CT111 Intro to Communication Systems
Lecture 5: Data Compression

Yash M. Vasavada

Associate Professor, DA-IICT, Gandhinagar

10th Feb 2020

Overview of Today's Talk

Overview of Today's Talk

1. Compression of Digital Data
2. Discrete Memoryless Source

Overview of Today's Talk

1. Compression of Digital Data
2. Discrete Memoryless Source
3. Information Theoretic Aspects

Overview of Today's Talk

1. Compression of Digital Data
2. Discrete Memoryless Source
3. Information Theoretic Aspects
4. Practical Data Compression Schemes
   - Huffman Coding
   - Lempel-Ziv Coding

Source Coding

Solves the Problem of Data Compression

- Many digital data streams contain a lot of redundant information. For example, a digital image file may contain more zeros than ones: $0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1$
- We wish to squeeze out the redundant information to minimize the amount of data needed to be stored or transmitted
- Definition of Data Compression Problem:
  1. What are the good algorithms that achieve the maximal data compression?
  2. What is the maximum data compression that can be achieved if we want to recover the exact bit sequence after decompression?
- Importance of Data Compression Problem:
  - Cannot be overstated given so much data is getting uploaded/downloaded and stored in today's world of YouTube, Facebook, etc.

Source Coding

Data Compression Problem: Definitions

Source Alphabet: source is modeled as generating symbols each of which can take L symbols from a set $\{x_1, x_2, \ldots, x_L\}$. This set is the source alphabet.

- Output of the quantizer is binary ($L = 2$)
- However, in general (e.g., for emails, webpages, etc. written in English), L has to be at least equal to the number of letters ($= 26$) of English alphabet
- A common method for text encoding: ASCII character set ($L = 128$)

Source Coding

Data Compression Problem: Definitions

Source Alphabet: source is modeled as generating symbols each of which can take L symbols from a set $\{x_1, x_2, \ldots, x_L\}$. This set is the source alphabet.

- Output of the quantizer is binary ($L = 2$)
- However, in general (e.g., for emails, webpages, etc. written in English), L has to be at least equal to the number of letters ($= 26$) of English alphabet
- A common method for text encoding: ASCII character set ($L = 128$)

Symbol Probabilities: some of the source symbols are more likely than the others. Probability $P(X = x_k)$ of k^{th} symbol is denoted as p_k. Since the source alphabet is the universal set, $\sum_{k=1}^{K} p_k = 1$.

Source Coding

Data Compression Problem: Definitions

Source Alphabet: source is modeled as generating symbols each of which can take L symbols from a set $\{x_1, x_2, \ldots, x_L\}$. This set is the source alphabet.

- ▷ Output of the quantizer is binary ($L = 2$)
- ▷ However, in general (e.g., for emails, webpages, etc. written in English), L has to be at least equal to the number of letters ($= 26$) of English alphabet
- ▷ A common method for text encoding: ASCII character set ($L = 128$)

Symbol Probabilities: some of the source symbols are more likely than the others. Probability $P(X = x_k)$ of k^{th} symbol is denoted as p_k. Since the source alphabet is the universal set, $\sum_{k=1}^{K} p_k = 1$.

Discrete Memoryless Source (DMS): in DMS, each source symbol occurs independently of the other symbols. We will mostly assume that the source is DMS

→ With more complicated notations, the theory for DMS can be extended to correlated sources

Source Coding
for DMS

- A source code is a rule which maps symbols from the source alphabet to sequence of bits
- Fixed length source code maps source symbols into codewords of the same length:
  - → Example: $x_1 \Rightarrow 00, x_2 \Rightarrow 01, x_2 \Rightarrow 10, x_4 \Rightarrow 11$
- Variable length source code maps source symbols into codewords of different lengths:
  - → Example: $x_1 \Rightarrow 1, x_2 \Rightarrow 01, x_2 \Rightarrow 101, x_4 \Rightarrow 11$
- Variable length source code is uniquely decodeable if no codeword is a prefix of another codeword:
  - → Uniquely decodeable: $x_1 \Rightarrow 01, x_2 \Rightarrow 000, x_2 \Rightarrow 10, x_4 \Rightarrow 111$
  - → Counter-example: $x_1 \Rightarrow 1, x_2 \Rightarrow 00, x_2 \Rightarrow 01, x_4 \Rightarrow 10$. If we get 001001, should we decode it as x_2, x_1, $x_2\,x_1$ or as x_2, x_4, x_3?

