Module 2 - Source codes

* Introdultion

→ Source codes are used to compress or encode data in order to reduce its size for efficient storage and transmission

→ The goal of source coding is to minimize the expected length of codeword while maintai lossy or lossless compression

→ A lossless compression ensures orignal data is reconstructed perfectly.

→ A lossy compression allows some loss of information but still relains fidelity for practical purposes.

* Coding Parameters

→ Various Parameters are associated with source codes :-

① Code Length - The length of a codeword is the number of symbols in the sequence that represents it. Shorter codes provide better compression but potentially require complex decoding algorithms.

② Code efficiency :- The efficiency of code is a measure of how close it comes to achieving the theoritical lower bound on the avg. code length.

③ Code Redundancy - Redundancy in a code refers to the amount of additional information contained in the codeword length beyond requirement.

Redundancy can be measured in terms of avg. codeword length and entropy.

④ Code prefixness - A prefix code is one in which no codeword is a prefix of another codeword. Prefix codes allow unambiguous decoding with no need for delimitters.

⑤ Code Universality - A universal code is one that can be used to compress any source with arbitrarily high probability of error free decoding.

---

* The Source code theorem.

→ Source coding Theorem / Shannon's coding Theorem is a fundamental result in information theory that establishes the theoritical limits of lossless data.

→ It states that, for any source with H it is possible to construct a uniquely decodable code with an average codeword length close to H; and that no uniquely decodable code can achieve avg. codeword length less than H.

→ So the source coding theorem sets a lower bound on the avg. length of codewords required to encode a source.

→ This theorem has important applications in data compression, implying that no lossless compression algorithm can achieve rates greater than the entropy of source.

---

* Classification of codes

① Shannon - Faro coding

→ It is a loss-less data compression technique used to encode data taking up less space than before.

→ Thes steps in the algorithm are:-

① Calculate probability of each symbol.

② Sort the symbols in decreasing probability.

③ More probable half is assigned a value {0,1} and less probable is opposite to more probable.

④ This is repeated recursively until all symbols are assigned a binary code.

Example ① → Encode $[m] = [m_1, m_2, m_3, m_4, m_5]$

$[p] = [0.30, 0.25, 0.15, 0.12, 0.08, 0.10]$

m	P(m)	1st	2nd	3rd	codeword	length
m_1	0.30	0	0		00	2
m_2	0.25	0	1		01	2
m_3	0.15	1	0	0	100	3
m_4	0.12	1	0	1	101	3
m_5	0.10	1	1	0	110	3
m_6	0.08	1	1	1	111	3

$$H = -\sum P_K \log_2 P_K$$

$$= -[0.30 \times \log_2 0.30 + 0.25 \log_2 (0.25) + 0.15 \log_2 (0.15) + 0.12 \log_2 (0.12) + 0.10 \log_2 (0.10) + 0.08 \log_2 (0.08)]$$

$$= +[0.521 + 0.5 + 0.41 + 0.367 + 0.332 + 0.291]$$

$$= 2.421 \text{ bits}$$

$$L = \sum n \times P_k$$

$$2\times0.3 + 2\times0.25 + 3\times0.15 + 3\times0.12 + 3\times0.10 + 3\times0.08$$

$$L = 2.45.$$

$$\therefore \eta = \frac{H(S)}{L}$$

$$= \frac{2.421}{2.45} = 0.988$$

$$= 98.8\%$$

Example (2) $\left\{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}\right\}$

$\{S_1, S_2, S_3, S_4\}$

		1st	2nd	3rd	Length
S_1	1/2	0			1
S_2	1/4	1	0		2
S_3	1/8	1	1	0	3
S_4	1/8	1	1	1	3

$$H(x) = -\left[\frac{1}{2}\log_2\left(\frac{1}{2}\right) + \frac{1}{4}\log_2\left(\frac{1}{4}\right) + \frac{1}{8}\log_2\left(\frac{1}{8}\right) + \frac{1}{8}\log_2\left(\frac{1}{8}\right)\right]$$

$$= \left[2 \times 0.5 + 0.5 + 0.375 + 0.375 \right]$$

$$= 1.75 \text{ bits}$$

$$L = \sum P_k n_k$$

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

$$= 1.75 \text{ symbol.}$$

$$\eta = \frac{H}{L} = \frac{1.75}{1.75} = 1.00$$

Efficiency: 100%.

