes uncertainty in X AFTER OBSERVING X, which is 0. Reduction in avg uncertainty of X achieved by observing X is $H(X) - H(X|X) = H(X)$. This is if P_X is known. Now, if we have two variables represented by their joint probability distribution, we can say that if a variable Y is observed before X,

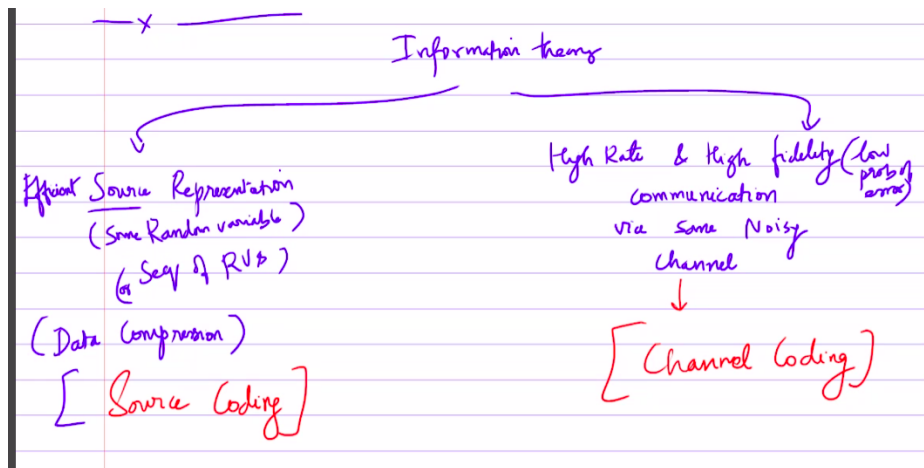
Remaining uncertainty in X is $H(X) - H(X|Y)$, because Y may give certain info about X which reduces its uncertainty.

This can be called “Information gained about X after observing Y” or, more technically “MUTUAL INFORMATION BETWEEN X AND Y” and is denoted by $I(X; Y) = H(X) - H(X|Y)$.

It can be shown that $H(X) - H(X|Y) = H(Y) - H(Y|X)$. This can be proven by taking the conditional terms to opposite sides and making them joint entropies, which are equal.

So, $I(X; Y) = I(Y; X)$. We can also say that $I(X; Y) \leq \min(H(X), H(Y))$, the proof being trivial.

$\rightarrow I(X; Y) \geq 0$ represent this using divergence/relative entropy.



Stuff that we WILL be doing.

Overview of Source Coding

Suppose we have $X \in \{a, b\}$ which is a binary source with probability distribution P_X . If the observer observes one instance of X , then wants to store/communicate it through a noise free medium, which can carry only $\{0, 1\}$ bits. We will only need one bit to convey this information with 2 choices.

If receiver knows $P_X(a) = 0, P_X(b) = 1$ then receiver need not even receive the signal and hence, we need 0 bits to convey this info. This information has no chance of error too, but kind of pointless. This is basically an example to show that the probability can decrease the number of required bits I guess?

Now, if we have a margin of error ϵ , then if any one of the events have a probability $\leq \epsilon$, then we can make do with no bits, as we can always declare the other event to be true (as it is within the bounds of error).

Basically, tolerating some amount of error, can give huge advantages.

14 June 2021

- Toleration of small errors can lead to us being able to compress the information better.
- Decoder is assumed to know the probability distribution of the source, so the decoder can interpret the data accordingly.
- We can club multiple random variable instances together, to get a lower probability of error. Individual distributions that we get from joint distributions are called marginal distributions. We can get joint distribution

from marginal distribution if the random variables are independent. The picture below shows this idea.