We will work only with the prefix-free codes.

Source Coding

for DMS

- Rate R of a source code is the average number of bits required to represent a source symbol
- Let n_k be the number of bits in the codeword which represents a source symbol x_k.

Source Coding

for DMS

- Rate R of a source code is the average number of bits required to represent a source symbol
- Let n_k be the number of bits in the codeword which represents a source symbol x_k.

$\rightarrow$ Therefore, $R = \sum_{k=1}^{L} p_k n_k$

Source Coding
for DMS

- Rate R of a source code is the average number of bits required to represent a source symbol
- Let n_k be the number of bits in the codeword which represents a source symbol x_k.
  - → Therefore, $R = \sum_{k=1}^{L} p_k n_k$
  - → For fixed length codes, $R \geq \lceil \log_2 L \rceil$

Source Coding
for DMS

- Rate R of a source code is the average number of bits required to represent a source symbol
- Let n_k be the number of bits in the codeword which represents a source symbol x_k.
  - → Therefore, $R = \sum_{k=1}^{L} p_k n_k$
  - → For fixed length codes, $R \geq \lceil \log_2 L \rceil$
- **Goal of Source Encoding:** find a *uniquely decodeable* code that minimizes R

Source Coding
for DMS

- Rate R of a source code is the average number of bits required to represent a source symbol
- Let n_k be the number of bits in the codeword which represents a source symbol x_k.
  - → Therefore, $R = \sum_{k=1}^{L} p_k n_k$
  - → For fixed length codes, $R \geq \lceil \log_2 L \rceil$
- **Goal of Source Encoding:** find a *uniquely decodeable* code that minimizes R
- **Strategy:** assign short codewords (small n_k) to the most likely symbols (large p_k)

Kraft's Inequality

- For prefix free code, where individual codewords have lengths $n_1 \leq n_2 \leq \ldots \leq n_L$,

$$\sum_{\ell=1}^{L} 2^{-n_\ell} \leq 1$$

A Measure of Information Content of the Source

Entropy

- What is the information generated when the source emits k^{th} symbol x_k with probability p_k?
  - $\rightarrow$ Information should be inversely proportional to p_k
  - $\rightarrow$ Information, when measured in bits, should be a logarithmic function
- Information generated is therefore $\log_2\left(\frac{1}{p_k}\right) = -\log_2 p_k$ bits. This is called self-information of the source.

A Measure of Information Content of the Source

Entropy

- What is the information generated when the source emits k^{th} symbol x_k with probability p_k?
  - $\rightarrow$ Information should be inversely proportional to p_k
  - $\rightarrow$ Information, when measured in bits, should be a logarithmic function
- Information generated is therefore $\log_2\left(\frac{1}{p_k}\right) = -\log_2 p_k$ bits. This is called self-information of the source.
- What is R if each symbol gets encoded using the number of bits that equals its self-information?
  - $R = -\sum_{k=1}^{L} p_k \log_2 p_k$ bits

A Measure of Information Content of the Source

Entropy

- Entropy $H(X)$ is a measure of uncertainty in the source output, which is modeled as a random variable X
- $H(X)$ is defined as the average information generated by the source, where average is taken over the entire source alphabet
  - $\rightarrow$ Information generated by k^{th} symbol with probability of p_k: $n_k = -\log_2 p_k$ bits

A Measure of Information Content of the Source

Entropy

- Entropy $H(X)$ is a measure of uncertainty in the source output, which is modeled as a random variable X
- $H(X)$ is defined as the average information generated by the source, where average is taken over the entire source alphabet
  - → Information generated by k^{th} symbol with probability of p_k: $n_k = -\log_2 p_k$ bits
  - → Average of the information generated by the source:

$$H(X) = -\sum_{k=1}^{L} p_k \log_2 p_k, \quad \text{in bits}$$

A Measure of Information Content of the Source

Entropy

- Entropy $H(X)$ is a measure of uncertainty in the source output, which is modeled as a random variable X
- $H(X)$ is defined as the average information generated by the source, where average is taken over the entire source alphabet
  - → Information generated by k^{th} symbol with probability of p_k: $n_k = -\log_2 p_k$ bits
  - → Average of the information generated by the source:

$$H(X) = -\sum_{k=1}^{L} p_k \log_2 p_k, \quad \text{in bits}$$

  - → $H(X)$ is maximized if $p_k = \frac{1}{L}$

A Measure of Information Content of the Source

Entropy

- Entropy $H(X)$ is a measure of uncertainty in the source output, which is modeled as a random variable X
- $H(X)$ is defined as the average information generated by the source, where average is taken over the entire source alphabet
  - → Information generated by k^{th} symbol with probability of p_k: $n_k = -\log_2 p_k$ bits
  - → Average of the information generated by the source:

