Outstanding Reading

- Read chapters 1, and 2 in C&T.
- Read chapter 3 in C&T.
- Read section 11.1,11.3, method of types and universal source coding.
- Read chapter 4.
- Read chapter 5.
Announcements, Assignments, and Reminders

- New homework out soon.
- Late policy: 10% every 24 hour period that you are late, and no more than 3 days late accepted.
- Lowest grade out of all HW grades is not counted towards final grade (so you can skip one HW with impunity).
- Please do use our discussion board (https://catalyst.uw.edu/gopost/board/karna/25503/) for all questions, so that all will benefit from them being answered.

Class Road Map

- L1 (1/3): Overview, Entropy
- L2 (1/5): Props. Entropy, Mutual Information, KL-Divergence
- L3 (1/10): KL-Divergence, Jensen, properties, Data Proc. Inequality
- L5 (1/17): Fano, AEP
- L6 (1/19): snow
- L6 (1/24): AEP, source coding
- L7 (1/26): Method of Types
- L9 (2/2): HMMs, coding
- L10 (2/7): Coding, Kraft, Huffman
- L11 (2/9): Huffman, Arithmetic, midterm
- L12 (2/14): Midterm
- L13 (2/16):
- L14 (2/21):
- L15 (2/23):
- L16 (2/28):
- L17 (3/1):
- L18 (3/6):
- L19 (3/8):

Finals Week: March 12th–16th.
Practical Coding

- We want to develop practical coding algorithms that still approach, or achieve, the entropy limit.
- They might use the distribution \( p(x) \) which is either given or is estimated in some way.
- We won’t get into any details on how to estimate \( p(x) \) (that is a density estimation problem) but we assume we either have it or some approximation.
- We will look, however, at what happens if the true distribution is \( p(x) \) and we use \( q(x) \) instead.

Source Code

Definition 2.1 (source code)
A source code \( C \) for r.v. \( X \) is a mapping

\[
C : \mathcal{X} \rightarrow \mathcal{D}^*
\]  

(1)

from \( \mathcal{X} \) to \( \mathcal{D}^* \), the set of finite strings from a \( D \)-ary alphabet. \( C(x) \) is the codeword corresponding to \( x \), and \( \ell(x) \) is the length of the codeword.

Definition 2.2 (expected length)
The expected length \( L(C) \) of code \( C \) for r.v. \( X \) with distribution \( p(x) \) is

\[
L(C) = \sum_{x} p(x) \ell(x)
\]  

(2)
Aside: English

- isenglishselfpunctuating


how long does it take you to read this sentence that is written without any punctuation marks or even end of sentence marks such as a question mark or even inter-sentence spaces?

- nowhere

“no where” (or “now here”).

Codes

- Assume $D = \{0, 1, 2, \ldots, D - 1\}$ in general (but often $D = 2$).
- Another code,

**Example 2.3**

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>$c(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1/4</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>1/8</td>
<td>110</td>
</tr>
<tr>
<td>4</td>
<td>1/8</td>
<td>111</td>
</tr>
</tbody>
</table>

Let $\mathcal{X} = \{1, 2, 3, 4\}$ and $\mathcal{D} = \{0, 1\}$. We can define the code with a table.

- In this case, $H(X) = 1.75$. But $L(C) = E\ell(X) = 1.75$, so this code is pretty good.
- Moreover, it is easy to code. What source symbols correspond to the string 010110110110100? 1, 2, 3, 4, 4, 3, 2, 1
- With punctuation: 0,10,110,111,111,110,10,0, so code in some sense is “self punctuating”

Another code

- Here, $\mathcal{X} = \{1, 2, 3\}$ and $\mathcal{D} = \{0, 1\}$
- Code is:

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>$c(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1$</td>
<td>1/3</td>
<td>1</td>
</tr>
<tr>
<td>$2$</td>
<td>1/3</td>
<td>1</td>
</tr>
<tr>
<td>$3$</td>
<td>1/3</td>
<td>11</td>
</tr>
</tbody>
</table>

- So $H = 1.58$ but $E\ell(X) = 1.66 > H$ bits.
- Can we easily decode? 10110010 = 2,3,1,1,2
### Code types

**Definition 2.4 (non-singular)**

A code is said to be non-singular if every element of the range of $X$ (i.e., all elements of $X$) maps to a different string in $D^*$. I.e.,

$$x_i \neq x_j \Rightarrow C(x_i) \neq C(x_j) \quad (3)$$

- We can view this as a mapping. It is less strict than onto but sufficient for being able to decode individual symbols.

