Optimal Variable-Length Codes

<table>
<thead>
<tr>
<th>$a_k$</th>
<th>$p_k$</th>
<th>codewords</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.16</td>
<td>111</td>
</tr>
<tr>
<td>b</td>
<td>0.04</td>
<td>0001</td>
</tr>
<tr>
<td>c</td>
<td>0.04</td>
<td>0000</td>
</tr>
<tr>
<td>d</td>
<td>0.16</td>
<td>110</td>
</tr>
<tr>
<td>e</td>
<td>0.23</td>
<td>01</td>
</tr>
<tr>
<td>f</td>
<td>0.07</td>
<td>1001</td>
</tr>
<tr>
<td>g</td>
<td>0.06</td>
<td>1000</td>
</tr>
<tr>
<td>h</td>
<td>0.09</td>
<td>001</td>
</tr>
<tr>
<td>i</td>
<td>0.15</td>
<td>101</td>
</tr>
</tbody>
</table>
Unique Decodability

- Necessary condition: Kraft-McMillan inequality for codeword lengths: \[ \sum_k 2^{-\ell_k} \leq 1 \]
- Sufficient condition: Prefix codes (can be represented as binary trees)

Prefix Codes

- Uniquely and **instantaneously decodable**, simple encoding and decoding (e.g., via binary tree)
- Can always construct prefix code for codeword lengths that satisfy Kraft-McMillan inequality

\[ \rightarrow \text{There are no better uniquely decodable codes than the best prefix codes} \]

Average Codeword Length and Entropy

- Average codeword length for pmf \( p \):
  \[ \bar{\ell} = \sum_k p_k \ell_k \] (efficiency measure for lossless codes)
- Entropy of a source with pmf \( p \):
  \[ H(p) = - \sum_k p_k \log_2 p_k \]
- Entropy = lower bound for \( \bar{\ell} \):
  \[ \bar{\ell} \geq H(p) \]
- Can always construct prefix code with
  \[ H(p) \leq \bar{\ell} < H(p) + 1 \] (e.g., Shannon code)
Shannon Code

\[ \bar{\ell} \geq H(p) \]

\[ \sum_k p_k \ell_k \geq - \sum_k p_k \log_2 p_k \]

⇒ equality if and only if \( \ell_k = - \log_2 p_k \)

⇒ only possible if: \( \forall k, p_k = 2^{-n_k} \) with \( n_k \in \mathbb{Z} \)

Shannon Code:

⇒ Round codeword lengths to next integer: \( \ell_k = \lceil - \log_2 p_k \rceil \)

⇒ Construct prefix code with these codeword lengths (always possible)

⇒ Typically not optimal (structural redundancies)

Example: Shannon Code

<table>
<thead>
<tr>
<th>( a )</th>
<th>( p_k )</th>
<th>( \ell_k = \lceil - \log_2 p_k \rceil )</th>
<th>codewords</th>
<th>( H \approx 2.171 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>0.40</td>
<td>2 = ([1.32...])</td>
<td>00</td>
<td>( \bar{\ell} = 2.6 )</td>
</tr>
<tr>
<td>( e )</td>
<td>0.15</td>
<td>3 = ([2.73...])</td>
<td>010</td>
<td>( \varrho \approx 19.8% )</td>
</tr>
<tr>
<td>( i )</td>
<td>0.15</td>
<td>3 = ([2.73...])</td>
<td>011</td>
<td>structural redundancy!</td>
</tr>
<tr>
<td>( o )</td>
<td>0.15</td>
<td>3 = ([2.73...])</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>( u )</td>
<td>0.15</td>
<td>3 = ([2.73...])</td>
<td>101</td>
<td></td>
</tr>
</tbody>
</table>

Note: Removing the redundant bit would yield \( \bar{\ell} = 2.3, \varrho \approx 5.9\% \) (still not optimal)
Shannon-Fano Code: Construct Full Binary Code Tree

1. Sort symbols in alphabet according to their probability masses.
2. Divide sorted list into two groups $A$ and $B$, so that $P(A)$ is as close to $P(B)$ as possible.
3. Create a node and assign the first group $A$ to one branch and the other group $B$ to the other branch.
4. Recursively apply steps 2 and 3 to the groups $A$ and $B$ until all symbols are assigned to terminal nodes.

→ No Guarantee for Optimality (but yields prefix code without structural redundancy)

Example: Shannon-Fano Code

<table>
<thead>
<tr>
<th>$a_k$</th>
<th>$p_k$</th>
<th>codewords</th>
<th>better code</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>0.40</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>$e$</td>
<td>0.15</td>
<td>01</td>
<td>100</td>
</tr>
<tr>
<td>$i$</td>
<td>0.15</td>
<td>10</td>
<td>101</td>
</tr>
<tr>
<td>$o$</td>
<td>0.15</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>$u$</td>
<td>0.15</td>
<td>111</td>
<td>111</td>
</tr>
</tbody>
</table>

$H \approx 2.171$  
$\bar{\ell} = 2.3$  
$\bar{\ell} = 2.2$  
$\varrho \approx 5.9\%$  
$\varrho \approx 1.3\%$

<table>
<thead>
<tr>
<th>$a_k$</th>
<th>$p_k$</th>
<th>Shannon</th>
<th>Shannon-Fano</th>
<th>optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.40</td>
<td>00</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>e</td>
<td>0.15</td>
<td>010</td>
<td>01</td>
<td>100</td>
</tr>
<tr>
<td>i</td>
<td>0.15</td>
<td>011</td>
<td>10</td>
<td>101</td>
</tr>
<tr>
<td>o</td>
<td>0.15</td>
<td>100</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>u</td>
<td>0.15</td>
<td>101</td>
<td>111</td>
<td>111</td>
</tr>
</tbody>
</table>

$H \approx 2.171$  
$\bar{\ell} = 2.6$  
$\varrho \approx 19.8\%$  
$\bar{\ell} = 2.3$  
$\varrho \approx 5.9\%$  
$\bar{\ell} = 2.2$  
$\varrho \approx 1.3\%$

**Question**

Is there a low-complex algorithm for constructing optimal prefix codes?

