# Unit - 2

# Mathematical Preliminaries For Lossless Compression Models

# Outline…

- Physical Models
- Probability Models
- Markov Models
- Composite Source Model
- Coding
  - Uniquely Decodable Codes
  - Prefix Codes
- Algorithmic Information Theory
- Minimum Description Length Principle

# A Brief Introduction to Information Theory

- Although the idea of a quantitative measure of information has been around for a while, the person who pulled everything together into what is now called information theory was Claude Elwood Shannon, an electrical engineer at Bell Labs.
- **Shannon defined a quantity called *self-information*.**
- Suppose we have an event A, which is a set of outcomes of some random experiment. If P(A) is the probability that the event A will occur, then the self-information associated with A is given by

$$\mathbf{i(A) = \log_b 1 / P(A) = -\log_b P(A)}$$

# A Brief Introduction to Information Theory

- Suppose A and B are two independent events. The self-information associated with the occurrence of both event A *and event B is, by Equation.*

$$\mathbf{i\,(AB) = \log b\ 1 \,/\, P\,(AB)}$$

- As A and B are independent,

$$\mathbf{P(AB) = P(A)\ P(B)}$$

- and

$$\begin{aligned} \mathbf{i(AB)} &= \mathbf{\log b\ 1 \,/\, P(A)\ P(B)} \\ &= \mathbf{\log b\ 1 \,/\, P(A) + \log b\ 1 \,/\, P(B)} \\ &= \mathbf{i\,(A) + i\,(B)} \end{aligned}$$

# Example

- Let H and T be the outcomes of flipping a coin. If the coin is fair, then

$$P(H) = P(T) = 1 / 2$$

- and

$$i(H) = i(T) = 1 \text{ bit}$$

- If the coin is not fair, then we would expect the information associated with each event to be different. Suppose

$$P(H) = 1/8, P(T) = 7/8$$

- Then

$$i(H) = 3 \text{ bit}, i(T) = 0.193 \text{ bit}$$

# The entropy can then be calculated

$$H = -\sum_{i=1}^{10} P(i) \log_2 P(i).$$

Shannon **entropy** provides an absolute limit on the best possible average length of lossless encoding or **compression** of an information source.

# Example

- If we have a symbol set {A,B,C,D,E} where the symbol occurrence frequencies are:

  **A = 0.5 , B = 0.2 , C = 0.1 , D = 0.1 , E = 0.1**

- The average minimum number of bits needed to represent a symbol is

  $$H(X) = -[(0.5\log_2 0.5 + 0.2\log_2 0.2 + (0.1\log_2 0.1)*3)]$$
  $$H(X) = -[-0.5 + (-0.46438) + (-0.9965)]$$
  $$H(X) = -[-1.9]$$
  $$H(X) = 1.9$$

- Rounding up, we get 2 bits/per symbol. To represent a ten character string AAAAABBCDE would require 20 bits if the string were encoded optimally.

# Physical Models

- If we know something about the physics of the data generation process, we can use that information to construct a model.
- For example, in speech-related applications, knowledge about the physics of speech production can be used to construct a mathematical model for the sampled speech process.
- Sampled speech can then be encoded using this model.

# Probability Models

- The simplest statistical model for the source is to assume that each letter that is generated by the source is independent of every other letter, and each occurs with the same probability.
- We could call this the ***ignorance model***, *as it would generally be useful only when we know* nothing about the source.
- For a source that generates letters from an alphabet

  A = {a1,a2,.....aM}, we can have a *probability model*

  P = {P(a1),P(a2),...P(aM)}.