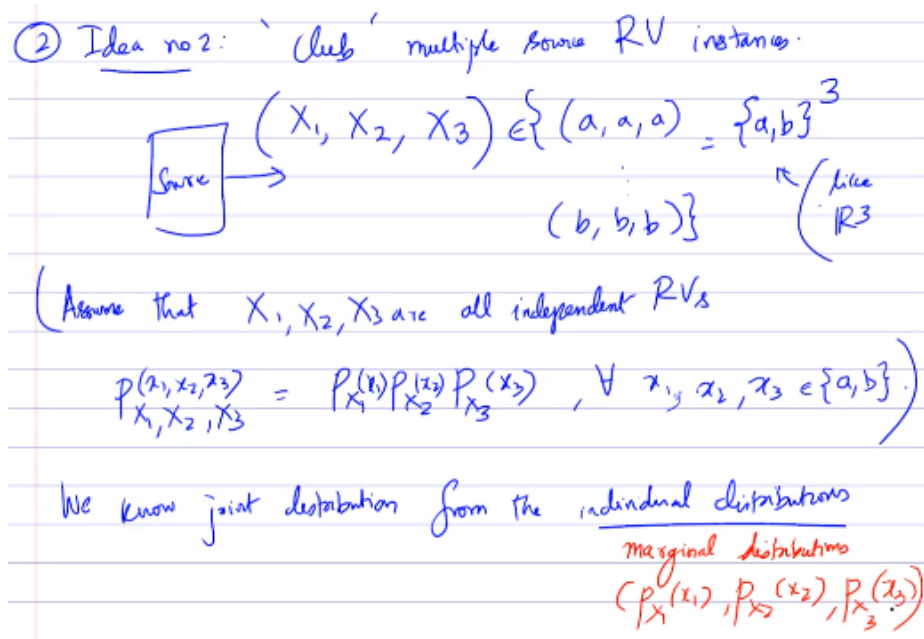


Figure 2: id2

We can code this accordingly, one such example being coding them with one bit and concatenating them together, which will give us the same amount of error as one bit at a time.

In the case of encoding one symbol, we only have two possible code lengths, which are 0 and 1. In this case however, we have 0, 1, 2 or 3 length strings.

A 1 length code could be very useful when a single tuple has a very high probability and the total probability of all other distributions is within our error limit.

In summary, this idea details the logic of combining multiple source symbols and encoding them together into some fixed length binary string (length chosen according to the distribution), which gives us a more efficient source code (smaller normalized length, where normalized length is the length of the compressed binary string divided by the length of the actual string).

Did an example to depict this idea. Only using fixed length binary strings here, though variable length seems to have a lot more potential in compression.

Example:

$$\text{Source } X \sim p_X \quad ; \quad \begin{aligned} p_X(a) &= p \\ p_X(b) &= 1-p \end{aligned}$$

Remember : ① We are allowing for encoding long source strings & we can tolerate some small prob of error.

② We have to a fixed length source code
(every 'n' length source string is to be encoded into a 'l' length binary vector/string/tuple .
Fixed length means l doesn't change with the source string.)

If a group of random variables have the same distribution and are independent, they are said to be independently and identically distributed. Like in the above case, the source string can be thought of as a string of random variables that have the same distribution and are independent.

Suppose we have a large length binary string, how many a's and b's do we expect to see in the random source string?

Number of a's $\rightarrow n \cdot p(a)$ Number of b's $\rightarrow n \cdot (1 - p(a))$ Total number of such strings $\rightarrow \binom{n}{n \cdot p(a)}$

For all such sequences, we will use unique strings. For EVERYTHING else, we will use ONLY ONE CODEWORD. This can increase our efficiency by A LOT, depending on our probability distribution.

16 June 2021

In last class we said,

Total number of such strings $\rightarrow \binom{n}{n \cdot p(a)}$

A better term for this would be "Typical sequences".

Atypical sequences are sequences where the distribution is very different from the expected distribution. We can conclude with high probability that any n-length sequence from the source is going to be a typical sequence.

As $n \rightarrow \infty$, the probability of atypical sequences tends to zero.

Block to block source coding

Assign a unique codeword of length l to each typical sequence.

Assign some “Same” codeword of length l , different from all other codewords that represents all atypical sequences.

We can now deduce that we will need

$$l = \log_2 \binom{n}{n \cdot (a)} + 1$$

to represent all these words. On simplifying the expression in the log (using massive amounts of approximation like in time complexity analysis) we get

$$l = n \log_2 n - np \log_2(np) - n(1-p) \log_2(n(1-p)) - C$$

where C is comparatively a very small quantity. On simplifying that expression, we get,

$$l = n[p \log(1/p) + (1-p) \log(1/(1-p)) - \text{smallnum}/n]$$

This is clearly the entropy of the random variable X (source string).

So, we get,

$$l = n \cdot H(X)$$

18 June 2021

Fixed to variable length source coding

If we have 4 symbols, $\{A, B, C, D\}$, and a code,

$$A \rightarrow 0 \quad B \rightarrow 1 \quad C \rightarrow 10 \quad D \rightarrow 11$$

Here, the source sequences ‘BA’ and ‘C’ are not uniquely decodable. This has a non zero probability of error which we are not okay with.

One solution is to use a PREFIX-FREE CODE.

Clarification in Terminology:

- Codeword: The string resulting from a particular input.
- Code: Set of all codewords.

Both refer to the mapping. Not the source signal or input.

Continuing...

A code C is called a Prefix free code if no Codeword in C is a prefix of another codeword, i.e., the example shown above shouldn't happen. In the example above, 1 is a prefix of the code words 11 and 10. So, it is not a prefix free code.

Example of a Prefix free code $\rightarrow \{10, 01, 11, 00\}$

Note: A variable length code does not mean that the length of the codewords are different. It just means that when we design the code, we are free to choose the length of the codewords, whereas in a fixed length code, all codewords must be of the same length.

