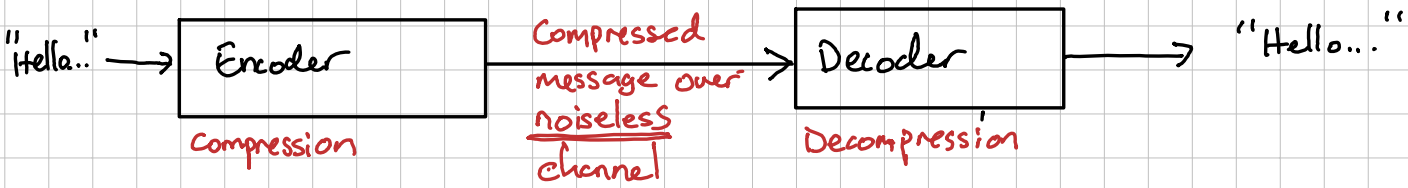
Compression

Entropy is particularly important in understanding data compression.



We care about:

Lossless Compression

AKA. "Symbol coding"
Guaranteed to reconstruct the original message perfectly.

Lossy Compression

Output approximates the input.
E.g. JPEG image.

We will start looking at lossless compression.

Symbol Coding (Lossless Compression)

Goal: Give each $x \in A_x$ a code to represent it

E.g. $A_x = \{a, b, c, \ldots\}$ ← symbols

x	c
a	01
b	11
⋮	

w.l.o.g. we will use alphabetic symbols and binary codes.

Fixed length codes are the simplest but they are only efficient if every input symbol is equiprobable...

Variable Length Codes

Imagine a 4-symbol source $\{a,b,c,d\}$ where

$P(a) = 0.98$
$P(b) = 0.01$
$P(c) = 0.005$
$P(d) = 0.005$

Fixed code:

a	00
b	01
c	10
d	11

aaaaaba

→ 00000000000100

simpler to decode.

Variable code:

a	0
b	10
c	110
d	111

→ 00000100

shorter

Probable symbol → small codeword
Improbable " → long codeword.

Compression can make things longer!

Input N bits $\xrightarrow{\text{compress}}$ M bits.

$M < N$ for compression.

$\#\text{ inputs} = 2^N$

$\#\text{ outputs} = 2^M$

If $M < N$ always for some comp. scheme some strings must map to the same output.

∴ Not every string can be compressed in a given scheme

$\Rightarrow$ Some strings will get longer

Decodability

Uniquely decodable codes give an unambiguous decode.

x	C_3
a	1
b	0
c	00
d	11

$r = 100011$

$r = \underset{a}{1}\underset{}{0}\underset{}{0}\underset{}{0}\underset{}{1}\underset{}{1}$