**Shannon code**

$\ell_k = \lceil - \log_2 p_k \rceil$

![Shannon code diagram]

**Shannon-Fano code**

recursive equal probability split

![Shannon-Fano code diagram]

**optimal code**

? test all full binary trees with given number of terminal nodes

![Optimal code diagram]
Optimal Prefix Codes / Optimality of Prefix Codes

Optimal Lossless Codes

Optimal Prefix Code

- Any prefix code that achieves the minimum possible average codeword length $\bar{\ell}$ for a given pmf.
- Each optimal prefix code is also an optimal uniquely decodable code.

Construction of Optimal Prefix Codes?

- Finite alphabets: Huffman algorithm (1952) yields a prefix code with minimum redundancy.
Consider any prefix code for an alphabet $A$ which includes two letters $a$ and $b$ with associated probabilities $p_a$ and $p_b$

- $a$ and $b$ are associated with codewords of lengths $\ell_a$ and $\ell_b$

What happens if we exchange the codewords for $a$ and $b$?

→ Obtain a new prefix code
→ Average codeword length changes from $\bar{\ell}$ to $\bar{\ell}_{\text{new}}$

\[
\bar{\ell}_{\text{new}} = \bar{\ell} - (p_a \ell_a + p_b \ell_b) + (p_a \ell_b + p_b \ell_a)
\]
\[
= \bar{\ell} - p_a (\ell_a - \ell_b) + p_b (\ell_a - \ell_b)
\]
\[
= \bar{\ell} - (p_a - p_b)(\ell_a - \ell_b)
\]

→ Different cases (for $p_a \geq p_b$):

1. $p_a = p_b$ or $\ell_a = \ell_b$: $\Rightarrow \bar{\ell}_{\text{new}} = \bar{\ell}$
2. $p_a > p_b$ and $\ell_a < \ell_b$: $\Rightarrow \bar{\ell}_{\text{new}} > \bar{\ell}$
3. $p_a > p_b$ and $\ell_a > \ell_b$: $\Rightarrow \bar{\ell}_{\text{new}} < \bar{\ell}$

In an optimal prefix code, we require

\[\forall a, b: \quad p_a > p_b \quad \Rightarrow \quad \ell_a \leq \ell_b\]
Lemma (class of optimal prefix codes)

For any finite alphabet $A$, there exists an optimal prefix code $C$ with the following property:

There are two codewords that have the maximum length, differ only in the final bit, and correspond to the two least likely alphabet letters.

Proof

1. For each codeword of maximum length, the code includes a codeword of the same length that differs only in the final bit.

In the binary tree representation, this means that the corresponding terminal node has a sibling.

Codes without that property cannot be optimal, since a removal of the last bit of the considered codeword of maximum length would reduce the average codeword length without violating the prefix property.

$$
\bar{\ell}_{\text{new}} = \bar{\ell} - p_a \cdot \ell_a + p_a \cdot (\ell_a - 1) \\
= \bar{\ell} - p_a \\
< \bar{\ell}
$$
Lemma (class of optimal prefix codes)

For any finite alphabet $\mathcal{A}$, there exists an optimal prefix code $\mathcal{C}$ with the following property:

There are two codewords that have the maximum length, differ only in the final bit, and correspond to the two least likely alphabet letters.

Proof

2 Two of the codewords of maximum length correspond to the two least likely alphabet letters.

For any two alphabet letters $a$ and $b$ with $p_a > p_b$, the codeword length must satisfy $\ell_a \leq \ell_b$. Otherwise, an exchange of the codewords would reduce the average codeword length without violating the prefix property.

Note: For two alphabet letters $a$ and $b$ with $p_a = p_b$, an exchange of the codewords does not modify the average codeword length.

$$\bar{\ell}_{\text{new}} = \bar{\ell} - (p_a - p_b)(\ell_a - \ell_b)$$

$\Rightarrow$ $p_a > p_b, \ell_a > \ell_b : \bar{\ell}_{\text{new}} < \bar{\ell}$

$\Rightarrow$ $p_a = p_b : \bar{\ell}_{\text{new}} = \bar{\ell}$
Lemma (class of optimal prefix codes)

For any finite alphabet \( A \), there exists an optimal prefix code \( C \) with the following property:

There are two codewords that have the maximum length, differ only in the final bit, and correspond to the two least likely alphabet letters.

Proof

3. Two codewords of maximum length that differ only in the last bit are assigned to the two least likely alphabet letters.

Not necessary for optimality.

But we can always exchange any two codewords of maximum length (i.e., the same length \( \ell_{\text{max}} \)) without impacting the average codeword length.

\[
\bar{\ell}_{\text{new}} = \bar{\ell} - (p_a - p_b)(\ell_{\text{max}} - \ell_{\text{max}})
\]

\[
= \bar{\ell}
\]

\[\Rightarrow \text{There exists an optimal prefix code } C \text{ with the above stated property.}\]
The Huffman Algorithm

Idea for Construction of Binary Code Tree

- Consider optimal prefix codes for which the two least likely symbols correspond to codewords of maximum length that differ only in the final bit
- Choose the two least likely symbols and create a parent node
- Repeat the procedure with the reduced alphabet

Huffman Algorithm (via construction of binary code tree)

1. Select the two letters \( a \) and \( b \) with the smallest probabilities \( p_a \) and \( p_b \)
2. Create a parent node for the two letters \( a \) and \( b \) in the binary code tree
3. Replace the letters \( a \) and \( b \) with a new letter with probability \( p_a + p_b \)
4. If the resulting new alphabet contains more than a single letter, repeat all previous steps with this alphabet
5. Convert the obtained binary code tree into a prefix code
Example: Construction of a Huffman Code

\[ \bar{\ell} = 2.98 \]

\[ H(p) \approx 2.9405 \]

\[ \varrho \approx 0.0395 \ (1.34\%) \]