Example of a variable length prefix free code $\rightarrow \{0, 10, 110, 1110\}$

If we think of the code as a binary tree

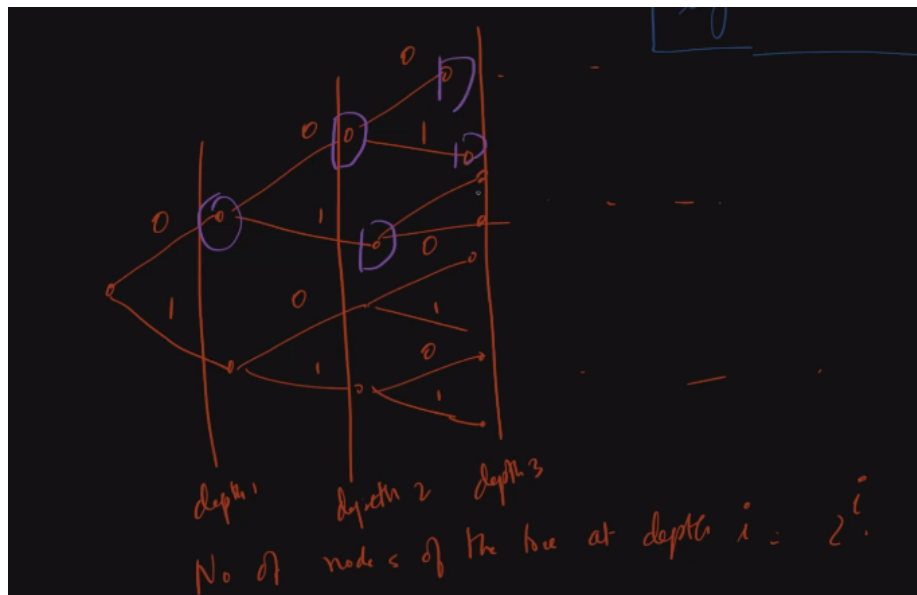


Figure 3: bt

If we represent the code like this, we can see that successor node of any codeword will not be part of a prefix free code, as the predecessor node is a prefix codeword. This is a necessary and sufficient condition. So, our code will be a set of nodes which are neither successors nor predecessors of each other.

More terminology:

- Leaves of the binary tree are nodes in the tree with no children.

If we have $X \in \mathcal{X}$, where X is the source string random variable, and we have $|supp(P_X)| = s$, then we can have l_i , where each represents the length of the binary code words associated to the i th symbol in $|supp(P_X)|$.

So, we can now say that the expected length of the codeword will be,

$$Expected\ length(\bar{L}) = \sum_{i=0}^{|supp(P_X)|} P_i \cdot l_i$$

Now, we want to find a code that minimises this expected length.

Did an example to show this formula in use.

The goal of Prefix free variable length source coding is to reduce expected length.

This is error free as EACH SEQUENCE of source symbols will have a UNIQUE codeword sequence. The decoder will start from the leftmost end of the codeword sequence, traverse the tree, and when it finds a sequence corresponding to a codeword, it declares the codewords' corresponding source symbol and restarts from the beginning. This will always work, as the code is prefix free.

The minimum possible \bar{L} among all codes is denoted as \bar{L}^* .

We will show that

$$H(X) + 1 \geq \bar{L}^* \geq H(X)$$

no matter what code we use.

21 June 2021

Some mad discussion about achievability and converse.

Channel coding

We give input, which is then translated and passed through a channel to give an output in a different language.

We say that a channel is noisy when we have a many one relation from input to output. The different one one relationships won't matter because we'll still have unique relations between input and output, which can be decoded by some decoder. Will be discussed later apparently.

If we have a many one relation, then we only choose one of the many words that map to one codeword, and we omit the rest from our vocabulary, which

is called the set of all transmittable sequences... Seems like a pretty scammy workaround?

This subset of transmittable sequences is called channel code, or just code. Denoted by \mathcal{C} . Note that $\mathcal{C} \subseteq \mathcal{X}^n$ where \mathcal{X}^n is the input vocabulary. Each vector in the input vocabulary requires the channel to be used n times, because n characters. Each vector in \mathcal{C} is a codeword.

Number of bits that is required to represent $|\mathcal{C}|$ codewords is $\log_2 |\mathcal{C}|$

Now, the rate of the code is defined as,

$$\text{Rate of } \mathcal{C} = \frac{\log_2 |\mathcal{C}|}{n}$$

Unit -> bits per channel use Higher the rate, more the chance of error, as there is more chance of a many one system forming.

Probabilistically noisy channel (or) Random channel

Same input can give many different outputs with different probabilities. We use conditional probability here.