# Markov Models

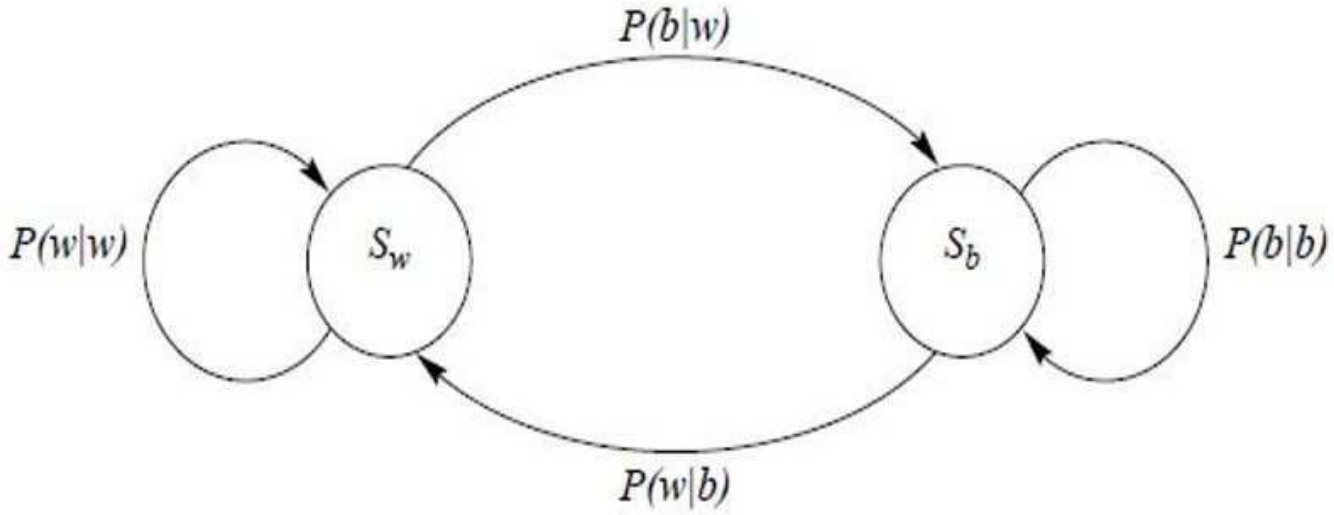
- For models used in lossless compression, we use a specific type of Markov process called a ***discrete time Markov chain***.
- For example, consider a binary image.
- The image has only two types of pixels, white pixels and black pixels. We know that the appearance of a white pixel as the next observation depends, to some extent, on whether the current pixel is white or black.
- Therefore, we can model the pixel process as a discrete time Markov chain.
- Define two states Sw and Sb.
- We define the transition probabilities P(w/b) and P(b/w), and the probability of being in each state P(Sw) and P(Sb).

# Markov Models

- The entropy of a finite state process with states Si is simply the average value of the entropy at each state:

$$H = \sum_{i=1}^{M} P(S_i) H(S_i)$$

# A two-state Markov model for binary images




For our particular example of a binary image

**H(Sw)=–P(b/w) log P(b/w) – P(w/w) log P(w/w)**

where P(w/w) = 1–P(b/w)

# Example Markov model

- To see the effect of modeling on the estimate of entropy, let us calculate the entropy for a binary image, first using a simple probability model and then using the finite state model described above. Let us assume the following values for the various probabilities:

  **P(Sw) = 30/31** **P(Sb) = 1/31**

  **P(w/w)=0.99 P(b/w)=0.01 P(b/b)=0.7 P(w/b)=0.3**

- Then the entropy using a probability model and the iid assumption is

  **H = –0.96 log 0.96–0.03 log 0.03 = 0.213 bits**

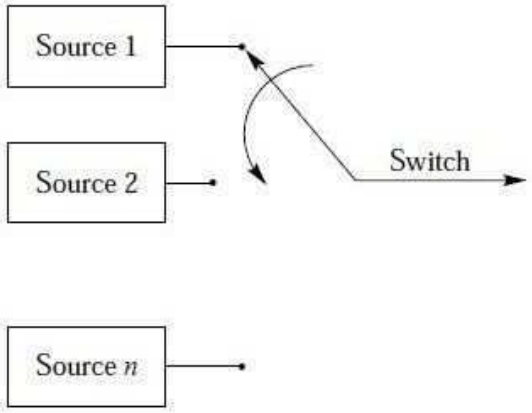
- Now using the Markov model

  **H(Sb)=–0.3 log 0.3–0.7 log 0.7 = 0.881 bits**

  **H(Sw)=–0.01 log 0.01–0.99 log 0.99 = 0.081 bits**

# Composite Source Model

- In many applications, it is not easy to use a single model to describe the source. In such cases, we can define a *composite source, which can be viewed as a combination or composition of* several sources, with only one source being *active at any given time.*
- A composite source can be represented as a number of individual **sources Si**, each with its own **model Mi**, and a switch that selects a source Si with **probability Pi**.



# Coding

- When we talk about *coding* its mean the assignment of binary sequences to elements of an alphabet.
- **The set of binary sequences is called a code, and the individual members of the set are called codewords**.
- **An *alphabet is a* collection of symbols called *letters***.
- The ASCII code for the letter *a* is 1000011, the letter *A is coded as 1000001, and the letter "," is coded as 0011010. Notice* that the ASCII code uses the same number of bits to represent each symbol.
- Such a code is called a ***fixed-length code***.