<table>
<thead>
<tr>
<th>( a_k )</th>
<th>( p_k )</th>
<th>codewords</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.16</td>
<td>111</td>
</tr>
<tr>
<td>b</td>
<td>0.04</td>
<td>0001</td>
</tr>
<tr>
<td>c</td>
<td>0.04</td>
<td>0000</td>
</tr>
<tr>
<td>d</td>
<td>0.16</td>
<td>110</td>
</tr>
<tr>
<td>e</td>
<td>0.23</td>
<td>01</td>
</tr>
<tr>
<td>f</td>
<td>0.07</td>
<td>1001</td>
</tr>
<tr>
<td>g</td>
<td>0.06</td>
<td>1000</td>
</tr>
<tr>
<td>h</td>
<td>0.09</td>
<td>001</td>
</tr>
<tr>
<td>i</td>
<td>0.15</td>
<td>101</td>
</tr>
</tbody>
</table>

\[ H(p) \approx 2.9405 \]

\[ \varrho \approx 0.0395 \ (1.34\%) \]
Average Codeword Length of Huffman Codes

All codes obtained by the Huffman algorithm are optimal prefix codes for the given pmf

- Rigorous proof (by induction) can be found in [Cover, Thomas: “Elements of Information Theory”]
- There are multiple Huffman codes (labeling of branches, same probability for multiple nodes)
- There might be optimal prefix codes that cannot be obtained by the Huffman algorithm

 Bounds on Average Codeword Length

- Entropy is a lower bound for all lossless codes: \( \bar{\ell} \geq H(p) \)
- We showed that Shannon code has property: \( \bar{\ell}_{\text{Shannon}} < H(p) + 1 \)
- Huffman codes are optimal uniquely decodable codes: \( \bar{\ell}_{\text{opt}} \leq \bar{\ell}_{\text{Shannon}} \)

⇒ Average codeword length of Huffman codes (optimal codes) is bounded by

\[ H(p) \leq \bar{\ell}_{\text{opt}} < H(p) + 1 \]
Unix File Compression Utility “Pack”

Basic principle
- Design Huffman code for bytes of file to be compressed
- Transmit codeword table as part of the bitstream

Encoding
- Determine relative frequencies of bytes in input file
- Generate Huffman code for these statistics
- Code “Huffman tree” at the beginning of bitstream
- Code bytes of input file using Huffman code

Decoding
- Decode “Huffman tree” from the compressed bitstream
- Read variable-length codewords and output bytes

syntax of bitstream
- 16 bits magic header ("1f 1e")
- 32 bits number of symbols
- 8 bits depth of Huffman tree
- 8 bits number of terminal nodes in level 1
- 8 bits number of terminal nodes in level 2
  ...
- 8 bits character for 1-st terminal node
- 8 bits character for 2-nd terminal node
  ...

sequence of codewords
Huffman Coding in Practice

- Huffman coding is used in many compression tools and standards.
- There are basically two variants:

1. **Sending Huffman tables as part of the bitstream**
   - Unix file compression utility **pack**
   - Image compression standard **JPEG**

2. **Huffman tables are fixed in standard**
   - Audio coding standard **MP3**
   - Video coding standards **MPEG-2 Video, H.263, MPEG-4 Visual, H.264 | AVC**
Conditional Codes / Conditional Huffman Codes

Lossless Coding of Sources with Memory

Example: Stationary Markov Process

- Consider stationary Markov source \( S = \{ S_n \} \) with symbol alphabet \( A = \{ a, b, c \} \)
- Statistically properties are completely specified by conditional pmf \( p(x \mid y) = P(S_n = x \mid S_{n-1} = y) \)

<table>
<thead>
<tr>
<th>( x )</th>
<th>( p(x \mid a) )</th>
<th>( p(x \mid b) )</th>
<th>( p(x \mid c) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>0.90</td>
<td>0.15</td>
<td>0.25</td>
</tr>
<tr>
<td>( b )</td>
<td>0.05</td>
<td>0.80</td>
<td>0.15</td>
</tr>
<tr>
<td>( c )</td>
<td>0.05</td>
<td>0.05</td>
<td>0.60</td>
</tr>
</tbody>
</table>

Calculate marginal pmf

<table>
<thead>
<tr>
<th>( x )</th>
<th>( p(x) )</th>
<th>Huffman</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>29/45</td>
<td>1</td>
</tr>
<tr>
<td>( b )</td>
<td>11/45</td>
<td>01</td>
</tr>
<tr>
<td>( c )</td>
<td>5/45</td>
<td>00</td>
</tr>
</tbody>
</table>

There are dependencies between successive source symbols!

Can we exploit these dependencies for improving the coding efficiency?

**Idea:**
- Design Huffman code for each condition \( (S_{n-1} = y) \)
- Select codeword table for a symbol \( s_n \) based on previous symbol \( s_{n-1} \)

\[ H(S_n) \approx 1.2575 \]

\[ \bar{\ell} \approx 1.3556 \ (61/45) \]

\[ \rho \approx 0.0980 \ (7.8\%) \]
### Conditional Huffman Codes

**conditional Huffman code** (assume "$S_{n-1} = a$" for first symbol)

<table>
<thead>
<tr>
<th>$S_{n-1} = a$</th>
<th>$S_{n-1} = b$</th>
<th>$S_{n-1} = c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>$p(x \mid a)$</td>
<td>codeword</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>$a$</td>
<td>0.90</td>
<td>0</td>
</tr>
<tr>
<td>$b$</td>
<td>0.05</td>
<td>10</td>
</tr>
<tr>
<td>$c$</td>
<td>0.05</td>
<td>11</td>
</tr>
</tbody>
</table>

$$\bar{\ell}(S_{n-1}=a) = 1.1 \quad \bar{\ell}(S_{n-1}=b) = 1.2 \quad \bar{\ell}(S_{n-1}=c) = 1.4$$

**Example:**
- **Encoding:** "abba" ➔ "010010"
- **Decoding:** "010010" ➔ "abba"

Average codeword length $\bar{\ell}_{\text{cond}}$ for conditional Huffman code

$$\bar{\ell}_{\text{cond}} = \sum_{y \in A} p(y) \cdot \bar{\ell}(S_{n-1}=y) = \frac{29}{45} \cdot 1.1 + \frac{11}{45} \cdot 1.2 + \frac{5}{45} \cdot 1.4 = \frac{521}{450} \approx 1.1578$$