23 June 2021

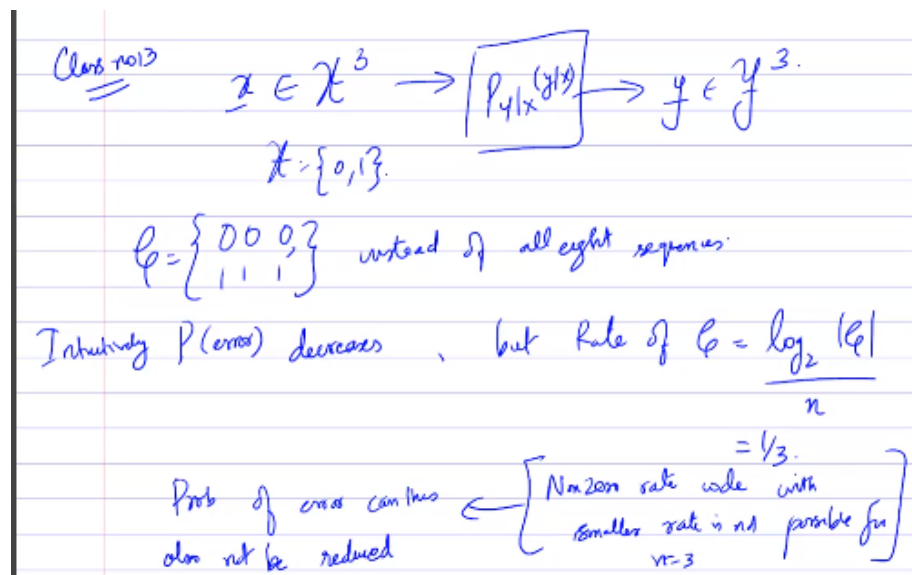


Figure 4: exr

Intuitively there is no way to reduce error in this case unless we change n . So if we increase n , then we get a code with a smaller rate, and hence intuitively lesser error.

So for negligible probability of error, intuition would lead us to a code with nearly zero rate.

Surprisingly, this is not the case in the ideal sense. Assuming we have freedom to choose n , we actually end up saying for ANY small $\epsilon > 0$, there exists a code \mathcal{C} with $P(\text{error}) < \epsilon$, and the rate of that code is,

$$R(\mathcal{C}) = \max_{P_X} (I(X; Y)) - f(\epsilon)$$

$f(\epsilon)$ here is negligibly small.

MAKES INTUITIVE SENSE AS THE INITIAL STATE IS UNCERTAINTY IN X AND THE CHANGE IN UNCERTAINTY AFTER OBSERVING Y IS $H(X) - H(X|Y)$ WHICH IS WHAT IS GIVEN IN THE FORMULA.

Now,

- $I(X; Y)$ depends on the conditional distribution we have for Y and X depending on the channel. Cannot be directly controlled by us.
- Also depends on distribution of P_X . P_Y can be derived from these two itself. As X is not a “natural source”, we can manipulate it, and hence we consider it to be controllable, and use it to maximise our requirements. So, we have

$$R(\mathcal{C}) = \max_{P_X} I(X; Y)$$

This is also called as the “**Channel capacity**” denoted by C .

Channel Coding theorem (Converse of the achievability discussed above)

No matter what we do, we CANNOT get a code with $\text{rate} > C$ and expect small probability of error.

To make the rate very close to C , we will have to take a very high value for the code length n .

Ex: Binary Symmetric Channel (or) bit flip channel Binary input and output.
 $\mathcal{X} = \{0, 1\} = \mathcal{Y}$

So, we can say

$$P_{Y|X}(y|x=0) = (p, \text{for } y=1) \text{ and } (1-p, \text{for } y=0)$$

$$P_{Y|X}(y|x=1) = (p, \text{for } y=0) \text{ and } (1-p, \text{for } y=1)$$

This is a symmetric distribution, hence the name “symmetric channel”. So, we can say that a bit flip for an input occurs with a probability p .

This implies $H(Y|X = 0) = H(Y|X = 1) = H_2(P)$, where $H_2(P)$ is that value. It attains a max value of 1 for a uniform distribution.

Pictorial representation:

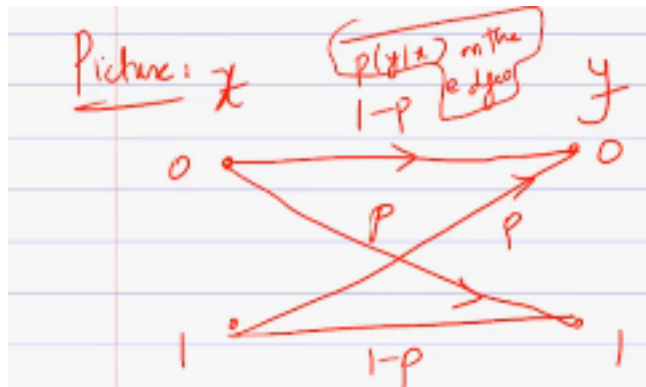


Figure 5: picr

We want to calculate capacity of this channel now. So,

$$I(X; Y) = H(Y) - H(Y|X)$$

But $H(Y|X) = H_2(P)$ and that is not influenced by choice of P_X , as the probability of bit flip is determined by the channel.