Not uniquely decodable

x	C_1
a	101
b	00
c	0001
d	1

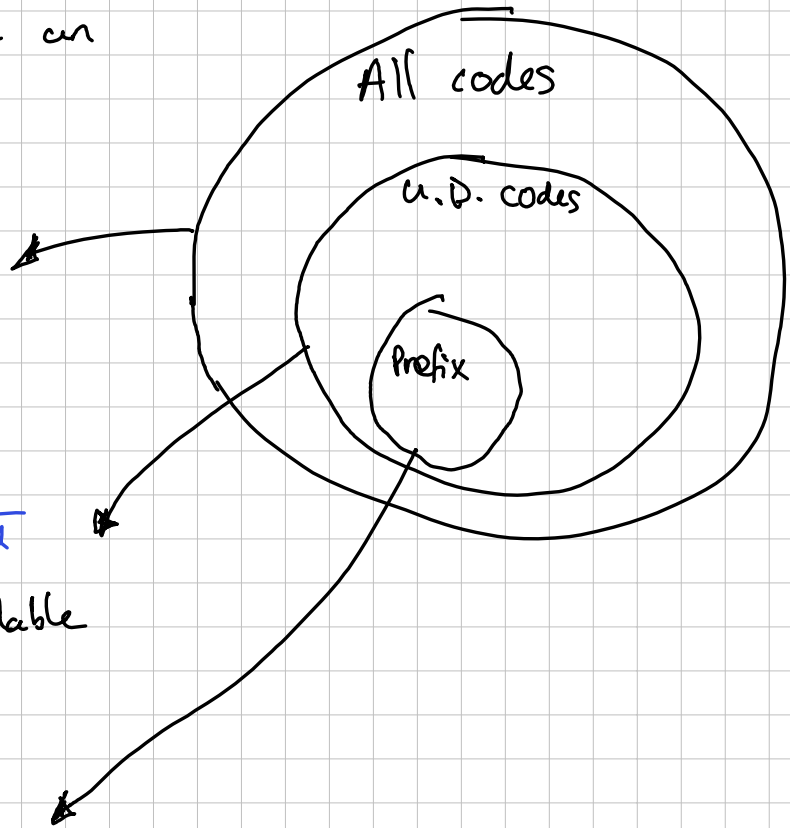
$r = \underset{a}{101}\underset{b}{00}\underset{c}{0001}\underset{d}{1}$

uniquely decodable

x	C_2
a	1
b	01
c	001
d	000

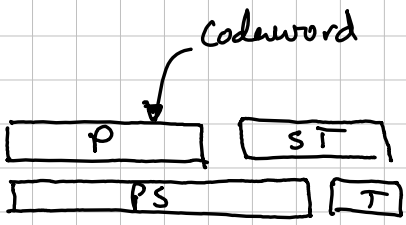
$r = \underset{a}{1}\underset{b}{01}\underset{c}{001}\underset{d}{000}$

prefix code

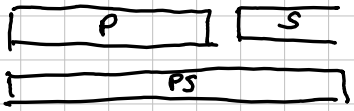


Testing for U.D.

Ambiguity when eg.

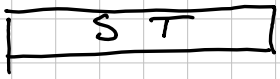


① Find PS



find all prefixes and note the remainders

② Look for suffixes to S



Find all T for S in codewords.

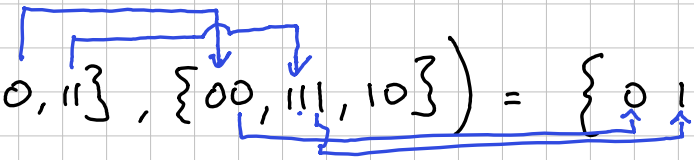
③ If T is a codeword ⇒ not U.D.

Sardinas - Patterson

Take a set of codewords, C

Define $S(A,B)$ = The set of remainder suffixes from B using A as prefixes.

$$= \{ w \mid uv = w \text{ where } u \in A, uv \in B \}$$



E.g. $S(\{0, 11\}, \{00, 111, 10\}) = \{0, 1\}$

$S(\{1, 11, 111\}, \{0, 011\}) = \emptyset$

$S(\{10, 111, 000\}, \{1010, 11, 0001\}) = \{10, 1\}$

$R_1 = \{$ Remainders of all prefixes in $C\}$

if $R_1 = \phi$:
    return DECODABLE

for $(i = 2, i < \infty, i{+}{+})$:

    $R_i = S(R_{i-1}, C) \cup S(C, R_{i-1})$

    if $R_i = \phi$:
        return DECODABLE
    elif $R_i \cap C \neq \phi$
        return NOT-DECODABLE
    elif $R_i = R_{i-1}$:
        return DECODABLE