![Diagram showing non-singular mapping]

### Our goal

**Definition 2.5 (code extension)**

A code extension $C^*$ of $C$ is a mapping from finite length strings of $D$, defined as:

$$C(x_1, x_2, \ldots, x_n) = C(x_1)C(x_2) \ldots C(x_n) \quad (4)$$

- Note that there are no commas in the extension, rather concatenation.
- Ex: If $C(x_1) = 0$ and $C(x_2) = 1$ then $C(x_1, x_2) = 01$. 

![Diagram showing code extension]
### Code types

**Definition 2.6 (uniquely decodable)**

A code $C$ with extension $C^*$ is **uniquely decodable** if the extension $C^*$ is non-singular.

- But how long must we wait until we know the source? In some even uniquely decodable cases, we might need to wait until the end.
- Ex: consider the code

<table>
<thead>
<tr>
<th>$x$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C(x)$</td>
<td>10</td>
<td>00</td>
<td>11</td>
<td>110</td>
</tr>
</tbody>
</table>

- Is this code uniquely decodable? Yes.

- Code string: 1100000000 = 3,2,2,2,2
- Code string: 11000000000 = 4,2,2,2,2
- So we don’t know identity of first symbol until end of code string.

---

### Prefix codes

**Definition 2.7 (prefix code)**

A code is called a **prefix code** or an **instantaneous code** if no codeword is a prefix of any other codeword.

- We know the end of a codeword because it can’t be a prefix of any other codeword.
- Code in previous page is not prefix free, 11 was a prefix of 110 so we couldn’t decide between 11 or 110 until we could count the number of zeros.
- A prefix code is self-punctuating (since there are implicit punctuation marks between codewords).
- Prefix code $\Rightarrow$ uniquely decodable. But (as we saw) uniquely decodable $\not\Rightarrow$ prefix code.
Goal is to find a code with the shortest possible expected length.

From the above code class, we might think that we want to use codes from the largest class possible (since we might think we’re more likely to get shorter codes).

We can do better than entropy with non-singular codes, but we want lossless encoding $x = \text{ungzip}(\text{gzip}(x))$.

For $X = \{1, 2, 3, 4\}$ and binary code, consider the following 4 codes.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>$C_1$</th>
<th>$C_{II}$</th>
<th>$C_{III}$</th>
<th>$C_{IV}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>3</td>
<td>0.125</td>
<td>1</td>
<td>00</td>
<td>110</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>0.125</td>
<td>10</td>
<td>11</td>
<td>111</td>
<td>0111</td>
</tr>
<tr>
<td>$H(X)$</td>
<td>1.75</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$E\ell(X)$</td>
<td>-</td>
<td>1.125</td>
<td>1.25</td>
<td>1.75</td>
<td>1.875</td>
</tr>
</tbody>
</table>

- $C_1$ is singular.
- $C_{II}$ is non-singular, but not uniquely decodable.
- $C_{III}$ is non-singular, uniquely decodable, and prefix.
- $C_{IV}$ is non-singular, uniquely decodable, and not a prefix code.
Kraft inequality

**Theorem 3.1 (Kraft inequality)**

*For any instantaneous code (prefix code) over alphabet of size $D$, the codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$ must satisfy*

\[
\sum_i D^{-\ell_i} \leq 1
\]  

*(5)*

*Conversely, given a set of codeword lengths satisfying the above inequality, $\exists$ an instantaneous code with these word lengths.*

- Note what the converse is saying: there exists a code with these lengths, not that all codes with these lengths will satisfy the inequality.
- Key point: for $\ell_i$ satisfying Kraft, no further restriction imposed by also wanting a prefix code, so we might as well use a prefix code (assuming it is easy to find given the lengths)

**proof of Kraft inequality.**

- Represent set of codes on a $D$-ary (not necessarily balanced) tree:

- Codewords correspond to leaves
- Path from root to leaf determines a codeword
- Prefix condition: won’t get to a codeword until we get to a leaf (no descendants of codewords are codewords)
Kraft inequality

...proof of Kraft inequality cont.