**Better than scalar Huffman code:** $\bar{\ell}_{\text{cond}} < \bar{\ell}_{\text{scal}}$

**We also have:** $\bar{\ell}_{\text{cond}} < H(S_n)$ ➔ **What’s wrong?**

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$29/45$</td>
<td>0</td>
</tr>
<tr>
<td>$b$</td>
<td>$11/45$</td>
<td>10</td>
</tr>
<tr>
<td>$c$</td>
<td>$5/45$</td>
<td>11</td>
</tr>
</tbody>
</table>

$\bar{\ell}_{\text{scal}} = 61/45 \approx 1.3556$

$H(S_n) \approx 1.2575$
Consider an individual condition “\(S_{n-1} = y\)” (i.e., fixed value \(y\))

- Code is constructed for the conditional probability masses \(p(x \mid y) = P(S_n = x \mid S_{n-1} = y)\)
- Average codeword length \(\bar{\ell}(S_{n-1} = y)\) for given condition “\(S_{n-1} = y\)” is bounded by

\[
\left( - \sum_{x} p(x \mid y) \cdot \log_2 p(x \mid y) \right) \leq \bar{\ell}(S_{n-1} = y) < \left( - \sum_{x} p(x \mid y) \cdot \log_2 p(x \mid y) \right) + 1
\]

\[
H(S_n \mid S_{n-1} = y) \leq \bar{\ell}(S_{n-1} = y) < H(S_n \mid S_{n-1} = y) + 1
\]

- \(H(S_n \mid S_{n-1} = y)\) is referred to as conditional entropy given the event \(S_{n-1} = y\)

Average codeword length \(\bar{\ell}_{\text{cond}}\) for conditional Huffman coding

- Each condition “\(S_{n-1} = y\)” occurs with probability \(p(y) = P(S_{n-1} = y) = P(S_n = y)\)

\[
\left( \sum_{y} p(y) \cdot H(S_n \mid S_{n-1} = y) \right) \leq \left( \sum_{y} p(y) \cdot \bar{\ell}(S_{n-1} = y) \right) < \left( \sum_{y} p(y) \cdot H(S_n \mid S_{n-1} = y) \right) + \left( \sum_{y} p(y) \cdot 1 \right)
\]

\[
H(S_n \mid S_{n-1}) \leq \bar{\ell}_{\text{cond}} < H(S_n \mid S_{n-1}) + 1
\]
Conditional Entropy

Conditional entropy of $S_n$ given $S_{n-1}$

$$H(S_n \mid S_{n-1}) = \sum_{\forall y} p(y) \cdot H(S_n \mid S_{n-1} = y)$$

$$= \sum_{\forall y} p(y) \left( -\sum_{\forall x} p(x \mid y) \log_2 p(x \mid y) \right)$$

$$= -\sum_{\forall x, y} p(x, y) \log_2 p(x \mid y)$$

$$\Rightarrow H(S_n \mid S_{n-1}) = \mathbb{E}\left\{ -\log_2 p(S_n \mid S_{n-1}) \right\}$$

with $H(S_n \mid S_{n-1} = y) = -\sum_{\forall x} p(x \mid y) \log_2 p(x \mid y)$

remember: $p(x, y) = p(x \mid y) p(y)$

similarly: $H(S_n) = \mathbb{E}\left\{ -\log_2 p(S_n) \right\}$

Conditional Huffman Codes

$$H(S_n \mid S_{n-1}) \leq \bar{\ell} < H(S_n \mid S_{n-1}) + 1$$

Scalar Huffman Codes

$$H(S_n) \leq \bar{\ell} < H(S_n) + 1$$
Generalized Conditional Coding

Usage of two or more preceding symbols

- Design a Huffman code for each condition: "$S_{n-1} = a, S_{n-2} = b, \cdots$"

- Bounds on average codeword length:
  
  \[ H(S_n \mid S_{n-1}, S_{n-2}, \cdots) < \ell < H(S_n \mid S_{n-1}, S_{n-2}, \cdots) + 1 \]

- Conditional entropy of $S_n$ given $S_{n-1}, S_{n-2}, \cdots$

  \[ H(S_n \mid S_{n-1}, S_{n-2}, \cdots) = E\left\{ -\log_2 p(S_n \mid S_{n-1}, S_{n-2}, \cdots) \right\} = -\sum_{x, y_1, y_2, \cdots} p(x, y_1, y_2, \cdots) \cdot \log_2 p(x \mid y_1, y_2, \cdots) \]

Arbitrary function of preceding symbols

- Design a Huffman code for possible value of: $C = f(S_{n-1}, S_{n-2}, \cdots)$

- Bounds on average codeword length:

  \[ H(S_n \mid C) < \ell < H(S_n \mid C) + 1 \]

- Conditional entropy of $S_n$ given $C = f(S_{n-1}, S_{n-2}, \cdots)$

  \[ H(S_n \mid C) = E\left\{ -\log_2 p(S_n \mid C) \right\} = -\sum_{x, c} p_{SC}(x, c) \cdot \log_2 p_{S \mid C}(x \mid c) \]
### Conditioning Does Not Increase Entropy

Consider general case with condition $C = f(S_{n-1}, S_{n-2}, \cdots)$

\[
H(S_n \mid C) = - \sum_{s,c} p_{SC}(s,c) \log_2 p_{S \mid C}(s \mid c)
\]

\[
= - \sum_{s,c} p_{SC}(s,c) \log_2 \left( \frac{p_{SC}(s,c)}{p_C(c)} \cdot \frac{p_S(s)}{p_S(s)} \right)
\]

\[
= - \sum_{s,c} p_{SC}(s,c) \log_2 p_S(s) - \sum_{s,c} p_{SC}(s,c) \log_2 \left( \frac{p_{SC}(s,c)}{p_C(c) p_S(s)} \right)
\]

\[
= - \sum_s p_S(s) \log_2 p_S(s) - \sum_{s,c} p_{SC}(s,c) \log_2 \left( \frac{p_{SC}(s,c)}{p_C(c) p_S(s)} \right)
\]

\[
= H(S_n) - D(p_{SC} \mid\mid p_S p_C)
\]

\[
\rightarrow H(S_n \mid C) \leq H(S_n)
\]