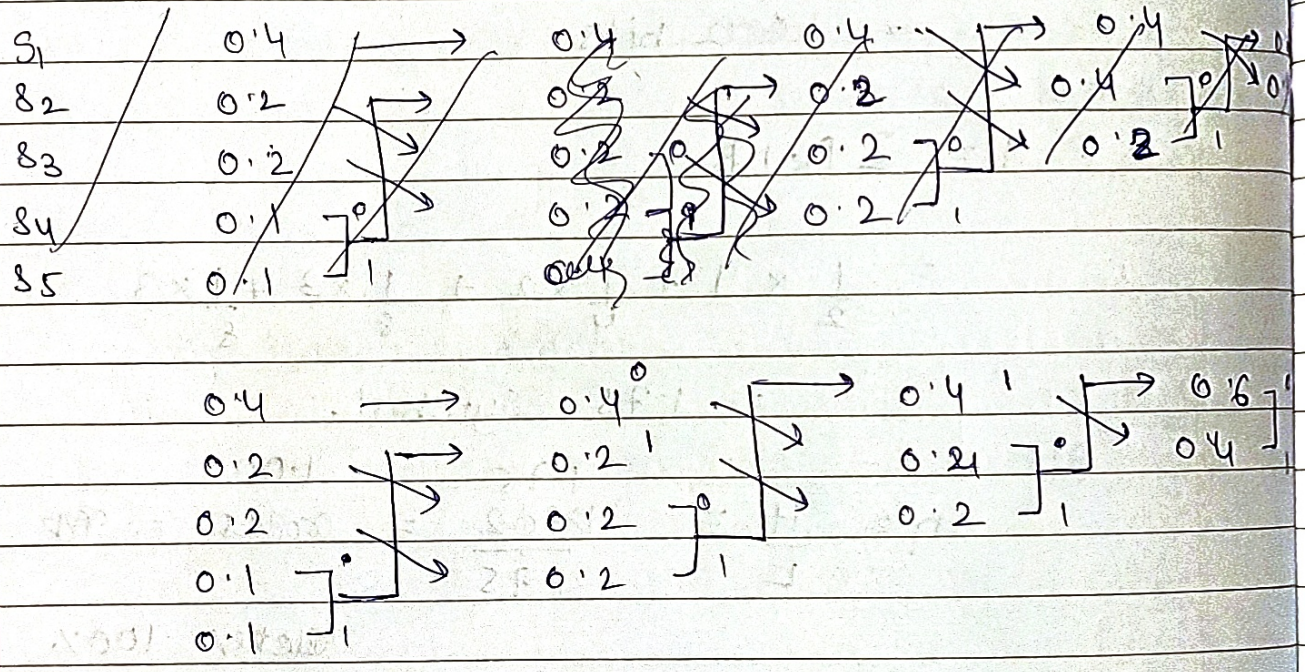
---

* Huffmann Coding.

→ This algorithm works by first analyzing the data to be compressed and determining the frequency of occurrence of each symbol.

→ Then a binary tree is constructed, with each leaf node representing a symbol and the frequency of that symbol.

Example ① → P = { 0.4, 0.2, 0.2, 0.1, 0.1



		codeword	Length
S_1	0.4	00	2
S_2	0.2	01	2
S_3	0.2	11	2
S_4	0.1	010	3
S_5	0.1	011	3

$$H = -\sum P_n \log P_n$$

$$= -[0.4 \log_2(0.4) + 0.2 \log_2(0.2) \times 2 + 0.1 \log_2(0.1) \times 2]$$

$$= -[0.528 + 0.928 + 0.664]$$

$$= 2.12 \text{ bits}$$

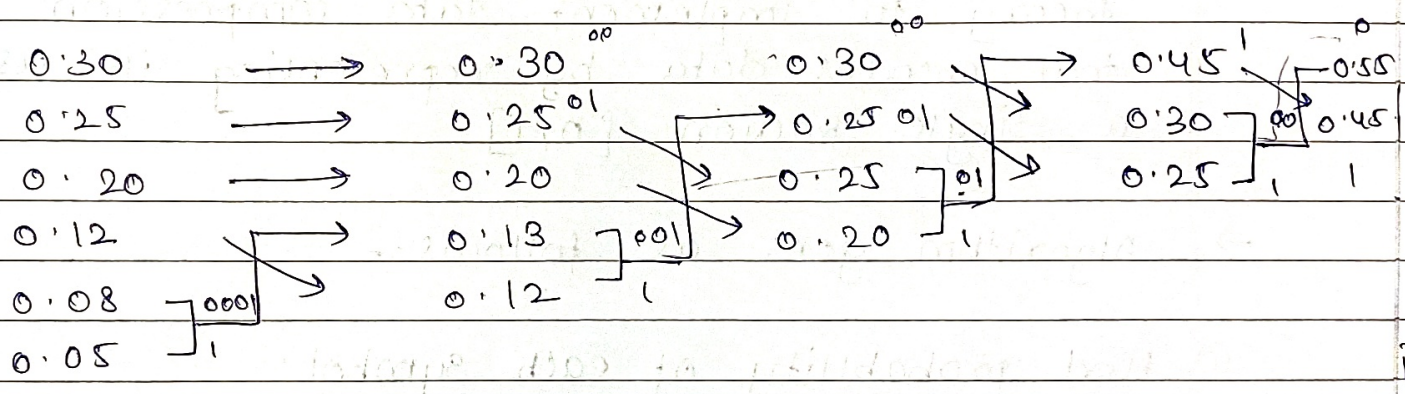
$0.4 \times 2 + 0.2 \times 2 + 0.2 \times 2 + 0.1 \times 3 + 0.1 \times 3$

$= 2.2$ bits.

$$\eta = \frac{2.12}{2.2} = 0.963$$

$= 96.36\%$.