- $\ell_{\max} = \max_i(\ell_i)$ is the length of the longest codeword.
- We can expand the full-tree down to depth $\ell_{\max}$.
  Some nodes at that level $\ell_{\max}$ are either:
  - codewords,
  - descendants of codewords, or
  - neither
- Consider a codeword $i$ at level $\ell_i$ in tree (so it has length $\ell_i$).
- Then, there are $D^{\ell_{\max} - \ell_i}$ descendants in the tree at level $\ell_{\max}$.
- Because of prefix condition, descendants of code $i$ at level $\ell_i$ are disjoint from descendants of code $j$ at level $\ell_j$ when $i \neq j$ (i.e., descendant sets for different codewords are disjoint).
- Also, total number of nodes in set of all descendants is $\leq D^{\ell_{\max}}$.

All of the above implies:

$$\sum_i D^{\ell_{\max} - \ell_i} \leq D^{\ell_{\max}} \quad \Rightarrow \quad \sum_i D^{-\ell_i} \leq 1 \quad (6)$$

Conversely: given codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$ satisfying Kraft inequality (we must construct a prefix code with these lengths).
- Consider a full $D$-ary tree of depth $\ell_{\max}$ with $D^{\ell_{\max}}$ terminal nodes.
- @ level 0, $\exists$ fraction 1 of the descendants at each node at that level;
- @ level 1, $\exists$ fraction $1/D$ descendants at each node at that level;
- @ level 2, $\exists$ fraction $1/D^2$ . . .
- In general, at each level $i \in [0, \ell_{\max}]$ in tree, there is a fraction $D^{-i}$ terminal nodes that are descendants that stem from each of the $D^i$ nodes at level $i$.

...
Kraft inequality

... proof of Kraft inequality cont.

- Sort the lengths \((\ell_1, \ell_2, \ldots, \ell_m)\) ascending to \((s_1, s_2, \ldots, s_m)\) with \(s_1 \leq s_2 \leq \cdots \leq s_m\). Note there are as many lengths as there are codewords.
- For length \(s_1\) chose any node at level \(s_1\) to indicate the code.
- To ensure prefix free property, the node becomes a terminal node, thus eliminating a fraction \(D^{-s_1}\) of the terminal nodes at depth \(\ell_{\text{max}}\) (which would have been potential code words of longer length, but now they are out of the running).
- Next: chose any remaining node at level \(s_2\), thus eliminating a fraction \(D^{-s_2}\) of the nodes \((D^{s_1} - 1)D^{s_2-s_1}\) choices at this point).
- Total fraction eliminated is \(D^{-s_1} + D^{-s_2}\).

Continuing this process, we eliminate a fraction \(\sum_{i=1}^{m} D^{-s_i}\) of the nodes, while retaining that the code is instantaneous (a codeword can’t be a prefix of another).
- But since by assumption \(\sum_{i=1}^{m} D^{-s_i} \leq 1\) we never eliminate more than all of the codewords, so this process won’t run out of codewords.
- Thus, we have created a prefix-free code with the desired lengths.
Infinite Kraft

**Theorem 3.2 (countably infinite Kraft)**

For any countably infinite set of codewords that form a prefix set, this satisfies the extended Kraft inequality, i.e.

\[
\sum_{i=1}^{\infty} D^{-\ell_i} \leq 1
\]

(7)

Conversely, given \( \ell_i \) satisfying the above, \( \exists \) a prefix code with these lengths.

**Proof of countably infinite Kraft.**

- Assume we have such a prefix code, and let the \( D \)-ary alphabet be \( \{0, 1, \ldots, D - 1\} \).
- Consider the \( i^{th} \) codeword \( y_1, y_2, \ldots, y_{\ell_i} \).

Kraft inequality

... proof of infinite Kraft.