(Equality if and only if $S_n$ and $C = f(S_{n-1}, S_{n-2}, \cdots)$ are independent)

- Conditioning does never increase entropy
- Also: Conditional coding does never increase average codeword length
Example: Stationary Markov Process

### Previous Example: Stationary Markov Source

#### conditional Huffman code

<table>
<thead>
<tr>
<th></th>
<th>$S_{n-1} = a$</th>
<th>$S_{n-1} = b$</th>
<th>$S_{n-1} = c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>$p(x \mid a)$</td>
<td>$p(x \mid b)$</td>
<td>$p(x \mid c)$</td>
</tr>
<tr>
<td>$a$</td>
<td>0.90</td>
<td>0.15</td>
<td>0.25</td>
</tr>
<tr>
<td>$b$</td>
<td>0.05</td>
<td>0.80</td>
<td>0.15</td>
</tr>
<tr>
<td>$c$</td>
<td>0.05</td>
<td>0.05</td>
<td>0.60</td>
</tr>
</tbody>
</table>

- $\bar{\ell}(S_{n-1}=a) = 1.1$
- $\bar{\ell}(S_{n-1}=b) = 1.2$
- $\bar{\ell}(S_{n-1}=c) = 1.4$

- $H(S_n \mid a) \approx 0.5690$
- $H(S_n \mid b) \approx 0.8842$
- $H(S_n \mid c) \approx 1.3527$

- Average codeword length: $\bar{\ell}_{\text{cond}} \approx 1.1578$
- Conditional entropy: $H(S_n \mid S_{n-1}) \approx 0.7331$

#### scalar Huffman code

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>29/45</td>
<td>0</td>
</tr>
<tr>
<td>$b$</td>
<td>11/45</td>
<td>10</td>
</tr>
<tr>
<td>$c$</td>
<td>5/45</td>
<td>11</td>
</tr>
</tbody>
</table>

- $\bar{\ell}_{\text{scal}} \approx 1.3556$
- $H(S_n) \approx 1.2575$

Conditioning reduces entropy: from 1.2575 to 0.7331

Conditioning reduces average codeword length: from 1.3556 to 1.1578
### Practical Example: Conditional Coding in H.264 | AVC (CAVLC)

<table>
<thead>
<tr>
<th>TrailingOnes (coeff_token)</th>
<th>TotalCoeff (coeff_token)</th>
<th>0 ≤ nC &lt; 2</th>
<th>2 ≤ nC &lt; 4</th>
<th>4 ≤ nC &lt; 8</th>
<th>8 ≤ nC</th>
<th>nC = -1</th>
<th>nC = -2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>11</td>
<td>1111</td>
<td>0000 11</td>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0001 01</td>
<td>0010 11</td>
<td>0011 11</td>
<td>0000 00</td>
<td>0001 11</td>
<td>0001 11</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>01</td>
<td>10</td>
<td>1110</td>
<td>0000 01</td>
<td>1</td>
<td>01</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>0000 0111</td>
<td>0001 11</td>
<td>0010 11</td>
<td>0001 00</td>
<td>0001 00</td>
<td>0001 10</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0001 00</td>
<td>0011 1</td>
<td>0111 1</td>
<td>0001 01</td>
<td>0001 10</td>
<td>0001 101</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>001</td>
<td>011</td>
<td>1101</td>
<td>0001 10</td>
<td>001</td>
<td>001</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>0000 0011 1</td>
<td>0000 111</td>
<td>0010 00</td>
<td>0010 00</td>
<td>0000 11</td>
<td>0000 0011 1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0000 0110</td>
<td>0010 10</td>
<td>0110 0</td>
<td>0010 01</td>
<td>0000 011</td>
<td>0001 100</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0000 101</td>
<td>0010 01</td>
<td>0111 0</td>
<td>0010 10</td>
<td>0000 010</td>
<td>0001 011</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0001 1</td>
<td>0101</td>
<td>1100</td>
<td>0010 11</td>
<td>0001 01</td>
<td>0000 1</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>0000 0001 1</td>
<td>0000 0111</td>
<td>0001 111</td>
<td>0011 00</td>
<td>0000 10</td>
<td>0000 0011 0</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>0000 0011 0</td>
<td>0001 10</td>
<td>0101 0</td>
<td>0011 01</td>
<td>0000 011</td>
<td>0000 0010 1</td>
</tr>
</tbody>
</table>

(continued)
Example: Binary Markov Process

Binary random process $S = \{S_n\}$:

- $S_n = 0 \rightarrow$ white sample
- $S_n = 1 \rightarrow$ black sample

Statistics measured over a large set of documents

$$p(0) = 0.8$$
$$p(0|0) = 0.9$$

Model: Stationary Markov process

Determine remaining probabilities

$$p(1) = 1 - p(0) = 0.2$$
$$p(1|0) = 1 - p(0|0) = 0.1$$
$$p(0|1) = p(1|0) \cdot p(0) = 0.4$$
$$p(1|1) = 1 - p(0|1) = 0.6$$
Example: Scalar and Conditional Coding

### Conditional Huffman Code

<table>
<thead>
<tr>
<th>$S_{n-1} = 0$</th>
<th>$S_{n-1} = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>$p(x \mid 0)$</td>
</tr>
<tr>
<td>0</td>
<td>0.9</td>
</tr>
<tr>
<td>1</td>
<td>0.1</td>
</tr>
</tbody>
</table>

- $\bar{\ell}(S_{n-1}=0) = 1$
- $H(S_n \mid 0) \approx 0.4690$
- $\bar{\ell}(S_{n-1}=1) = 1$
- $H(S_n \mid 1) \approx 0.9710$

- Average codeword length: $\bar{\ell}_{\text{cond}} \approx 1$
- Conditional entropy: $H(S_n \mid S_{n-1}) \approx 0.5694$

### Scalar Huffman Code

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.8</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.2</td>
<td>1</td>
</tr>
</tbody>
</table>

