

ECE6258 Lecture 15

Introduction to Entropy Coding: Source Models

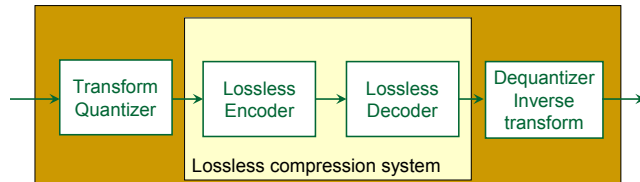
How does compression work?

- Exploit statistical redundancy
 - Take advantage of patterns in the signal.
 - Describe frequently occurring events efficiently.
 - **Lossless coding**: only statistical redundancy.
- Introduce acceptable deviations
 - Omit information that humans cannot perceive.
 - Match the signal resolution (in space, time, amplitude) to the application.
 - **Lossy coding**: exploit both visual and statistical redundancy.

Lossless component of lossy compression systems

- Almost every lossy compression system contains a lossless compression system

Lossy compression system



- We will discuss the basics of lossless compression first, then move on to lossy compression

Topics in lossless compression

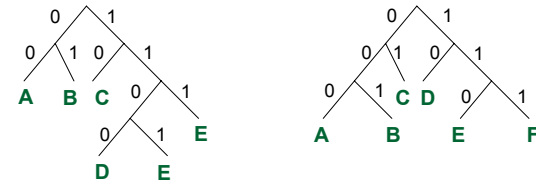
- Binary decision trees and variable length coding
- Entropy and bit rate
- Prefix codes, Huffman codes
- Joint entropy, conditional entropy, sources with memory
- Arithmetic coding
- Redundancy reduction by prediction
- Binary image compression

Example: 20 Questions

- *Alice* thinks of an outcome (from a finite set), but does not disclose her selection.
- *Bob* asks a series of yes-no questions to uniquely determine the outcome chosen. The goal of the game is to ask as few questions as possible **on average**.
- **Our goal:** Design the best strategy for *Bob*.

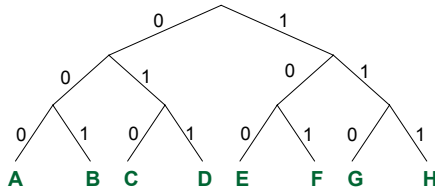
Example: 20 Questions (cont'd)

- **Observation:** The collection of questions and answers yields a binary code for each outcome.



- Which strategy (code) is better ?

Fixed length codes



- Average description length for K outcomes $l_{\text{ave}} = \log_2 K$
- Optimum for equally likely outcomes
- Verify by modifying tree

Variable length codes

- If outcomes are not equally probably
 - Use shorter descriptions for likely outcomes
 - Use longer descriptions for less likely outcomes
- Intuition
 - Optimum balanced code trees (with equally likely outcomes) can be pruned to yield unbalanced trees with unequal probabilities.
 - The resulting unbalanced code trees are also optimum
 - An outcome of probability p should require about

$$\log_2 \left(\frac{1}{p} \right) \text{ bits}$$

Zero-Memory Information Sources



- For a zero-memory source the source symbols are independent.
- Let S_i be some symbol that occurs with probability $P(S_i)$. If we receive S_i we have received
 $I(S_i) = -\log_2 [P(S_i)]$ bits
of information.
- The average number of information per symbol is

$$H(s) = \sum_i P(S_i) \log_2 \frac{1}{P(S_i)}$$

Entropy of a Source

$$H(S) = \sum_i Pr(S_i) \log_2 \frac{1}{Pr(S_i)}$$

- $H(S)$ is called the **entropy** of the source.
- It is the average amount of information per symbol.