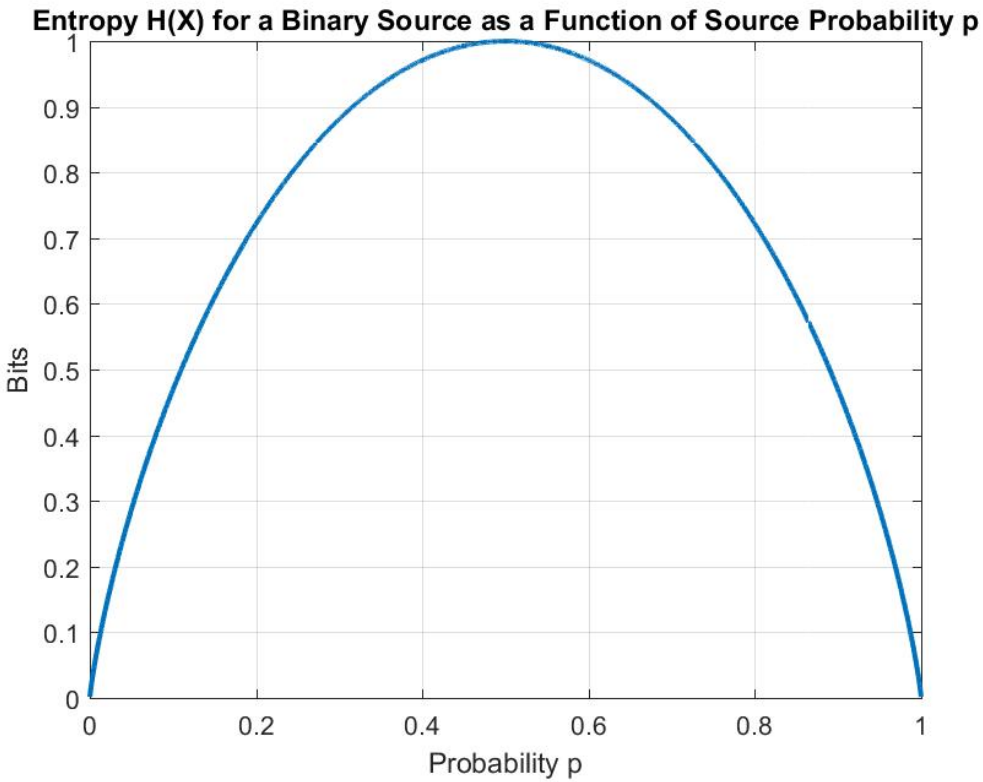
$$H(X) = -\sum_{k=1}^{L} p_k \log_2 p_k, \quad \text{in bits}$$

  - → $H(X)$ is maximized if $p_k = \frac{1}{L}$
  - → $H(x)$ is minimized if $p_k = 1$ for some k

A Measure of Information Content of the Source

Entropy for Binary RV

- For binary RV X for which $p_X(x) = \begin{cases} 1-p, & x=0, \\ p, & x=1 \end{cases}$, entropy $H(X)$ is given as $H(X) = -p\log_2 p - (1-p)\log_2(1-p)$



Source Coding Theorem

- *A fundamental theorem of the Information Theory [Shannon 1948 paper]:*
  - → For a uniquely decodeable source code that provides lossless compression (i.e., perfect reconstruction of the source symbols at the source decoder), $R \geq H(X)$.
    - ▷ Rate can be made arbitrarily close to, but not less than, $H(X)$