- $\bar{\ell}_{\text{scal}} \approx 1$
- $H(S_n) \approx 0.7219$

- Conditioning does not improve coding efficiency, even though $H(S_n \mid S_{n-1}) < H(S_n)$
- No codeword can be shorter than 1 bit $\Rightarrow \bar{\ell} \geq 1$
- Problem for sources with probability masses $\gg 0.5$

**How can we improve coding efficiency?** $\Rightarrow$ Joint coding of multiple symbols?
Block Codes

- Code $N > 1$ successive symbols jointly (using a single codeword)
- Design code for $N$-dimensional joint pmf $p(x_1, x_2, \ldots, x_N) = P(S_1 = x_1, S_2 = x_2, \ldots, S_N = x_N)$
- Optimal block code: Huffman algorithm

### Block Huffman Coding for Black and White Document Scans

#### $N = 2$ symbols

<table>
<thead>
<tr>
<th>$s_1 s_2$</th>
<th>$p(s_1, s_2)$</th>
<th>codewords</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0.72</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>0.08</td>
<td>010</td>
</tr>
<tr>
<td>10</td>
<td>0.08</td>
<td>011</td>
</tr>
<tr>
<td>11</td>
<td>0.12</td>
<td>00</td>
</tr>
</tbody>
</table>

$\bar{\ell}_2 = 1.44$

$\bar{\ell} = \bar{\ell}_2 / 2 = 0.72$

#### $N = 3$ symbols

<table>
<thead>
<tr>
<th>$s_1 s_2 s_3$</th>
<th>$p(s_1, s_2, s_3)$</th>
<th>codewords</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0.648</td>
<td>1</td>
</tr>
<tr>
<td>001</td>
<td>0.072</td>
<td>000</td>
</tr>
<tr>
<td>010</td>
<td>0.032</td>
<td>01000</td>
</tr>
<tr>
<td>011</td>
<td>0.048</td>
<td>0101</td>
</tr>
<tr>
<td>100</td>
<td>0.072</td>
<td>001</td>
</tr>
<tr>
<td>101</td>
<td>0.008</td>
<td>01001</td>
</tr>
<tr>
<td>110</td>
<td>0.048</td>
<td>0110</td>
</tr>
<tr>
<td>111</td>
<td>0.072</td>
<td>0111</td>
</tr>
</tbody>
</table>

$\bar{\ell}_3 = 1.952 \Rightarrow \bar{\ell} \approx 0.65$
Block Entropy

- Similar to marginal entropy, but for blocks of $N$ symbols

$\Rightarrow$ Block entropy for $N$ successive symbols

$$
H_N(S) = H(S_1, S_2, \cdots, S_N) = \mathbb{E}\left\{ -\log_2 p(S_1, S_2, \cdots, S_N) \right\}
$$

$$
= - \sum_{x_1, x_2, \cdots, x_N} p(x_1, x_2, \cdots, x_N) \cdot \log_2 p(x_1, x_2, \cdots, x_N)
$$

Bounds for Block Huffman Codes

- Average codeword length $\bar{\ell}_N$ for $N$ symbols: $H_N(S) \leq \bar{\ell}_N < H_N(S) + 1$

$\Rightarrow$ Average codeword length $\bar{\ell}$ per symbol:

$$
\frac{H_N(S)}{N} \leq \bar{\ell} < \frac{H_N(S)}{N} + \frac{1}{N}
$$
Chain Rule for Entropies

- Conditional probabilities

\[ P(X | B) = \frac{P(X, B)}{P(B)} \quad \Rightarrow \quad P(X, B) = P(B) \cdot P(X | B) \]

- Chain rule for joint probability masses

\[
\begin{align*}
p(x_1, x_2, \cdots, x_N) &= p(x_1, x_2, \cdots, x_{N-1}) \cdot p(x_N | x_1, x_2, \cdots, x_{N-1}) \\
&= p(x_1, x_2, \cdots, x_{N-2}) \cdot p(x_{N-1} | x_1, x_2, \cdots, x_{N-2}) \cdot p(x_N | x_1, x_2, \cdots, x_{N-1}) \\
&= p(x_1) \cdot p(x_2 | x_1) \cdot p(x_3 | x_1, x_2) \cdots p(x_N | x_1, x_2, \cdots, x_{N-1})
\end{align*}
\]

- Chain rule for entropies

\[
\begin{align*}
H(S_1, S_2, \cdots, S_N) &= \mathbb{E}\left\{ -\log_2 p(S_1, S_2, \cdots, S_N) \right\} \\
&= \mathbb{E}\left\{ -\log_2 \left( p(S_1) \cdot p(S_2 | S_1) \cdots p(S_N | S_1, S_2, \cdots, S_{N-1}) \right) \right\} \\
&= \mathbb{E}\left\{ -\log_2 p(S_1) \right\} + \mathbb{E}\left\{ -\log_2 p(S_2 | S_1) \right\} + \cdots + \mathbb{E}\left\{ -\log_2 p(S_N | S_1, S_2, \cdots, S_{N-1}) \right\} \\
&= H(S_1) + H(S_2 | S_1) + H(S_3 | S_1, S_2) + \cdots + H(S_N | S_1, S_2, \cdots, S_{N-1})
\end{align*}
\]
Increasing Block Size Does Not Increases Lower Bound

- Chain rule of entropies
  \[ H_N(S) = H(S_1) + H(S_2 \mid S_1) + H(S_3 \mid S_1, S_2) + \ldots + H(S_N \mid S_1, S_2, \ldots, S_{N-1}) \]
  \[ \geq N \cdot H(S_N \mid S_1, S_2, \ldots, S_{N-1}) \]  
  (conditioning does not increase entropy)

- Consider one step of chain rule
  \[ H_N(S_1, S_2, \ldots, S_N) = H_{N-1}(S_1, S_2, \ldots, S_{N-1}) + H(S_N \mid S_1, S_2, \ldots, S_{N-1}) \]
  \[ N \cdot H_N(S) = N \cdot H_{N-1}(S) + N \cdot H(S_N \mid S_1, S_2, \ldots, S_{N-1}) \]  
  (multiplied by \( N \))

  \[ N \cdot H_N(S) \leq N \cdot H_{N-1}(S) + H_N(S) \]  
  (applied above inequality)

\[ (N - 1) \cdot H_N(S) \leq N \cdot H_{N-1}(S) \]