- Consider expansion of codeword using fractional digits:

\[
0.y_1y_2y_3\ldots y_{\ell_i} = \sum_{j=1}^{\ell_i} y_j D^{-j}
\]

(8)

- Examples: When \( D = \{0, 1\} \) then 0.1 = 1/2, 0.01 = 1/4, 0.11 = 3/4, and 0.001 = 1/8 (so bits are after the binary point).
- Associate each codeword \( y_1: \ell_i \) with the half-open interval on the real line \( [0.y_1y_2\ldots y_{\ell_i}, 0.y_1y_2\ldots y_{\ell_i} + 1/D^{\ell_i}] \)
- Example: With \( D = 10 \), then if \( 0.y_1y_2y_3 = 0.157 \), the associated half-open interval is \( [0.157, 0.158) \), and if \( 0.y_1y_2y_3 = 0.159 \), the associated half-open interval is \( [0.159, 0.160) \).
Kraft inequality

...proof of infinite Kraft.

- So the interval for codeword $y_1y_2y_3 \ldots y_{\ell_i}$ corresponds to the set of all real numbers that begins with $0.y_1y_2y_3 \ldots y_{\ell_i}$ and is thus a sub-interval of the unit interval.
- Also $y_1y_2y_3 \ldots y_{\ell_i}$ is not a prefix of any other codeword, so the intervals must be disjoint.
- Length of interval for codeword $y_1y_2y_3 \ldots y_{\ell_i}$ is $D^{-\ell_i}$.
- And since all intervals live in $[0, 1)$ we must have

$$\sum_i D^{-\ell_i} \leq 1 \quad (9)$$

- Proof of converse is similar to finite case and also to arithmetic coding that we'll soon see, so we skip the proof here.

Towards Optimal Codes

- Summarizing: Prefix code $\Leftrightarrow$ Kraft inequality.
- Thus, we need only find lengths that satisfy Kraft to find a prefix code.
- Goal: find a prefix code with minimum expected length

$$L(C) = \sum_i p_i \ell_i \quad (10)$$

- This is an constrained optimization problem:

$$\min_{\{\ell_1:m\} \in \mathbb{Z}_+^+} \sum_i p_i \ell_i \quad (11)$$

subject to $\sum_i D^{-\ell_i} \leq 1$

- Linear integer program is an NP-complete optimization, not likely to be efficiently solvable (unless P=NP).
Towards Optimal Codes

- Relax the integer constraints on \( \ell_i \) for now, and consider Lagrangian

\[
J = \sum_i p_i \ell_i + \lambda \left( \sum_i D^{-\ell_i} - 1 \right)
\]  
(12)

- Taking derivatives and setting to 0,

\[
\frac{\partial J}{\partial \ell_i} = p_i - \lambda D^{-\ell_i} \ln D = 0
\]  
\Rightarrow D^{-\ell_i} = \frac{p_i}{\lambda \ln D}
\]  
(13)

\[
\frac{\partial J}{\partial \lambda} = \sum_i D^{-\ell_i} - 1 = 0
\]  
\Rightarrow \lambda = \frac{1}{\ln D}
\]  
(14)

\[
\Rightarrow D^{-\ell_i} = p_i \quad \text{yielding} \quad \ell^*_i = -\log_D p_i
\]  
(15)

This implies that:

\[
L^* = \sum_i p_i \ell^*_i = -\sum_i p_i \log_D p_i = H_D(X) = H(X)/\log D
\]  
(17)

- So the optimal expected code length, as a result of this optimization process, is the entropy assuming that we are allowed to have fractional code lengths

- Since \( \ell^*_i = -\log_D p_i \), this means that optimal code “length” (while fractional) is the same as the information about the event. I.e., shortest possible coding length is the inherent information about an event. This is like the MDL (minimum description principle), tries to find the simplest explanation about a source.

- Compare fractional codeword lengths to long block codes, what is the relation?
Entropy is the minimum expected length. That is, the expected length $L$ of any instantaneous $D$-ary code (which thus satisfies Kraft inequality) for a r.v. $X$ is such that

$$L \geq H_D(X)$$

(18)

with equality iff $D^{-\ell_i} = p_i$.