Example:

$$S = \{S_1, S_2, S_3\} \quad \begin{aligned} Pr(S_1) &= 0.5 \\ Pr(S_2) &= Pr(S_3) = 0.25 \end{aligned}$$

$$H(S) = 0.5 \log_2 2 + 0.25 \log_2 4 + 0.25 \log_2 4$$

$$= 0.5 + 0.5 + 0.5 = \mathbf{1.5 \text{ bits/symbol}}$$

An Entropy Bound

- For a zero-memory source with q symbols

$$H(S) \leq \log_2 q$$

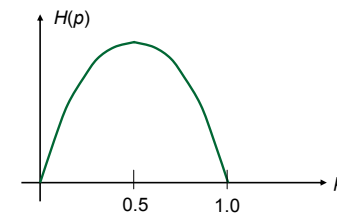
with equality if and only if $Pr(S_i) = \frac{1}{q}, \quad \forall i$

The maximum value of the entropy is achieved when the symbols are equiprobable.

Binary Sources

$$S = \{0, 1\} \quad \begin{aligned} Pr(0) &= p \\ Pr(1) &= 1 - p \end{aligned}$$

$$H(s) = p \log_2 \frac{1}{p} + (1 - p) \log_2 \frac{1}{1 - p} = H(p)$$



Entropy and bit-rate

- Consider an IID random process $\{X_n\}$ (or “source”) where each sample X_n (or “symbol”) possesses identical entropy $H(X)$.
- $H(X)$ is called the “entropy rate” of the random process.
- Noiseless Source Coding Theorem** (Shannon, 1948):
 - The entropy $H(X)$ is a lower bound for the average word length R of a decodable variable-length code for the symbols.
 - Conversely, the average word length R can approach $H(X)$, if sufficiently large blocks of symbols are encoded jointly.
- Redundancy of a code:**

$$\rho = R - H(X) \geq 0$$

Variable length codes

- Given an IID random process $\{X_n\}$ with alphabet A_X
- Task:** Assign a distinct code word, c_x , to each element $x \in A_X$, where c_x is a string of $\|c_x\|$ bits, such that each symbol x_n can be determined from a sequence of concatenated codewords c_{x_n} .
- Codes with the above property are said to be “**uniquely decodable**”.
- Prefix codes**
 - No code word is a prefix of any other codeword
 - Uniquely decodable, symbol by symbol, in natural order $0, 1, 2, \dots, n, \dots$

Example of non-decodable code

| α_i | code words |
|------------|------------|
| α_0 | 0 |
| α_1 | 01 |
| α_2 | 10 |
| α_3 | 11 |

Encode sequence of source symbols $\alpha_0, \alpha_2, \alpha_3, \alpha_0, \alpha_1$

Resulting bit-stream 0 10 11 0 01

Encode sequence of source symbols $\alpha_1, \alpha_0, \alpha_3, \alpha_0, \alpha_1$

Resulting bit-stream 01 0 11 0 01

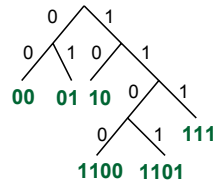
- Same bit-stream for different sequences of source symbols
- BTW: Not a prefix code.

Prefix codes are uniquely decodable

- Sequential Decoding Algorithm for Prefix Codes**
 - Match first code word c_{x_0} in alphabet against $\|c_{x_0}\|$ bits of the bit-stream.
 - If no match is found, consider next code word in alphabet and repeat step 1.
 - If a match is found for code word c_{x_n}
 - Generate decoder output x_n
 - Remove the leading $\|c_{x_n}\|$ bits from the bit-stream
 - Repeat steps 2 to 4.
 - If no match is found: ERROR
- Prefix condition guarantees that exactly one match exists.

Binary trees and prefix codes

- Every binary tree can be converted into a prefix code by traversing the tree from root to leaves.