→ Increasing block size never increases lower bound

\[ \frac{H_N(S)}{N} \leq \frac{H_{N-1}(S)}{N - 1} \]

\[ \left( \text{equality if and only if } S \text{ is iid: } H(S_N \mid S_{N-1}, \ldots) = H(S_N) \right) \]
Block Huffman Coding with Increasing Block Sizes

Example: Stationary Markov Source

| \( x \) | \( p(x|a) \) | \( p(x|b) \) | \( p(x|c) \) |
|-------|-------------|-------------|-------------|
| \( a  \) | 0.90        | 0.15        | 0.25        |
| \( b  \) | 0.05        | 0.80        | 0.15        |
| \( c  \) | 0.05        | 0.05        | 0.60        |

Scalar Huffman coding:
\( \bar{\ell}_{\text{scal}} = 1.3556 \)

Conditional Huffman coding:
\( \bar{\ell}_{\text{cond}} = 1.1578 \)

Block Huffman coding:

<table>
<thead>
<tr>
<th>( N )</th>
<th>( \frac{H_N(S)}{N} )</th>
<th>( \frac{\bar{\ell}}{N} )</th>
<th>number of codewords</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.2575</td>
<td>1.3556</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>0.9953</td>
<td>1.0094</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>0.9079</td>
<td>0.9150</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>0.8642</td>
<td>0.8690</td>
<td>81</td>
</tr>
<tr>
<td>5</td>
<td>0.8380</td>
<td>0.8462</td>
<td>243</td>
</tr>
<tr>
<td>6</td>
<td>0.8205</td>
<td>0.8299</td>
<td>729</td>
</tr>
<tr>
<td>7</td>
<td>0.8080</td>
<td>0.8153</td>
<td>2187</td>
</tr>
<tr>
<td>8</td>
<td>0.7987</td>
<td>0.8027</td>
<td>6561</td>
</tr>
<tr>
<td>9</td>
<td>0.7914</td>
<td>0.7940</td>
<td>19683</td>
</tr>
</tbody>
</table>
Example: Two Successive Quantization Indexes in MP3

Block Huffman Coding in MP3

- Joint coding of two successive quantization indexes
- Multiple Huffman tables specified in standard (designed offline)
Example: Coded Block Pattern in MPEG-2

Y: 0/1 0/1
Cb: 0/1
Cr: 0/1

\[
\text{coded\_block\_pattern} = \text{x}x\text{x}x\text{x} \quad \text{(bit mask)}
\]
(values: 0..63)

<table>
<thead>
<tr>
<th>coded_block_pattern VLC code</th>
<th>cbp</th>
<th>coded_block_pattern VLC code</th>
<th>cbp</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>60</td>
<td>0001 1100</td>
<td>35</td>
</tr>
<tr>
<td>1101</td>
<td>4</td>
<td>0001 1011</td>
<td>13</td>
</tr>
<tr>
<td>1100</td>
<td>8</td>
<td>0001 1010</td>
<td>49</td>
</tr>
<tr>
<td>1011</td>
<td>16</td>
<td>0001 1001</td>
<td>21</td>
</tr>
<tr>
<td>1010</td>
<td>32</td>
<td>0001 1000</td>
<td>41</td>
</tr>
<tr>
<td>1001 1</td>
<td>12</td>
<td>0001 0111</td>
<td>14</td>
</tr>
<tr>
<td>1001 0</td>
<td>48</td>
<td>0001 0110</td>
<td>50</td>
</tr>
<tr>
<td>1000 1</td>
<td>20</td>
<td>0001 0101</td>
<td>22</td>
</tr>
<tr>
<td>1000 0</td>
<td>40</td>
<td>0001 0100</td>
<td>42</td>
</tr>
<tr>
<td>0111 1</td>
<td>28</td>
<td>0001 0011</td>
<td>15</td>
</tr>
<tr>
<td>0111 0</td>
<td>44</td>
<td>0001 0010</td>
<td>51</td>
</tr>
</tbody>
</table>

(continued)
Lossless Source Coding Theorem

Entropy Rate
- Observation: Lower bound for block coding typically decreases with increasing $N$

**Entropy rate**: Limit $(N \to \infty)$ for lower bound of block coding

$$\bar{H}(S) = \lim_{N \to \infty} \frac{H(S_0, \cdots, S_{N-1})}{N} = \lim_{N \to \infty} \frac{H_N(S)}{N}$$

- The limit always exists for stationary random sources

Fundamental Lossless Source Coding Theorem
- Average codeword length for all lossless codes is bounded by

$$\bar{\ell} \geq \bar{H}(S) = \lim_{N \to \infty} \frac{H_N(S)}{N}$$

- Asymptotically achievable with block Huffman coding for $N \to \infty$
Entropy Rate for IID Sources

\[
\tilde{H}(S) = \lim_{N \to \infty} \frac{1}{N} H(S_1, S_2, \cdots, S_N)
\]

\[
= \lim_{N \to \infty} \frac{1}{N} \left( H(S_1) + H(S_2 \mid S_1) + H(S_3 \mid S_1, S_2) + \cdots + H(S_N \mid S_1, S_2, \cdots, S_{N-1}) \right)
\]

\[
= \lim_{N \to \infty} \frac{1}{N} \sum_{k=1}^{N} H(S_k)
\]

\[
= \lim_{N \to \infty} \frac{1}{N} \left( N \cdot H(S) \right)
\]

\[
= H(S)
\]

Note: Block Huffman coding may still improve coding efficiency
Entropy Rate for Stationary Markov Sources

\[ \bar{H}(S) = \lim_{N \to \infty} \frac{1}{N} H(S_1, S_2, \cdots, S_N) \]

\[ = \lim_{N \to \infty} \frac{1}{N} \left( H(S_1) + H(S_2 | S_1) + H(S_3 | S_1, S_2) + \cdots + H(S_N | S_1, S_2, \cdots, S_{N-1}) \right) \]

\[ = \lim_{N \to \infty} \frac{1}{N} \left( H(S_1) + \sum_{k=2}^{N} H(S_k | S_{k-1}) \right) \]

\[ = \lim_{N \to \infty} \frac{1}{N} \left( H(S) + (N - 1) \cdot H(S_n | S_{n-1}) \right) \]

\[ = \lim_{N \to \infty} \frac{H(S)}{N} + \lim_{N \to \infty} \frac{N - 1}{N} \cdot H(S_n | S_{n-1}) \]

\[ = H(S_n | S_{n-1}) \]
Summary of Lecture: Huffman Codes

Huffman Algorithm
- Generates prefix codes with minimum redundancy for any given pmf
- Yields optimal uniquely decodable codes