Proof of Theorem 3.3.

$$L - H_D(X) = \sum_i p_i \ell_i - \sum_i p_i \log_D 1/p_i$$

(19)

$$= - \sum_i p_i \log_D D^{-\ell_i} + \sum_i p_i \log_D p_i$$

(20)

$$= - \sum_i p_i \log_D D^{-\ell_i} + \log_D(\sum_i D^{-\ell_i}) - \log_D(\sum_i D^{-\ell_i})$$

(21)

$$+ \sum_i p_i \log_D p_i$$

(now define $r_i = \frac{D^{-\ell_i}}{\sum_i D^{-\ell_i}}$)

(22)

$$= \sum_i p_i \log \frac{p_i}{r_i} - \log_D(\sum_i D^{-\ell_i}) = D(p||r) + \log_D(1/c)$$

(23)

$$\geq 0$$ since $c \leq 1$ by Kraft, where $c = \sum_i D^{-\ell_i}$

(24)
Optimal Code Lengths

...Proof of Theorem 3.3.

- So we have that $L \geq H_D(X)$.
- Equality, $L = H$ is achieved iff $p_i = D^{-\ell_i}$ for all $i \iff -\log_D p_i$ is an integer . . .
- . . . in which case $c = \sum_i D^{-\ell_i} = 1$

Definition 3.4 ($D$-adic)

A probability distribution is called $D$-adic w.r.t. $D$ if each of the probabilities is $= D^{-n}$ for some $n$.

- Ex: when $D = 2$, the distribution $[\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}] = [2^{-1}, 2^{-2}, 2^{-3}, 2^{-3}]$ is 2-adic.
- Thus, we have equality above iff the distribution is appropriately $D$-adic.

Summary of recent results

To summarize the conditions and relations:

- In the relaxed optimization problem, we relaxed the lengths so that they need not be integers, and we get $E\ell = H$.
- Assuming Kraft is true (and thus a prefix code exists), we have (with integer lengths) that $E\ell \geq H$.
- i.e., if we assume Kraft, and $\ell_i = -\log_D p_i$ is an integer, then $E\ell = H$.
- i.e., if we assume Kraft, and $\ell_i \neq -\log_D p_i$, but the lengths $\ell_i$ are still integers, then we have $E\ell > H$ strictly.
Shannon Codes

- \( L - H = \mathcal{D}(p||r) + \log_D 1/c \), with \( c = \sum D^{-\ell_i} \)

- Thus, to code find closest (in the KL sense) \( D \)-adic distribution w.r.t. \( D \) to \( p \) and then construct the code as in the proof of the Kraft inequality converse.

- In general, however, unless \( P=NP \), it is hard to find the KL closest \( D \)-adic distribution (integer linear programming problem).

- **Shannon codes**: consider \( \ell_i = \lceil \log_D 1/p_i \rceil \) as the code lengths

- \( \sum_i D^{-\ell_i} = \sum_i D^{-\lceil \log 1/p_i \rceil} \leq \sum_i D^{-\log 1/p_i} = \sum_i p_i = 1 \)

- This means Kraft inequality holds for these lengths, so there is a prefix code (if the lengths were too short there might be a problem but we’re rounding up).

- Also, we have a bound on lengths in terms of real numbers

  \[
  \log_D \frac{1}{p_i} \leq \ell_i < \log_D \frac{1}{p_i} + 1
  \]  

(25)

Theorem 4.1

- Let \( \ell_1^*, \ell_2^*, \ldots, \ell_m^* \) be the optimal integral codeword lengths for source \( p \) and \( D \)-ary alphabet. \( L^* \) is the expected length. Then

  \[
  \mathcal{H}_D(X) \leq L^* < \mathcal{H}_D(X) + 1
  \]

(27)

- So average overhead of using integers (rather than fractional) codeword lengths is no more than one bit per symbol.
How bad is one bit?

- How bad is this overhead?
- Depends on $H$. Efficiency of code

$$0 \leq \text{Efficiency} \triangleq \frac{H_D(X)}{E\ell(X)} \leq 1 \quad (28)$$

- If $E\ell(X) = H_D(X) + 1$, then efficiency $\to 1$ as $H(X) \to \infty$.
- Efficiency $\to 0$ as $H(X) \to 0$, so entropy would need to be very large for this to be good.
- For small alphabets, impossible to have good efficiency. E.g., $D = \{0, 1\}$ then $\max H(X) = 1$, so best possible efficiency is 50%.