Source Coding Theorem

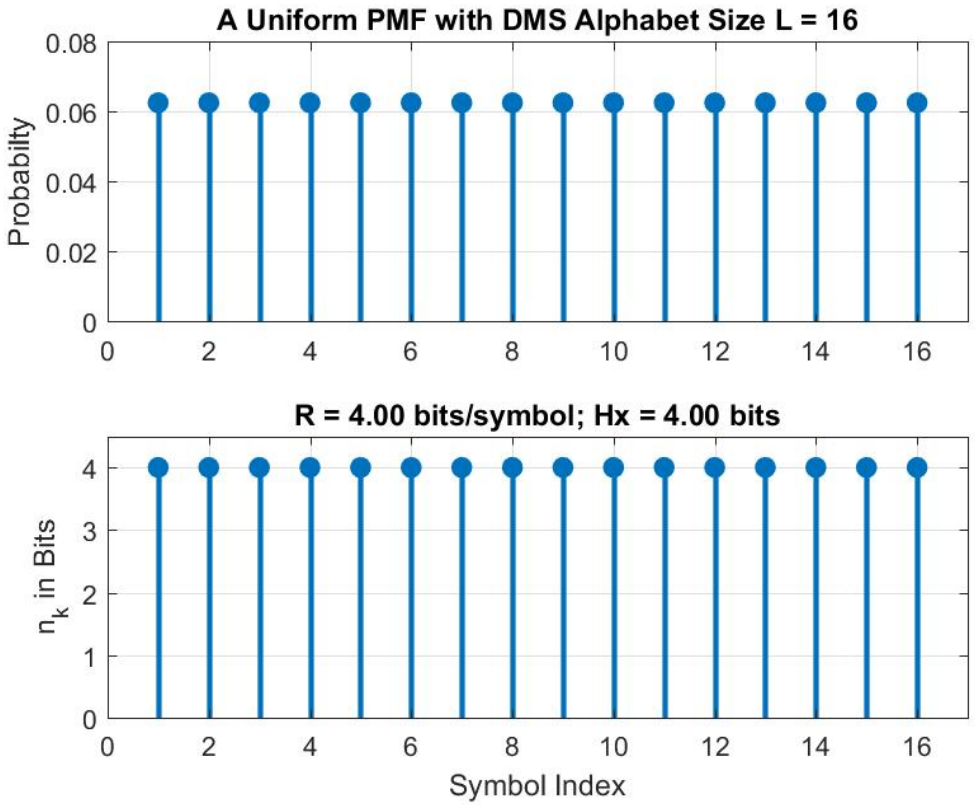
- *A fundamental theorem of the Information Theory [Shannon 1948 paper]:*
  - → For a uniquely decodeable source code that provides lossless compression (i.e., perfect reconstruction of the source symbols at the source decoder), $R \geq H(X)$.
    - ▷ Rate can be made arbitrarily close to, but not less than, $H(X)$
  - → This is the Source Coding (or Data Compression) Theorem

Source Coding Theorem

- *A fundamental theorem of the Information Theory [Shannon 1948 paper]:*
  - → For a uniquely decodeable source code that provides lossless compression (i.e., perfect reconstruction of the source symbols at the source decoder), $R \geq H(X)$.
    - ▷ Rate can be made arbitrarily close to, but not less than, $H(X)$
  - → This is the Source Coding (or Data Compression) Theorem
- How to *prove* this theorem?