We also know that $\max_{P_X} H(Y) \leq 1$.

Here, $H(Y) = 1$ for a uniform distribution of X (can be proved). This gives us

$$R(\mathcal{C}) = \max_{P_X} (I(X; Y)) = 1 - H_2(p)$$

25 June 2021

We will give an argument for the converse for capacity of a binary symmetric channel.

Let \mathcal{C} be a code which has rate R and a low probability of error.

Now, we know that

$R = \log|\mathcal{C}|/n \implies |\mathcal{C}| = 2^{nR}$ If we transmit a codeword C with a large length n , and get an output y , then we can expect approximately np bits to be flipped in y . (Channel is independently acting on each input) But we don't know which positions are flipped. So, we can expect any sequence in $S(C)$, where $S(C)$ can be defined as

$$S(C) = \{y \in \{0,1\}^n | d_H(C, y) = np\}$$

d_H here is called the **Hamming Distance** which is the number of positions we need to flip in C to get y . We can assume a venn diagram like image of the codewords and radius as Hamming distance. By this, we can conclude that we must not have intersecting balls, as that would imply many one relations. So that would lead us to say,

$$|C| \leq \frac{2^n}{|S(C)|}$$

Now here, $\binom{2^n}{np}$ and we can substitute that in the expression. Now, to find rate

$\log|C| \leq n - \log(|S(C)|)$ The second term is approximately $nH_2(p)$ (From source coding). Now dividing both sides by N , we get rate less than or equal to capacity.

The proof in words,

- Code with small error
- Balls should not intersect because of that
- So we only have those a certain size of code, which determined the rate of the code.

Achievability will be done later, this is the converse.

History talk and overview of the stuff we will be doing in the course.

28 June 2021

Recap and some pretty cool history stuff.

Recall Source coding.

$Source \rightarrow X \rightarrow Source\ encoder \rightarrow Codewords(C)$

Assume that the codewords are binary strings. Codewords for each symbol in \mathcal{X} that has non zero probability.

Now, if the random variable X is distributed according to P_X and $C(x)$ be the codeword assigned to $x \in \mathcal{X}$.

$l(x)$ be the length of the codeword assigned to x . Fixed length to variable length source coding. Here, the fixed length of source sequence is 1 (Because each symbol has a codeword).

Set of codewords, $\mathcal{C} = \{C(x) | x \in \mathcal{X}\}$

Our goal is to design a PREFIX FREE code \mathcal{C} which has optimal expected length (that summation formula from before).

Lemma: $L^* \geq H(X)$

This implies any prefix free code for X has a length of at least $H(X)$.

Proof:

To prove this, we need the following claim.

Claim (Kraft's inequality): Let \mathcal{C} be any prefix free code. Then

$$\sum_{x \in \mathcal{X}} 2^{-l(x)} \leq 1$$

- Proof for claim: We know that any prefix free code can be represented using a binary tree which has the leaves as codewords.

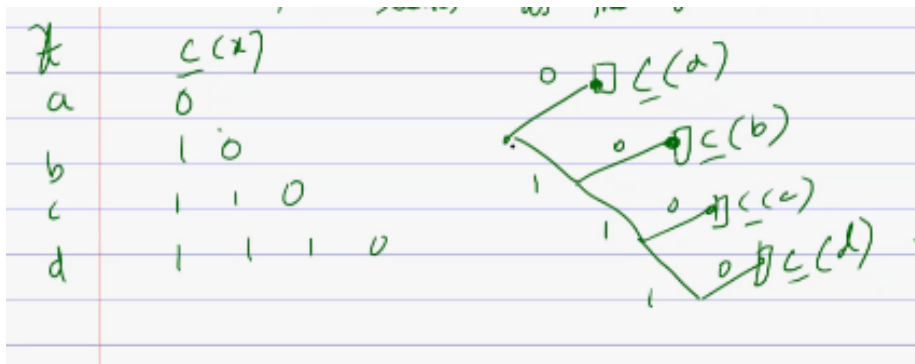


Figure 6: bt2

The depth of the binary tree corresponding to the code is the length of the largest codeword in that code ($l_{max} = \max(l(x))$).

Suppose there is a codeword with length l ($l \leq l_{max}$). If we look down from this point in the tree, there are $l_{max} - l$ levels below it. It will have $2^{l_{max} - l}$ successors in the complete binary tree, but none of them are codewords, as the code is prefix free. Idk how this helps, but there it is.

No two codewords IN A PREFIX FREE CODE, can have a common successor. So we can say,

$$\sum_{x \in \mathcal{X}} 2^{l_{max} - l} \leq 2^{l_{max}}$$

RHS is max number of codewords possible. Dividing both sides gives us the required result.