# Coding

- If we want to reduce the number of bits required to represent different messages, we need to use a different number of bits to represent different symbols.
- If we use fewer bits to represent symbols that occur more often, on the average we would use fewer bits per symbol.
- **The average number of bits per symbol is often called the *rate* of the code.**
- The idea of using fewer bits to represent symbols that occur more often is the same idea that is used in **Morse code**: the codewords for letters that occur more frequently are shorter than for letters that occur less frequently.
- For example, the codeword for **E is** ·, while the codeword for **Z is** ––·· .

# Uniquely Decodable Codes

- The average length of the code is not the only important point in designing a "good" code.
- Suppose our source alphabet consists of four letters a1, a2, a3, and a4, with probabilities P(a1)=1/2 , P(a2)=1/4 ,and P(a3)=P(a4)=1/8.
- The entropy for this source is 1.75 bits/symbol.
- Consider the codes for this source in Table
- The average length l for each code is given by

$$l = \sum_{i=1}^{4} P(a_i) n(a_i)$$

- where n(ai) is the number of bits in the codeword for letter ai and the average length is given in bits/symbol.

# Four different codes for a four-letter alphabet

| Letters | Probability | Code 1 | Code 2 | Code 3 | Code 4 |
|---|---|---|---|---|---|
| $a_1$ | 0.5 | 0 | 0 | 0 | 0 |
| $a_2$ | 0.25 | 0 | 1 | 10 | 01 |
| $a_3$ | 0.125 | 1 | 00 | 110 | 011 |
| $a_4$ | 0.125 | 10 | 11 | 111 | 0111 |
| *Average length* | | 1.125 | 1.25 | 1.75 | 1.875 |

# Uniquely Decodable Codes

- Based on the average length, Code 1 appears to be the best code.
- However, to be useful, a code should have the ability to transfer information in an unambiguous manner. This is obviously not the case with Code 1. Both a1 and a2 have been assigned the codeword 0. When a 0 is received, there is no way to know whether an a1 was transmitted or an a2. We would like each symbol to be assigned a *unique codeword.*
- At first glance Code 2 does not seem to have the problem of ambiguity; each symbol is assigned a distinct codeword. However, suppose we want to encode the sequence a2 a1 a1. Using Code 2, we would encode this with the binary string 100.
- However, when the string
- 100 is received at the decoder, there are several ways in which the decoder can decode this string. The string 100 can be decoded as a2 a1 a1, or as a2 a3.

# Uniquely Decodable Codes

- This means that once a sequence is encoded with Code 2, the original sequence cannot be recovered with certainty.
- We would like *unique decodability from* the code; that is, any given sequence of codewords can be decoded in one, and only one, way.
- We have already seen that Code 1 and Code 2 are not uniquely decodable. How about Code 3? Notice that the first three codewords all end in a 0. In fact, a 0 always denotes the termination of a codeword.
- The final codeword contains no 0s and is 3 bits long. Because all other codewords have fewer than three 1s and terminate in a 0, the only way we can get three 1s in a row is as a code for a4.
- The decoding rule is simple. Accumulate bits until you get a 0 or until you have three 1s. There is no ambiguity in this rule, and it is reasonably easy to see that this code is uniquely decodable.

# Uniquely Decodable Codes

- With Code 4 we have an even simpler condition. Each codeword starts with a 0, and the only time we see a 0 is in the beginning of a codeword. Therefore, the decoding rule is accumulate bits until you see a 0. The bit before the 0 is the last bit of the previous codeword.
- There is a slight difference between Code 3 and Code 4.
- In the case of Code 3, the decoder knows the moment a code is complete.
- In Code 4, we have to wait till the beginning of the next codeword before we know that the current codeword is complete.
- Because of this property, **Code 3 is called an *instantaneous code***.
- *Although Code 4 is not an instantaneous* code

# Uniquely Decodable Codes

**TABLE 2.2 Code 5.**

| Letter | Codeword |
|---|---|
| $a_1$ | 0 |
| $a_2$ | 01 |
| $a_3$ | 11 |

**TABLE 2.3 Code 6.**

| Letter | Codeword |
|---|---|
| $a_1$ | 0 |
| $a_2$ | 01 |
| $a_3$ | 10 |

# Uniquely Decodable Codes

- Let's decode the string 0111111111111111
- In this string, the first codeword is either 0 corresponding to a1 or 01 corresponding to a2.
- Starting with the assumption that the first codeword corresponds to a1, the next eight pairs of bits are decoded as a3.
- However, after decoding eight a3s, we are left with a single **(dangling)** 1 that does not correspond to any codeword.
- On the other hand, if we assume the first codeword corresponds to a2, we can decode the next 16 bits as a sequence of eight a3s, and we do not have any bits left over.
- The string can be uniquely decoded. In fact, **Code 5**, while it is certainly not instantaneous, is **uniquely decodable**.