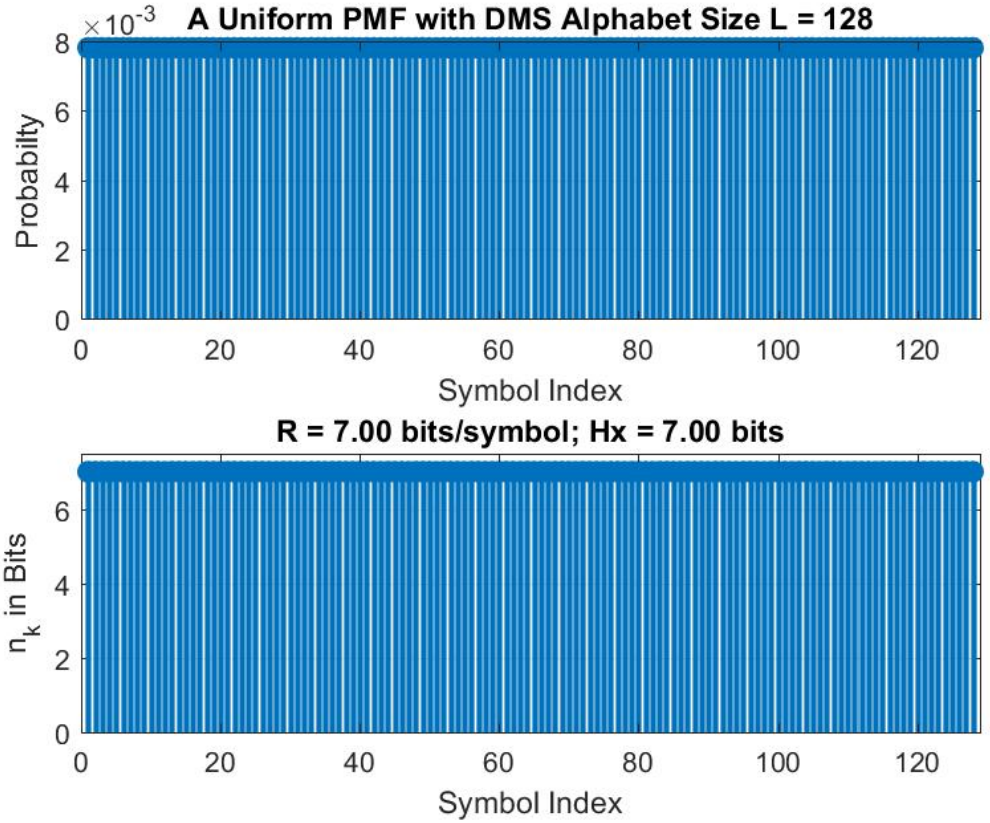
A Uniform PMF

- When the DMS alphabet size is $L = 2^n$, and its PMF is uniform, the probability of each letter is $p = 1/L = 2^{-n}$, and each letter is encoded using a fixed-length code of length $-\log_2(p) = n$ bits. Shown here when $n = 4$ bits.



A Uniform PMF

- Example of previous slide, shown when $n = 7$ bits. Probability p reduces, and hence the information content $-\log_2 p$ increases to 7 bits



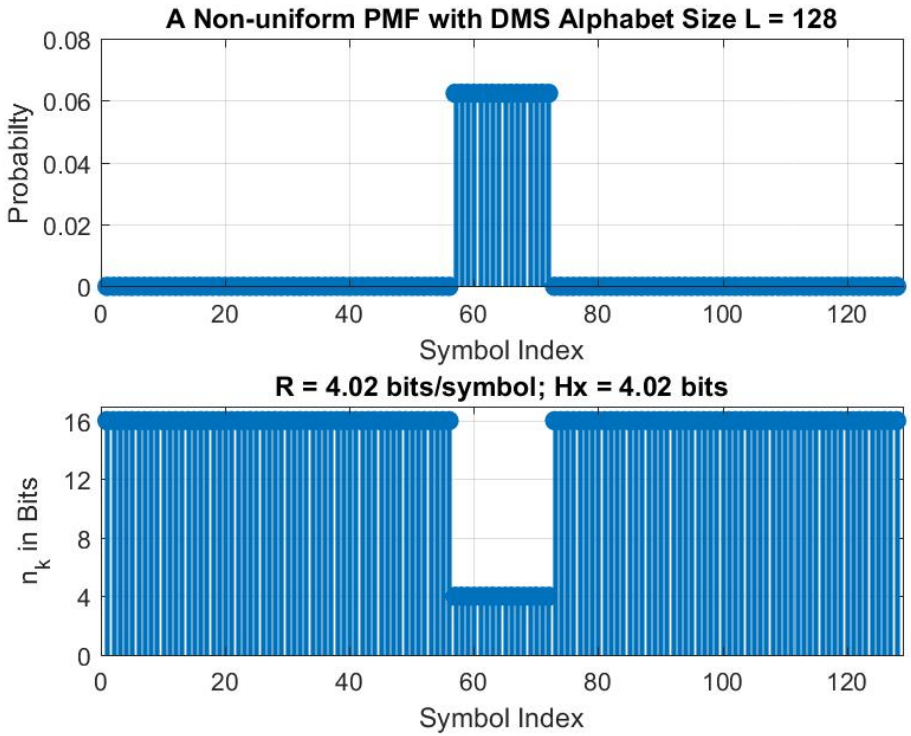
A Uniform PMF

- Thus, when the DMS has uniform PMF, and has a total of $L = 2^n$ symbols, we require n bits to represent each symbol
- *Cannot* use less than n bits and still represent all L symbols

A Non-uniform PMF

Data Compression Possible

- Suppose $L = 128$, however, only $M = 16$ symbols actually occur realistically. Remaining 112 symbols have vanishing probability of occurrence. In this case of non-uniform PMF, one needs $n = 4$ bits instead of 7 bits (using less than 4 bits will not be enough)



Proof

of the Source Coding Theorem (logical and not rigorously mathematical)

- An arbitrary DMS with non-uniform probabilities can be converted to a case in which most of its output sequences have vanishing probability and the probabilities of the remaining output sequences (that actually do occur) are all equal
- How?