Now, back to the Lemma... We need to prove that,

$$L^* - H(X) \geq 0$$

$$\implies \sum_{x \in \mathcal{X}} p(x) l(x) - \sum_{x \in \mathcal{X}} p(x) \log\left(\frac{1}{p(x)}\right) \geq 0$$

If we think of $D(p_X || q_X) \geq 0$ We can define $q_X(x)$ as,

Handwritten definition of $q_X(x)$ with annotations:

Let $q_X(x) \triangleq \left(\frac{2^{-l(x)}}{\sum_{x \in \mathcal{X}} 2^{-l(x)}} \right) (\geq 0)$

Annotations:

- A box on the left says: "Hence this is a valid probability!"
- A note above the denominator says: " $x \in \text{supp}(p_X)$ "
- A circled equation on the right says: $\sum_{x \in \mathcal{X}} q_X(x) = 1$
- An arrow points from the circled equation to the text: "Note that this satisfies"

Figure 7: defq

Using this in the expression for $D(p_X || q_X)$, (take image from recording and put it here).

Handwritten derivation of $D(p_X || q_X)$:

Now $D(p_X || q_X) = \sum_{x \in \text{supp}(p_X)} p(x) \log \frac{1}{q(x)}$

Substituting the definition of $q_X(x)$:

$$= - \sum_{x \in \text{supp}(p_X)} p(x) \log \frac{1}{p(x)} + \sum_{x \in \text{supp}(p_X)} p(x) \log \left(\frac{1}{\frac{2^{-l(x)}}{\sum_{x \in \mathcal{X}} 2^{-l(x)}}} \right)$$

$$\begin{aligned}
&= -H(X) + \sum_{x \in \text{supp}(p_X)} p(x) \log_2 2^{l(x)} \\
&\quad + \sum_{x \in \text{supp}(p_X)} p(x) \log_2 \left(\sum_{x' \in \text{supp}} 2^{-l(x')} \right) \\
&\leq -H(X) + L \quad \left(\text{this is some nonpositive number} \right) \rightarrow \leq 1 \text{ by Kraft}
\end{aligned}$$

- o $D(p_X \| q_X) \leq -H(X) + L$

Equality happens when $p_X = q_X$, and that non positive term is zero.

30 June 2021

Now suppose we have a RV $X \in \{x_1, \dots, x_k\}$ and +ve integers such that $\sum_{x \in \mathcal{X}} 2^{-l_i} \leq 1$, then there exists a prefix free code for X with codeword lengths l_i .

Proof:

We will show that we can construct a binary tree with leaves at depths l_1, \dots, l_k .

Assume that,

$$l_1 \leq l_2 \leq \dots \leq l_k$$

WLOG. Now, for any $i \leq k$,

$$\sum_{j=1}^{i-1} 2^{-l_j} < 1 \dots \dots \dots (A)$$

as we are only summing up to $k - 1$ terms at max. Now we construct a tree with l_k levels, and we pick an available node at every step of the algorithm, and delete all of its successors. This gives us a prefix free tree.

We have to show that at each step, there is at least one undeleted node left at level i . For this, we use that above statement from Kraft's inequality (A).

Trivially, there is a node at l_1 .

Now for some level $i \leq k$,

Total nodes deleted at level k because of deletions of successors will be,

$$\sum_{j=1}^{i-1} 2^{l_k - l_j}$$

So nodes remaining at k are

$$2^{l_k} \left(1 - \sum_{j=1}^{i-1} 2^{-l_j}\right)$$

From (A), we can say that this value will always be positive for all i .

So, we always have nodes remaining at level k . By this, we can say that every level i must have nodes too, as level k is below i . Hence proved.

Remark: We construct the tree from smallest length codewords to largest length. The optimal code construction we will discuss will be constructed in reverse.

Now, if we have an RV X with a probability distribution P_X , we want to obtain the collection of integers l_1, \dots, l_k such that Kraft inequality is satisfied. Then we can construct the code as shown above. We can technically pick any l_i that satisfies it, but then we need to minimise the average length of the code as well. So we give small l_i for larger p_i and vice versa. Essentially, more probable symbols get smaller codewords.

Idea behind Shannon Fano codes

To do what we stated above, we can fix $l_i = \lceil \log(1/p_i) \rceil$ where p_i is probability of i th symbol. Clearly, $l_i \geq 1$. Now checking Kraft's inequality,

$$\begin{aligned} \text{Checking } K\text{-inequality} \\ \sum_{i=1}^k 2^{-l_i} &= \sum_{i=1}^k 2^{-\lceil \log_2 \frac{1}{p_i} \rceil} \leq \sum_{i=1}^k 2^{-\log_2 \frac{1}{p_i}} \\ &= \sum_{i=1}^k p_i = 1 \end{aligned}$$

Figure 8: cki

So we can use that tree construction for getting the code now. This code is called the Shannon Fano code for X .