Improving efficiency

- Such symbol codes are inherently disadvantaged, unless their distributions are $D$-adic.
- We can reduce overhead (improve efficiency) by coding $> 1$ symbol at a time (block code, or a vector code, the symbol is the vector).
- Let $L_n$ be the expected length of $n$ symbols $x_{1:n}$. $L_n$ is the expected per-symbol length, when encoding $n$ symbols.

$$L_n = \frac{1}{n} \sum_{x_{1:n}} p(x_{1:n}) \ell(x_{1:n}) = \frac{1}{n} E\ell(x_{1:n}) \quad (29)$$

- Let’s use Shannon coding lengths to get

$$\sum_i p_i \left( \log \frac{1}{p_i} \leq \ell_i \leq \log \frac{1}{p_i} + 1 \right) \quad (30)$$

$$\Rightarrow H(X_1, \ldots, X_n) \leq E\ell(X_{1:n}) < H(X_1, \ldots, X_n) + 1 \quad (31)$$
Improving efficiency

- If the $X_i$ are i.i.d. then $H(X_1, \ldots, X_n) = nH(X_i)$.
- ⇒ we have that
  \[ H(X) \leq L_n \leq H(X) + \frac{1}{n} \quad (32) \]

As $n$ gets big, per symbol penalty of a Shannon code decreases, and we approach the Entropy limit (per symbol), although once again we have to code a block at a time.

- Again, even if symbols are independent it is better to code jointly.

Stochastic processes

- Consider any stationary (ergodic) stochastic process. Then
  \[ H(X_1, \ldots, X_n) \leq E\ell(X_1, \ldots, X_n) < H(X_1, \ldots, X_n) + 1 \quad (33) \]
  \[ \Rightarrow \frac{H(X_1, \ldots, X_n)}{n} \leq L_n < \frac{H(X_1, \ldots, X_n)}{n} + \frac{1}{n} \quad (34) \]

- If stationary, than l.h.s. → $H(X)$ as $n \to \infty$.
- Thus, as $n$ gets large, expected length of code goes to the entropy rate of the stochastic process.
- We can make penalty per source symbol as small as we want if we don’t mind long block lengths. This can be stated as a theorem

**Theorem 4.2**

*Minimum expected codeword lengths per symbol satisfy*

\[ \frac{H(X_1, \ldots, X_n)}{n} \leq L^*_n < \frac{H(X_1, \ldots, X_n)}{n} + \frac{1}{n} \quad (35) \]

if $X_i$ is stationary. I.e., $L^* \to H(X)$.
Coding with the wrong distribution

- In general, we don’t have the “true” distribution (if there is one).
- With the wrong distribution, we’ll make mistakes. I.e., Shannon code would use lengths $\ell(x) = \lceil \log \frac{1}{q(x)} \rceil$ but the true probability is $p(x) \neq q(x)$. How does this hurt us?

$$E\ell(X) = \sum_x p(x)\lceil \log \frac{1}{q(x)} \rceil \leq \sum_x p(x)\left(\log \frac{1}{q(x)} + 1\right) \quad (36)$$

$$= \sum_x p(x)\left(\log \frac{p(x)}{q(x)} \frac{1}{p(x)} + 1\right) \quad (37)$$

$$= \sum_x p(x) \log \frac{p(x)}{q(x)} + \sum_x p(x) \log \frac{1}{p(x)} + 1 \quad (38)$$

$$= D(p||q) + H(p) + 1 \quad (39)$$

- Thus, $D(p||q)$ is per symbol bit penalty for using wrong distribution.

Theorem 4.3

**Expected length under $p(x)$ of code with $\ell(x) = \lceil \log \frac{1}{q(x)} \rceil$ satisfies**

$$H(p) + D(p||q) \leq E_p \ell(X) \leq H(p) + D(p||q) + 1 \quad (40)$$

- l.h.s. is the best we can do with the wrong distribution $q$ when the true distribution is $p$. 
Code classes

- Goal is to find a code with the shortest possible expected length.
- From the above code class, we might think that we want to use codes from the largest class possible (since we might think we’re more likely to get shorter codes).
- We can do better than entropy with non-singular codes, but we want lossless encoding $x = \text{ungzip}(\text{gzip}(x))$.

Kraft revisited

- We proved Kraft inequality is true for instantaneous codes (and vice verse).
- Could it be true for all uniquely decodable codes?
- Could larger class of codes have shorter expected codeword lengths?
- Since larger, we might (naïvely) expect that we could do better.