Proof

of the Source Coding Theorem (logical and not rigorously mathematical)

- An arbitrary DMS with non-uniform probabilities can be converted to a case in which most of its output sequences have vanishing probability and the probabilities of the remaining output sequences (that actually do occur) are all equal
- How?
  - Unless the PMF is uniform distribution (all L symbols have equal probability of $1/L$), *encoding* a *sequence* of symbols is better compared to encoding one symbol at a time
  - As we increase the length of the sequence of symbols, the sequence is likely (with probability approaching 1) to be a member of the *typical set*
  - All the members of the typical set have identical probability

Proof

of the Source Coding Theorem (logical and not rigorously mathematical)

- An arbitrary DMS with non-uniform probabilities can be converted to a case in which most of its output sequences have vanishing probability and the probabilities of the remaining output sequences (that actually do occur) are all equal
- How?
  - Unless the PMF is uniform distribution (all L symbols have equal probability of $1/L$), *encoding* a *sequence* of symbols is better compared to encoding one symbol at a time
  - As we increase the length of the sequence of symbols, the sequence is likely (with probability approaching 1) to be a member of the *typical set*
  - All the members of the typical set have identical probability
- We have seen that the size of the typical set (for the binary alphabet) is $2^{M\,H_b(p)}$, which implies that we need no more and no less than $M\,H_b(p)$ bits to encode a sequence of length M bits
- Thus, the average number of bit required to represent a source symbol (in this case, a bit) is $R = H_b(p)$.
- This completes the proof of the source coding theorem

Practical Source Coding Algorithms

- There are two widely used source coding methods which make R approach the lower bound of $H(X)$ promised by data compression theorem:
  1. Huffman Coding: requires the knowledge of probabilities of DMS
  2. Lempel-Ziv Coding: operates in a *blind* manner (i.e., without requiring the knowledge of the DMS' PMF)

Source Coding Theorem

Huffman Coding Algorithm

Huffman coding is based upon the idea of allocating short codewords to highly probable symbols. The algorithm for Huffman coding belongs to a class of algorithms based upon the Greedy Approach.

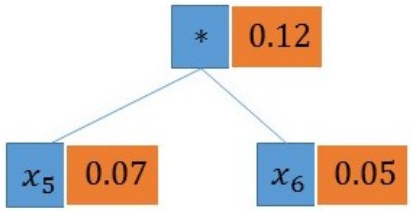
1. List all symbols in descending order of probability
2. Group the two lowest probability symbols into a new combined symbol
3. Repeat steps 1 and 2 until only a single combined symbol remains
4. Assign a 1 to each upper branch and 0 to each lower branch of the resulting binary tree
5. Codeword for each symbol is the sequence of bits from the root of the tree to that symbol