Expected length of Shannon Fano code

$$L_{sf} = \sum_{i=1}^k p_i \cdot \text{ceil}(\log(1/p_i))$$

Now,

$$\sum_{i=1}^k p_i \cdot \text{ceil}(\log(1/p_i)) < \sum_{i=1}^k p_i \cdot (\log(1/p_i) + 1)$$

which implies

$$\sum_{i=1}^k p_i \cdot \text{ceil}(\log(1/p_i)) < H(X) + 1$$

This also shows that Shannon Fano code isn't always the optimal code for all X .

2 July 2021

Example for Shannon Fano code

$$X \in \mathcal{X} = \{x_1, x_2, x_3, x_4\}$$

$$P_X(x_i) = 1/4 \text{ for } i = 1, 1/2 \text{ for } i = 2, 1/9 \text{ for } i = 3, 5/36 \text{ for } i = 4$$

Length of codewords satisfying Kraft inequality for the Shannon Fano code

$$l_i = \text{ceil}(\log(1/P_X(x_i)))$$

This gives us

$$l_1 = 2 \quad l_2 = 1 \quad l_3 = 4 \quad l_4 = 3$$

So, we get

$$\bar{L} = \sum_{i=1}^4 p_i l_i$$

This gives us $\bar{L} = 67/36$ We can verify that this value lies between $H(X)$ and $H(X) + 1$

Now we can construct the tree according to these lengths.

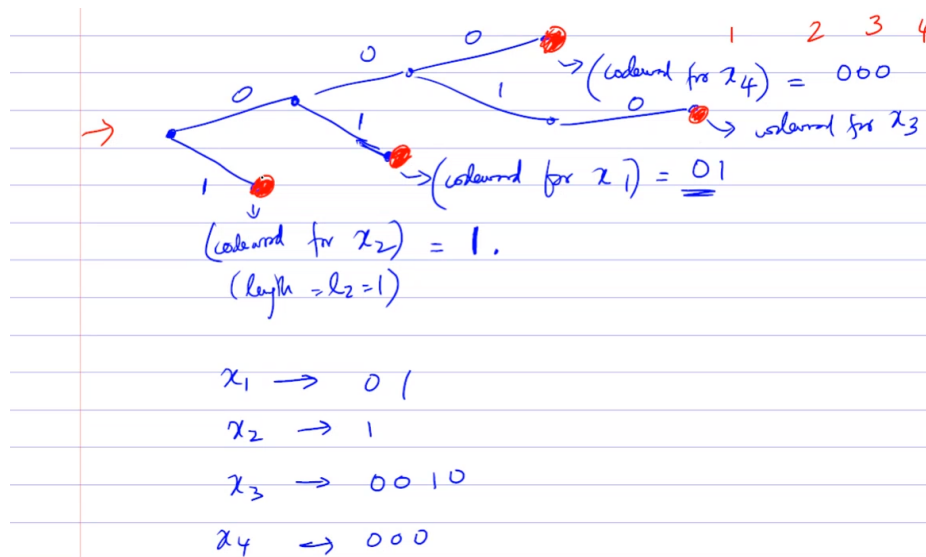


Figure 9: tree

That's it for Shannon Fano codes. This is NOT optimal in general. To prove this we will show a code construction that beats it in optimality, which is called the Huffman code.

Some Lemmas about an optimal code for an RV X

Assume that $X \in \mathcal{X} = \{x_1, x_2, \dots, x_k\}$ and $P_X(x_i) = p_i$. WLOG, we assume that $p_1 \geq p_2 \geq \dots \geq p_k$

- Lemma 1: Consider that l_1, \dots, l_k are lengths of codewords associated to the codewords in any optimal code for X . Then,

$$l_1 \leq l_2 \leq \dots \leq l_k$$

Proof: Suppose \mathcal{C} is an optimal code in which there exists $1 \leq i, j \leq k$ such that $p_i > p_j$, but $l_i > l_j$

We will show that there exists another code \mathcal{C}' which has smaller average length than \mathcal{C} , which contradicts the optimality of \mathcal{C} .

Consider \mathcal{C}' , which is the same as \mathcal{C} but the codewords for x_i, x_j are swapped.

Now, in \mathcal{C}' , $l_{i,\mathcal{C}'} = l_{j,\mathcal{C}}$ and same for the other codeword. So now, as there have been no CHANGES in codewords, we can conclude that \mathcal{C}' is also prefix free. Now, we can say

$$\overline{L}_{C'} = \sum_{k \in \{1, \dots, k\} - \{i, j\}} p_k l_k + p_i l_j + p_j l_i$$

$$\bar{L}_C = \sum_{k \in \{1, \dots, k\}} p_k l_k$$

So,

$$\overline{L_{C'}} - \overline{L_C} = p_i(l_j - l_i) + p_j(l_i - l_j)$$

$$\implies \overline{L_{C'}} - \overline{L_C} = (l_j - li)(p_i - p_j)$$

The RHS here is strictly negative, so here we reach a contradiction, as the initial code length is not optimal. Hence proved.