Example (2)



	codeword	length
0.30	00	2
0.25	01	2
0.20	11	2
0.12	101	3
0.08	1000	4
0.05	1001	4

$$H = -\{0.521 + 0.5 + 0.464 + 0.367 + 0.291 + 0.216\}$$

$= 2.359$ bits.

$$L = 0.3 \times 2 + 0.25 \times 2 + 0.20 \times 2 + 0.12 \times 3 + 0.08 \times 4 + 0.05 \times 4.$$

$$\eta = \frac{2.359}{2.38} = 99.11\%.$$

* Arithematic coding

→ It is a technique used in information theory to implement data compression that encodes data by representing it as a single fraction [0, 1].

→ Algorithm goes as follows :-

① Find probability of each symbol.

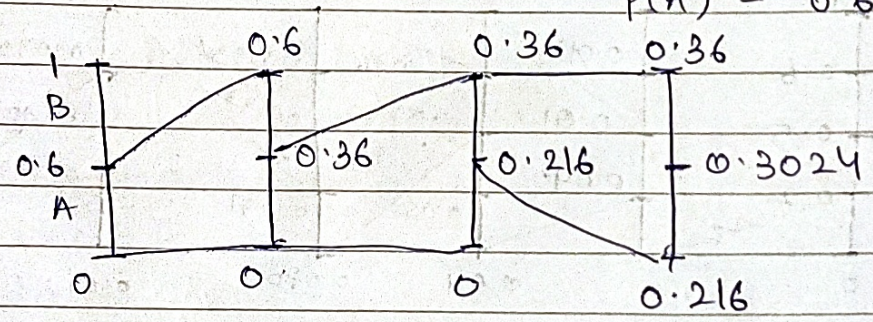
② A range is assigned to each symbol

③ The entire message is then represented as a single fraction that falls within the range of the first symbol.

④ The range keeps getting divided into sub-ranges that correspond to the probability distribution. This repeats till all the msgs are encoded.

Example ①

P(A) = 0.6 ; P(B) = 0.4



$R = LL + D \times P$

$R = 0 + 0.6 \times 0.6$

$= 0.36$

$R = 0 + 0.36 \times 0.6$

0.216

$R = 0.216 + 0.144 \ (0.6)$

$0.216 + 0.0864$

$= 0.3024$

---

* L2W coding

→ L2W (Lempel - Ziv - Welch) coding is a lossle compression commonly used for GIF, TIFF,

→ The algorithm works by replacing sequences o characters with codes. It starts by creating a dictionary that contains all possible single characters.

Example ① Encode ababbabc ababba using LZW Coding

① Initializing the base dictionary.

index	entry
1	a
2	b
3	c

② Encode O/P

Encode O/P
-
-
-
1
2
4
5
2
3
4
6
1

Dictionary

Index	Entry.
1	a
2	b
3	c
4	ab
5	ba
6	abb
7	bab
8	bc
9	ca
10	aba
11	abba

|| 1 2 4 5 2 3 4 6 1 ||.

Example ② AABABBBABABABBABBABB

①

Index	Entry
1	A
2	B

②

Encode O/P	Index	Entry
-	1	A
-	2	B
1	3	AB
2	4	BA
3	5	ABB

Encode O/P	Index	Entry
-	1	A
-	2	B
1	3	AA
1	4	AB
2	5	BA
4	6	ABB
2	7	BB
5	8	BAB
5	9	BAA
4	10	ABA
3	11	AAB
8	12	BABB
7	13	BBA
13	14	BBAB

1124255 438 713

* Run Length Encoding

→ Run-length encoding is a simple and effective lossless data compression technique used to compress data with repeated values

Simplest compression algorithm:

Example ① 00000 11111 00 1 0000 1 01

{0,5}, {1,5}, {0,2}, {1,1}, {0,4}, {1,1}, {0,1}, {1,1}
① ② ③ ④ ⑤ ⑥ ⑦ ⑧

Bits = $\log_2 8$ = 3 bits.

{0,101}, {1,101}, {0,010}, {1,001}, {0,100}

{1,001}, {0,001}, {1,001}

Final code -

0101 1101 0010 1001 0100 1001 0001 1001.