# Uniquely Decodable Codes

- The coded sequence is 0101010101010101010.
- The first bit is the codeword for a1. However, we can also decode it as the first bit of the codeword for a2.
- If we use this (incorrect) decoding, we decode the next seven pairs of bits as the codewords for a2. After decoding seven a2s, we are left with a single 0 that we decode as a1.
- Thus, the incorrect decoding is also a valid decoding, and this code is not uniquely decodable.

# A Test for Unique Decodability

- We have two binary codewords a and b, where a is k bits long, b is n bits long, and k < n. If the first k bits of b are identical to a, then a is **called a prefix of b**. The last n−k bits of b are **called the dangling suffix.**
- For example, if a = 010 and b = 01011, then **a is a prefix of b** and the **dangling suffix is 11.**
- Now repeat the procedure using this larger list. Continue in this fashion until one of the following two things happens:
    1. **You get a dangling suffix that is a codeword.**
    2. **There are no more unique dangling suffixes.**

- **If you get the first outcome, the code is not uniquely decodable. However, if you get the second outcome, the code is uniquely decodable.**

# Example

- Consider Code 5. First list the codewords

**{0, 01, 11}**

- The codeword 0 is a prefix for the codeword 01. The dangling suffix is 1. There are no other pairs for which one element of the pair is the prefix of the other. Let us augment the codeword list with the dangling suffix.

**{0, 01, 11, 1}**

- Comparing the elements of this list, we find 0 is a prefix of 01 with a dangling suffix of 1.
- But we have already included 1 in our list. Also, 1 is a prefix of 11. This gives us a dangling suffix of 1, which is already in the list.
- There are no other pairs that would generate a dangling suffix, so we cannot augment the list any further.
- Therefore, Code 5 is uniquely decodable.

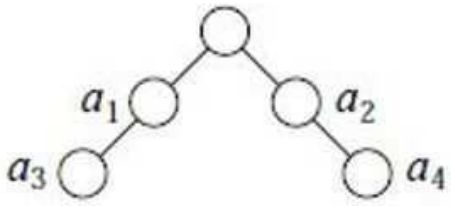
# Prefix Codes

- If the dangling suffix is itself a codeword, then the code is not uniquely decodable.
- **A code in which no codeword is a prefix to another codeword is called a *prefix code.***
- A simple way to check if a code is a prefix code is to draw the rooted binary tree corresponding to the code.
- One of these branches corresponds to a 1 and the other branch corresponds to a 0.
- we will adopt the convention that when we draw a tree with the root node at the top, the left branch corresponds to a 0 and the right branch corresponds to a 1.
- Using this convention, we can draw the binary tree for Code 2, Code 3, and Code 4.

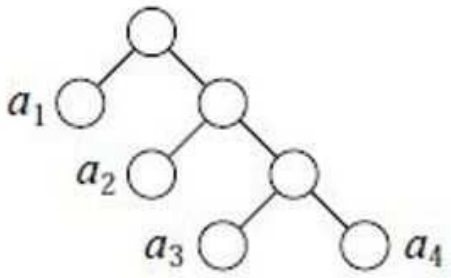
# Prefix Codes

- **Note** that apart from the root node, the trees have two kinds of nodes—nodes that give rise to other nodes and nodes that do not.
- The first kind of nodes are **called *internal nodes***.
- Second kind are **called external nodes or leaves**.
- In a prefix code, the codewords are only associated with the external nodes.
- A code that is not a prefix code, such as Code 4, will have codewords associated with internal nodes.

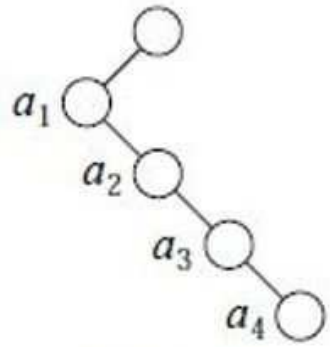
# Binary trees for three different codes



Code 2



Code 3



Code 4

# Algorithmic Information Theory

- Algorithmic information theory is a different way of looking at information that has not been as useful in practice but it gets around this theoretical problem.
- At the **heart of algorithmic information theory is a measure called *Kolmogorov complexity.***
- The Kolmogorov complexity K(x) of a sequence x is the size of the program needed to generate x.
- If x was a sequence of all ones, a highly compressible sequence, the program would simply be a print statement in a loop. On the other extreme, if x were a random sequence with no structure then the only program that could generate it would contain the sequence itself.
- The problem is we do not know of any systematic way of computing or closely approximating Kolmogorov complexity.