Source Coding Theorem

Huffman Coding Algorithm: An Example

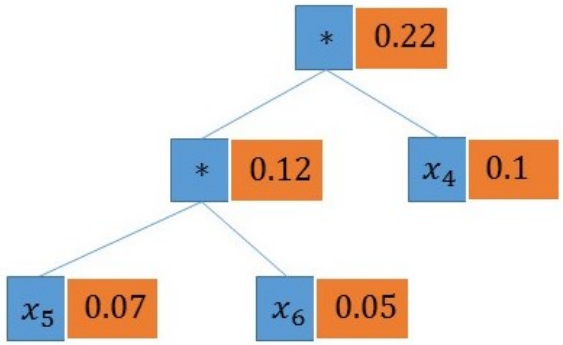
1

Symbol	Probability
x_1	0.45
x_2	0.2
x_3	0.13
x_4	0.1
x_5	0.07
x_6	0.05



2

Symbol	Probability
x_1	0.45
x_2	0.2
x_3	0.13
*	0.12
x_4	0.1

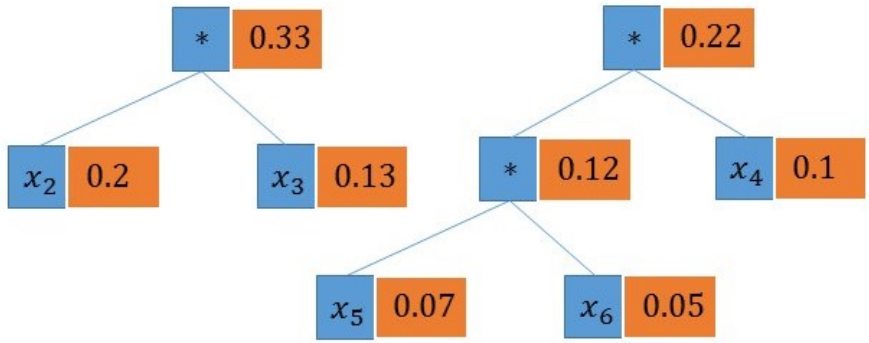


Source Coding Theorem

Huffman Coding Algorithm: An Example

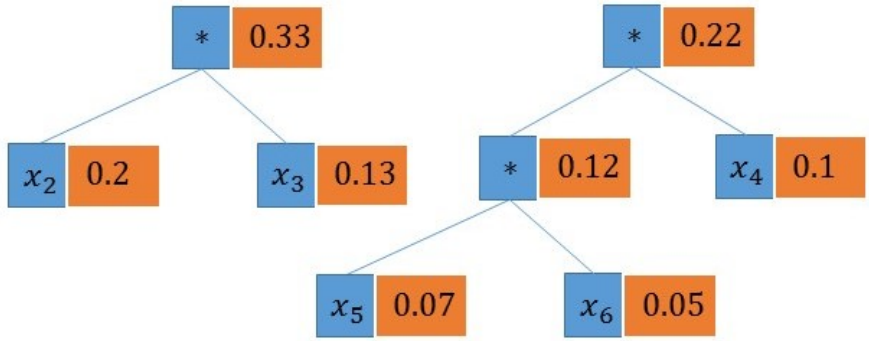
3

Symbol	Probability
x_1	0.45
$*$	0.22
x_2	0.2
x_3	0.13



4

Symbol	Probability
x_1	0.45
$*$	0.33
$*$	0.22

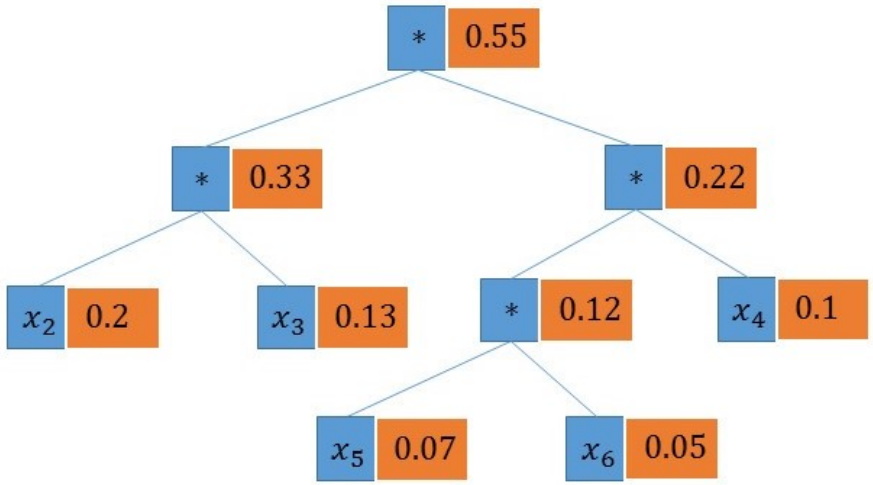


Source Coding Theorem

Huffman Coding Algorithm: An Example

5

Symbol	Probability
$*$	0.55
x_1	0.45

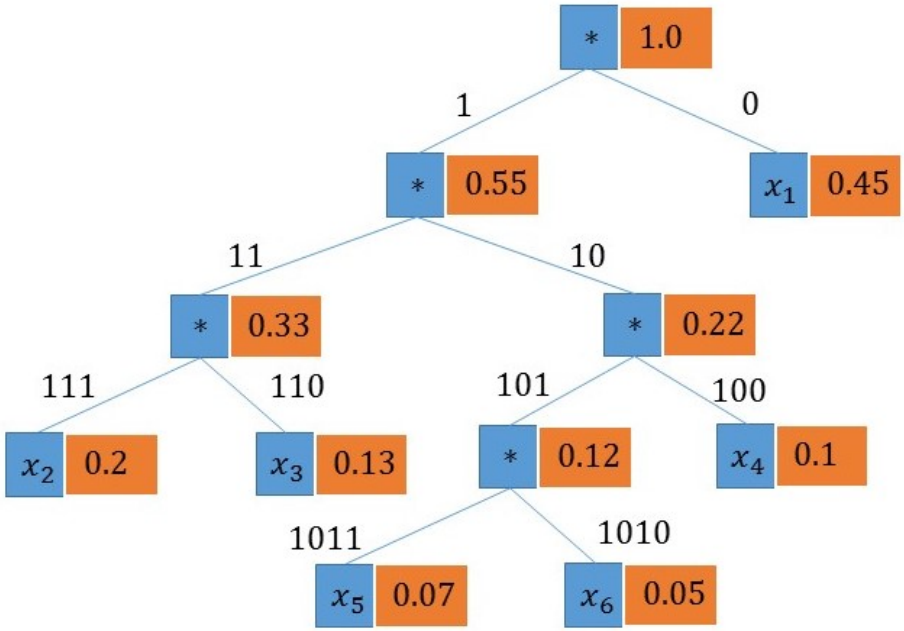


Source Coding Theorem

Huffman Coding Algorithm: An Example

6

Symbol	Probability	Codeword
x_1	0.45	0
x_2	0.2	111
x_3	0.13	110
x_4	0.1	100
x_5	0.07	1011
x_6	0.05	1010



Source Coding Theorem

Huffman Coding Algorithm: Numerical Analysis

Symbol	Probability p_k	Codeword	Length L_k	$p_k L_k$	$-log_2 p_k$	$-p_k log_2 p_k$
x_1	0.45	0	1	0.45	1.15	0.518
x_2	0.2	111	3	0.6	2.32	0.464
x_3	0.13	110	3	0.39	2.94	0.382
x_4	0.1	100	3	0.3	3.32	0.332
x_5	0.07	1011	4	0.28	3.83	0.268
x_6	0.05	1010	4	0.2	4.32	0.216
				2.2		2.18

Source Coding Theorem

Huffman Coding Algorithm: Summary

- How do we get even closer to Entropy?
  - → Use Huffman Algorithm on pairs of symbols, or
  - → Even better, on triplets of symbols, or quadruplets or,
  - → In general, m-tuple of symbols.
- A limitation of Huffman Algorithm is that it requires the source symbol probabilities to be known. These are not known exactly but they can be estimated from a sufficiently large sample of source output. Mismatch between the estimated and exact symbol probabilities contributes to a degraded performance of Huffman's Algorithm.