A Tale of Two Codes

| Message | Probability | Code 1 | Code 2 |
|---------|-------------|--------|--------|
| S_1 | 0.25 | 00 | 0 |
| S_2 | 0.25 | 01 | 10 |
| S_3 | 0.25 | 10 | 110 |
| S_4 | 0.25 | 11 | 111 |

- Both are prefix codes
- Avg. # bits/symbol (code 1) = 2
- Avg. # bits/symbol (code 2) = 2.25.
- $H(S) = 2$ bits/symbol

Two Codes (continued)

- but if we change the distribution of the symbols...

| Message | Probability | Code 1 | Code 2 |
|---------|-------------|--------|--------|
| S_1 | 0.50 | 00 | 0 |
| S_2 | 0.25 | 01 | 10 |
| S_3 | 0.125 | 10 | 110 |
| S_4 | 0.125 | 11 | 111 |

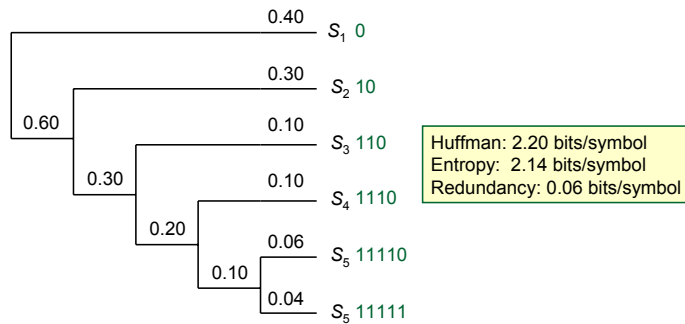
Avg. # bits/symbol (code 1) = 2
 Avg. # bits/symbol (code 2) = 1.75 = $H(S)$

To be efficient the code must exploit the probability distribution of the symbol set.

Huffman codes

- Design algorithm for variable length codes proposed by Huffman (1952) always finds a code with minimum redundancy.
- Obtain code tree as follows:
 - Pick the two symbols with lowest probabilities and merge them into a new auxiliary symbol.
 - Calculate the probability of the auxiliary symbol.
 - If more than one symbol remains, repeat steps 1 and 2 for the new auxiliary alphabet.
 - Convert the code tree into a prefix code.

Example



Redundancy of prefix code for general distribution

- Huffman code redundancy $0 \leq \rho < 1$ bit/symbol
- Theorem:** For any distribution p_X , a prefix code may be found, whose rate R satisfies

$$H(X) \leq R < H(X) + 1$$

- Proof
 - Left hand inequality: Shannon's noiseless coding theorem
 - Right hand inequality:

Choose word lengths $|c_x| = \lceil -\log_2 p_X(x) \rceil$
Resulting rate

$$R = \sum_x p_X(x) \lceil -\log_2 p_x(x) \rceil$$

$$< \sum_x p_X(x) (1 - \log_2 p_x(x))$$

$$= H(X) + 1$$

Vector Huffman coding

- Huffman coding very inefficient for $H(X) \ll 1$ bit/symbol.
- Remedy:
 - Combine m successive symbols to a new "block-symbol"
 - Huffman code the block symbols
 - Redundancy

$$H(X) \leq R < H(X) + \frac{1}{m}$$

- Can also be used to exploit statistical dependencies between successive symbols.
- Disadvantage: exponentially growing alphabet size

Example

- $S = \{a, b\}$
 - $\Pr(a) = 2/3$
 - $\Pr(b) = 1/3$

- Code A:
 - a 0
 - b 1

Average codeword length = 1 bit / symbol

Example (continued)

■ Code B

| | | |
|----|-----|-----------------|
| aa | 0 | $\Pr(aa) = 4/9$ |
| ab | 10 | $\Pr(ab) = 2/9$ |
| ba | 110 | $\Pr(ba) = 2/9$ |
| bb | 111 | $\Pr(bb) = 1/9$ |

$$\begin{aligned}\text{Average codeword length} &= 1(4/9) + 2(2/9) + 3(2/9) + 3(1/9) \\ &= 1.889 \text{ bits/(2 symbols)} \\ &= .944 \text{ bits/symbol}\end{aligned}$$

$$H(S) = .918 \text{ bits/symbol}$$