Eg. 3.1

C 11 00 01 110 001

C 11 00 110 001

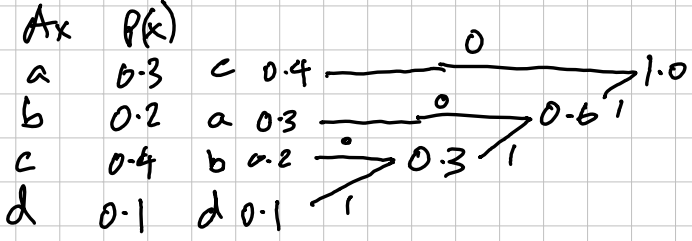
C 00 01 10 11

C 10 00 0001 1

Prefix codes

A prefix or prefix-free code is one where no codeword is a prefix of another

Huffman coding



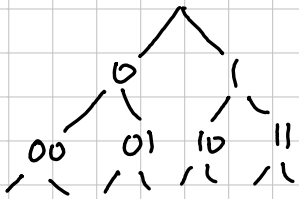
a	10
b	110
c	0
d	111

1. write them out in desc prob. order
2. Combine lowest two into a pseudosymbol with prob. the sum of the two inputs
3. Repeat until tree full
4. Label each branch
5. Read code backwards

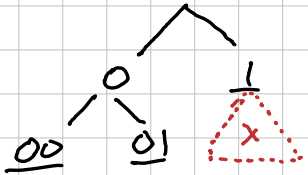
Eg. 3.2.

a	0.03
b	0.05
c	0.07
d	0.06
e	0.72
f	0.07

The Kraft Inequality



Consider the binary tree of codewords.



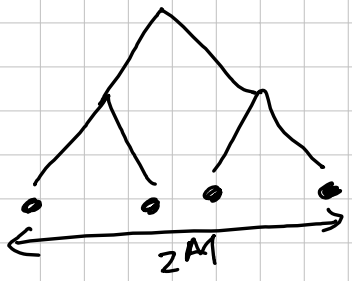
In a prefix code, a codeword removes the subtree beneath it.

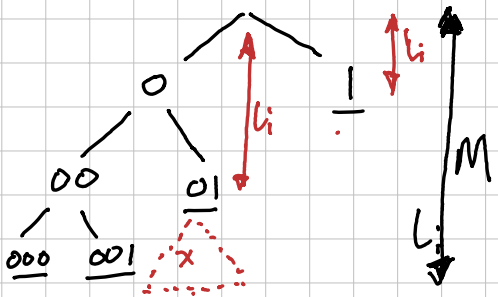
$\Rightarrow$ Codewords must be leaves.

Consider max. codeword length, M

Max. # codewords when all $l_i = M$

$$N_{max} = 2^M$$





Each codeword of length l_i removes 2^{M-l_i} from the depth M codewords

$$\text{Total removed} = \sum_i 2^{M-l_i}$$

$$\leq N_{max}$$

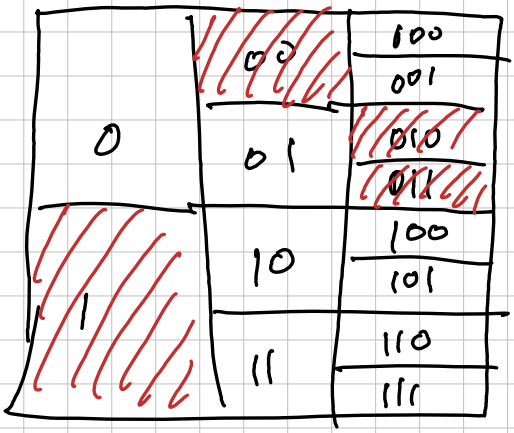
$$\leq 2^M$$

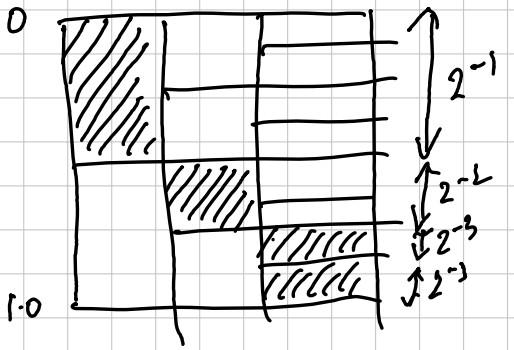
$$\Rightarrow \quad \sum 2^{M-l_i} \leq 2^M$$

$$\boxed{\sum_i 2^{-l_i} \leq 1}$$

Kraft Inequality

Alternative Codeword Budget View





A prefix code requires that the selected codeword occupies the horizontal space uniquely.