- Lemma 2: Consider the tree representation of an optimal prefix free code. For such a tree, every node must be a codeword, or it must have at least 2 successors which are codewords. This can be rephrased as “There are no unused leaves in the tree of an optimal code.”

3 July 2021

Continuing with lemma 2...

Proof:

Let \mathcal{C} be the optimal code. If we consider the tree corresponding to it, with an unused leaf,

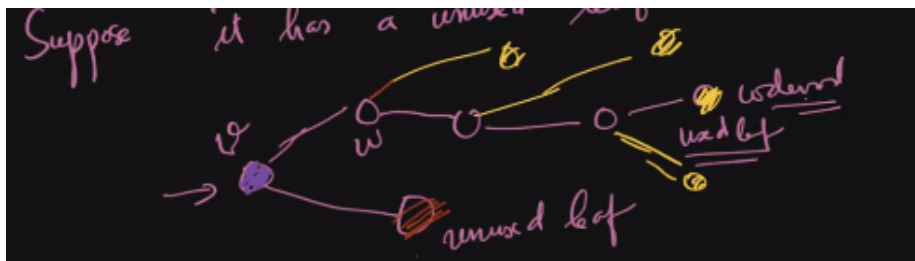


Figure 10: trul

We can transform such a tree into,

As we can see, here we have a shorter tree, which has a lower expected length than our previous code. The new code is still prefix free. This contradicts the optimality of \mathcal{C} . So, we have proved our claim.



Figure 11: trul2

Every optimal code must follow the above two lemmas.

- Lemma 3: There exists an optimal prefix free code for RV X such that the codewords associated to the two lowest probability symbols are siblings.

Proof: Let \mathcal{C} be some optimal code for X. If it already satisfies the property, there is nothing to show.

If it doesn't follow the lemma, we can use the fact that $l_{k-1} \leq l_k$ as $P_{k-1} \geq P_k$. So, as they are not siblings, P_k node will definitely have a sibling. Let us call that sibling P_i because of lemma one and lemma 2. Now, by lemma 1, we say that l_{k-1} is the second largest. So, l_i will have to be lesser than it. By this, we get

$$l_i \leq l_{k-1} \leq l_k$$

But we also know that $l_k = l_i$ as they are siblings. This leads us to

$$l_i \leq l_{k-1} \leq l_k = l_i$$

This implies

$$l_i = l_{k-1} = l_k$$

Now, we can interchange the codewords for the i th symbol and $(k-1)$ th symbol as they are at the same level, and obtain a new code. This code still has the same expected length, and is hence still optimal. This code however, satisfies our property. Hence, proved.

Huffman Coding

Assume that P_X is $p_1 \geq p_2 \dots \geq p_k$. Now, this code incorporates a bottom up approach to building the tree. We first mark out leaves, and then work our way up to the root.

We start by combining the two least probability symbols into one node with two leaves, and now consider a new probability distribution with $k-1$ terms, the last term being $p_k + p_{k-1}$. We perform the same operation again, and continue building our tree until we have nothing left to combine. Now we have our tree. This tree gives us the Huffman code, which is optimal.

Example:

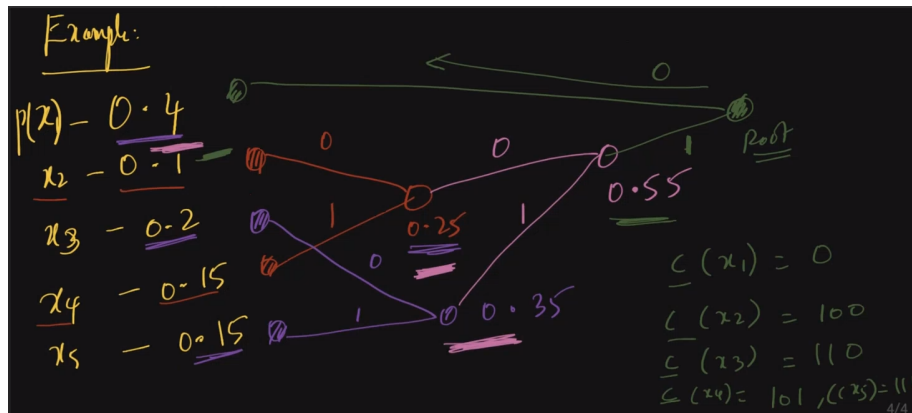


Figure 12: huffex

5 July 2021

Huffman code - Proof of optimality

Claim:

Let $\bar{L}_{Huffman} = \min(\bar{L}_C)$ for all prefix free codes C .

Proof: We will write the average length of some code for some random variable taking m values with probability q_1, q_2, \dots, q_m as L_C

Optimal length for the same distribution above is written as L^*