Theorem 5.1

Codeword lengths of any uniquely decodable code (not. nec. instantaneous) must satisfy Kraft inequality $\sum_i D^{-\ell_i} \leq 1$. Conversely, given a set of codeword lengths that satisfy Kraft, it is possible to construct a uniquely decodable code.

Proof.

Proof converse we already saw before (given lengths, we can construct a prefix code which is thus uniquely decodable). Thus we only need prove the first part.

...
Kraft and uniquely decodable

Proof of Theorem 5.1.

- Given: uniquely decodable (not necessarily instantaneous) code with lengths $\ell(x)$, and length of $k$-extension $\ell(x, \ldots, x_k) = \sum_{i=1}^{k} \ell(x_i)$ we wish to prove that $\sum_x D^{-\ell(x)} \leq 1$.
- Define $S = \sum_{x \in \mathcal{X}} D^{-\ell(x)}$, then

$$S^k = \left[ \sum_{x} D^{-\ell(x)} \right]^k = \sum_{x_{1:k} \in \mathcal{X}^k} D^{-\ell(x_1)} D^{-\ell(x_2)} \ldots D^{-\ell(x_k)}$$

(41)

$$= \sum_{x_{1:k} \in \mathcal{X}^k} D^{-[\sum_{i=1}^{k} \ell(x_i)]} = \sum_{x_{1:k} \in \mathcal{X}^k} D^{-\ell(x_{1:k})}$$

(42)

$$= \sum_{m=1}^{k\ell_{\text{max}}} a(m) D^{-m}$$

(43)

where $\ell_{\text{max}} = \max_x \ell(x)$ is the maximum codeword length.
- $a(m)$ = number of source sequences $x_{1:k}$ mapped into code words of length $m$, i.e.,

$$a(m) = \left| \left\{ x_{1:k} \in \mathcal{X}^k : \ell(x_{1:k}) = m \right\} \right|$$

(44)

There are $D^m$ codewords of length $m$, and each of them can have (at most) one associated source sequence (since code is uniquely decodable). Hence, $a(m) \leq D^m$.

Prof. Jeff Bilmes
page 43
Kraft and uniquely decodable

...proof of Theorem 5.1.

- So continuing,

\[ S^k = \sum_{m=1}^{k \ell_{\text{max}}} a(m) D^{-m} \leq \sum_{m=1}^{k \ell_{\text{max}}} D^m D^{-m} = k \ell_{\text{max}} \quad \forall k \] (45)

- So, \( S^k \) (exponential in \( k \)) never greater than \( k \ell_{\text{max}} \) (polynomial in \( k \)) \( \Rightarrow S \leq 1 \).
- Giving \( S = \sum_{x \in \mathcal{X}} D^{-\ell(x)} \leq 1 \).

Summary: uniquely decodable vs. instantaneous codes

- Set of achievable codeword lengths the same for uniquely decodable codes and for instantaneous codes.
- \( \Rightarrow \) optimal codeword length bound still holds.
- In fact, this is not surprising since we can get arbitrarily close to entropy rate already using instantaneous code (e.g., Shannon code) with long block words.
- So we can then just consider instantaneous codes with relative impunity.
Shannon Code optimal?

- Ex: \( X' = \{0, 1\} \) with \( p(X = 0) = 10^{-1000} = 1 - p(X = 1) \).
- What are Shannon lengths?
  - \( \ell(0) = \lceil \log_2 10^{1000} \rceil = 3322 \) bits
  - \( \ell(1) = \lceil \log_2 (1 - 10^{1000}) \rceil = 1 \) bit
- For symbol 0, we’re using 3321 too many bits.
- In general, for other distributions, one can construct cases where \( \lceil \log_D p_i \rceil \) is longer than necessary.
- Shannon length codes are not optimal integer length prefix codes.