The only way we run out of budget $(0 \rightarrow 1.0]$ is if

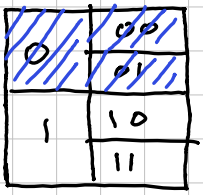
$$\sum 2^{-l_i} > 1$$

$\therefore$ if $\sum 2^{-l_i} \leq 1$ there is at least one prefix code that satisfies l_i

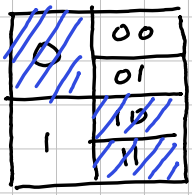
What this means

> **Any** set of integers $\{l_i\}$ where $\sum_j 2^{-l_j} \leq 1$ can form a prefix code

E.g. $\{1, 2, 2\}$



Not prefix



Prefix

Gibbs Inequality

$$\sum_i p_i \log \frac{1}{q_i} \geq \sum_i p_i \log \frac{1}{p_i}$$

Mackay pg 97

Bounding Compression : Linking to Entropy

Ave code length $L(C,X) = \sum_i P_i l_i$ — Can we bound this?

Lower Limit

Let $q_i = \frac{2^{-l_i}}{\sum_i 2^{-l_i}} = \frac{2^{-l_i}}{z}$

Then $l_i = -\log q_i z$ (rearrangement)

Then $L(C,X) = -\sum P_i \log q_i z$

$= -\sum P_i \log \frac{1}{q_i} - \sum P_i \log z$

$\sum P_i \log \frac{1}{q_i} \geq \sum P_i \log \frac{1}{P_i}$ ← Gibb's Inequality, Mackay pg. 97

$z \leq 1$ by Kraft

$\Rightarrow \quad L(C,X) \geq H(X) + k \qquad k \geq 0$

$$\boxed{L(C,X) \geq H(X)}$$

Can't go below entropy

Upper Limit $L(C,X) = H(X)$ may be impossible since it may need fractional lengths of codewords.

$$L(C,X) \geq H(X)$$

Let $l_i = \lceil \log_2 \frac{1}{p_i} \rceil$

Could this be a prefix code?? $\Rightarrow$

$$\sum 2^{-l_i} = \sum 2^{-\lceil \log_2 \frac{1}{p_i} \rceil} \leq \sum 2^{-\log \frac{1}{p_i}}$$

$$\leq \sum p_i$$

$$\leq 1 \quad \checkmark$$

Could be a prefix by Kraft.

$$L(C,X) = \sum p_i l_i = \sum p_i \lceil \log \frac{1}{p_i} \rceil$$

$$< \sum p \log \frac{1}{p_i} + 1$$

$$< H(X) + 1$$

$$H(X) \leq L(C,X) < H(X)+1$$

Source Coding Theorem for Prefix Codes

We can't squeeze below the entropy of our source but we're guaranteed there is a coding scheme within 1 bit of it !!

Shannon Coding

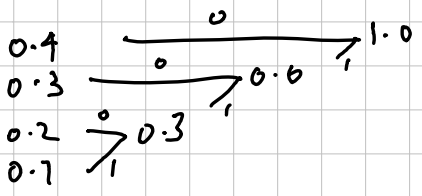
what if we just assigned codes according to $-\log p_i$, the Shannon Info.

	p_i	l_i	$\lceil l_i \rceil$
a	0.4	1.32	2
b	0.3	1.73	2
c	0.2	2.32	3
d	0.1	3.22	4

$L(C,X) = 2.4$ bits.

$H(X) = 1.84$ (So we still got $L(C,X) \leq H(X)+1$)

But Huffman:



l_i:
1
2
3
3

$L(C,X) = 1.9$ bits.