Scalar Huffman codes
- Codeword table assigns codeword to each individual symbol
- Table size is equal to alphabet size

Conditional Huffman codes
- Separate codeword table for each possible condition (e.g., preceding symbol)
- Switch codeword tables during encoding and decoding

Block Huffman codes
- Code $N > 1$ successive symbols jointly
- Codeword table assigns codeword to combination of $N$ successive symbols
Summary of Lecture: Entropy Measures and Bounds

Entropy Measures

- Scalar (or marginal) entropy: \( H(S) = - \sum_a p(a) \cdot \log_2 p(a) \)
- Conditional entropy: \( H(S \mid C) = - \sum_{a,c} p(a, c) \cdot \log_2 p(a \mid c) \)
- Block entropy: \( H_N(S) = - \sum_{x_1, \ldots, x_N} p(x_1, \ldots, x_N) \cdot \log_2 p(x_1, \ldots, x_N) \)
- Entropy rate: \( \bar{H}(S) = \lim_{N \to \infty} \frac{1}{N} H_N(S) \)

Bounds for Lossless Coding

- Scalar Huffman coding: \( H(S) \leq \bar{\ell} < H(S) + 1 \)
- Conditional Huffman coding: \( H(S \mid C) \leq \bar{\ell} < H(S \mid C) + 1 \)
- Block Huffman coding: \( \frac{1}{N} H_N(S) \leq \bar{\ell} < \frac{1}{N} H_N(S) + \frac{1}{N} \)

- All lossless codes: \( \bar{\ell} \geq \bar{H}(S) \) (fundamental lossless source coding theorem)
Exercise 1: Huffman Code

Given is a discrete iid process $X$ with the alphabet $\mathcal{A} = \{a, b, c, d, e, f, g\}$. The pmf $p_X(x)$ is specified in the following table.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p_X(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>1/3</td>
</tr>
<tr>
<td>$b$</td>
<td>1/9</td>
</tr>
<tr>
<td>$c$</td>
<td>1/27</td>
</tr>
<tr>
<td>$d$</td>
<td>1/27</td>
</tr>
<tr>
<td>$e$</td>
<td>1/27</td>
</tr>
<tr>
<td>$f$</td>
<td>1/9</td>
</tr>
<tr>
<td>$g$</td>
<td>1/3</td>
</tr>
</tbody>
</table>

(a) Develop a Huffman code for the given pmf $p_X(x)$.

(b) Calculate the average codeword length of the developed Huffman code.

(c) Calculate the absolute and relative redundancy for the developed Huffman code.
Let $Z = \{Z_n\}$ be a binary iid process with alphabet $\{0, 1\}$ and pmf $\{0.5, 0.5\}$ (e.g., coin toss).

Let $X = \{X_n\}$ be a random process given by $X_n = Z_{n-1} + Z_n$.

(a) Determine the marginal pmf $p_X(x)$ and the marginal entropy $H(X)$.

(b) Develop a scalar Huffman code and calculate its average codeword length.

(c) Determine the conditional pmf $p_{X_n|X_{n-1}}(x_n | x_{n-1})$ and the conditional entropy $H(X_n | X_{n-1})$.

(d) Develop a conditional Huffman code and calculate its average codeword length.

(e) Develop a block Huffman code for $N = 2$ symbols and calculate its average codeword length.

(f) Optional (more difficult):
   - Derive a formula for the $N$-th order block entropy $H_N(X_n, \cdots, X_{n+N-1})$.
   - Determine the entropy rate $\bar{H}(X)$.
   - Is $X$ a Markov process?
Exercise 3: Estimate Entropy Measures (Implementation Task)

Write a program (in a programming language of your choice) that estimates the following entropy measures based on the statistics of a given input file:

- Marginal entropy: \( H(S_n) \)
- 1-st order conditional entropy: \( H(S_n \mid S_{n-1}) \)
- Block entropy of size \( N = 2 \): \( H(S_n, S_{n+1}) \) [calculate also \( H(S_n, S_{n+1})/2 \)]

Assume that all files represent a sequence of 8-bit samples (i.e., each byte represents a symbol).

Test your program for the following sample files (from course website):

- white uniform noise: “whiteUniformNoise.raw”
- white Gaussian noise: “whiteGaussianNoise.raw”
- correlated Gaussian noise: “correlatedGaussianNoise.raw”
- English text file: “englishText.txt”
- 8-bit audio data: “audioData.raw”
- 8-bit image data: “imageData.raw”

What can you conclude about the potential to compress these files?
Exercise 4: Geometric Pmf (Optional)

Given is a Bernoulli process (binary iid process) $B = \{B_n\} = \{B\}$ with the alphabet $A_B = \{0, 1\}$ and the pmf $p_B(0) = p$, $p_B(1) = 1 - p$ with $0 < p < 1$.

Consider the random variable $X$ that specifies the number of successive random variables $B_n$ that have to be observed to get exactly one “1”.

(a) Determine the pmf for $X$ as function of $p$.
(b) Determine the entropy $H(B)$ as function of $p$.
(c) Determine the entropy $H(X)$ as function of $H(B)$ and $p$.
(d) What structure has an optimal scalar variable-length code for $X$ and $p \leq 0.5$? Calculate its average codeword length as function of $p$.

Calculate its relative redundancy as function of $p$.

Hints:

$\forall |a| < 1, \sum_{k=0}^{\infty} a^k = \frac{1}{1-a}$ and $\forall |a| < 1, \sum_{k=0}^{\infty} k a^k = \frac{a}{(1-a)^2}$