- Another limitation of Huffman Code is that the symbol probabilities are assumed to be fixed. If these probabilities change with time, Huffman Code may provide suboptimal data compression.

Lempel-Ziv Coding

Encoding Algorithm

- In Lempel-Ziv coding, the source sequence is sequentially parsed into strings that have not appeared so far.
  - → Continually parse the symbol sequence and insert a comma on detecting the shortest sequence that has not been seen earlier
- Example sequence: 1011010100010...
- Sequential parsing method in Lempel-Ziv encoding:
  - → 1,011010100010...
  - → 1,0,11010100010...
  - → 1,0,11,010100010...
  - → 1,0,11,01,0100010...
  - → 1,0,11,01,010,0010...
  - → 1,0,11,01,010,00,10...
  - → 1,0,11,01,010,00,10,...

Lempel-Ziv Coding

Encoding Algorithm

- Prefix of each "phrase" (phrase is the symbol string between two commas), i.e., the string consisting of all but the last symbol of each phrase, must have occurred earlier.
- Encode this phrase by giving the location of the prefix and the value of the last bit.
- Let $C(n)$ be the number of phrases in the parsing of the input sequence with n symbols.
- Thus, $\log C(n)$ bits are needed to describe the location of the prefix to the phrase and one bit to describe the last bit.
- In the above example, the code sequence is:
  $(000, 1), (000, 0), (001, 1), (010, 1), (100, 0), (010, 0), (001, 0)$
  - $\rightarrow$ First number of each pair gives the index of the prefix and the second number gives the last bit of the phrase.