Huffman coding

- A procedure for finding shortest expected length prefix code.
- You’ve probably encountered it in computer science classes (a classic algorithm).
- Here we analyze it armed with the tools of information theory.
- Quest: given a \( p(x) \), find a code (bit strings and set of lengths) that is as short as possible, and also an instantaneous code (prefix free).
- We could do this greedily: start at the top and split the potential codewords into even probabilities (i.e., asking the question with highest entropy)
- This is similar to the game of 20 questions. We have a set of objects, w.l.o.g. the set \( S = \{1, 2, 3, 4, \ldots, m\} \) that occur with frequency proportional to non-negative \( (w_1, w_2, \ldots, w_m) \).
- We wish to determine an object from this class asking as few questions as possible.
- Supposing \( X \in S \), each question can take the form “Is \( X \in A? \)” for some \( A \subseteq S \).
### 20 Questions

- **Question tree.** \( S = \{x_1, x_2, x_3, x_4, x_5\} \).

```
   Y
  / \  /  \\
 X ∈ \{x_2\}  X ∈ \{x_3\}
   Y  N
   N  Y
 X ∈ \{x_2, x_3\}  X ∈ \{x_1\}  X ∈ \{x_4\}
```

- **How do we construct such a tree?** Charles Sanders Peirce, 1901 said:

  *Thus twenty skillful hypotheses will ascertain what two hundred thousand stupid ones might fail to do. The secret of the business lies in the caution which breaks a hypothesis up into its smallest logical components, and only risks one of them at a time.*

### The Greedy Method

- **This suggests a greedy method.** “Do next whatever currently looks best.”

- **Consider following table:**

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p )</td>
<td>0.01</td>
<td>0.24</td>
<td>0.05</td>
<td>0.20</td>
<td>0.47</td>
<td>0.01</td>
<td>0.02</td>
</tr>
</tbody>
</table>

- **The question that looks best would infer the most about the distribution, one with the largest entropy.**

- **If we consider the partition** \( \{a, b, c, d, e, f, g\} = \{a, b, c, d\} \cup \{e, f, g\} \), the question “Is \( X ∈ \{a, b, c, d\} \)?” would have maximum entropy since \( p(X ∈ \{a, b, c, d\}) = p(X ∈ \{e, f, g\}) = 0.5 \).

- **Once we have each answer, we would continue splitting to maximize entropy.** I.e., partition \( \{a, b, c, d\} = \{a, b\} \cup c, d \) since \( p(\{a, b\}) = p(\{c, d\}) = 1/4 \), and \( \{e, f, g\} = \{e\} ∪ \{f, g\} \) since that split maximizes entropy.
This leads to the following tree:

The expected length of this code $E\ell = 2.53$.

Entropy: $H = 1.93$. The Huffman procedure has $E\ell_{\text{huffman}} = 1.97$

Key problem: Greedy procedure is not optimal here.

**Huffman**

- Procedure: take the two least probable symbols in the alphabet.
- These two will be given the longest codewords, will have equal length, and will differ in the last digit.
- Combine these two symbols into a joint symbol having probability equal to the sum, add the joint symbol and then remove the two symbols, and repeat.
**Huffman**

- Ex: $\mathcal{X} = \{1, 2, 3, 4, 5\}$ with probabilities $\{1/4, 1/4, 1/5, 3/20, 3/20\}$.
- So 4 and 5 should have longest code length.
- The following figure describes the construction of a huffman tree, going from left to right.

<table>
<thead>
<tr>
<th>log $\frac{1}{p(x)}$</th>
<th>length</th>
<th>codeword</th>
<th>$\lambda$</th>
<th>prob</th>
<th>step 1</th>
<th>prob</th>
<th>step 2</th>
<th>prob</th>
<th>step 3</th>
<th>prob</th>
<th>step 4</th>
<th>prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0</td>
<td>2</td>
<td>00</td>
<td>1</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.55</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>2</td>
<td>10</td>
<td>2</td>
<td>0.25</td>
<td>0.25</td>
<td>0</td>
<td>0.45</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td>2</td>
<td>11</td>
<td>3</td>
<td>0.2</td>
<td>0.25</td>
<td>0</td>
<td>0.45</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.7</td>
<td>3</td>
<td>010</td>
<td>4</td>
<td>0.15</td>
<td>0.25</td>
<td>0</td>
<td>0.3</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.7</td>
<td>3</td>
<td>011</td>
<td>5</td>
<td>0.15</td>
<td>0.25</td>
<td>0</td>
<td>0.3</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- So we have $E\ell = 2.3$ bits and $H = 2.2855$ bits, as you can see this code does pretty well (close to entropy).
- Some code lengths are shorter/longer than $I(x) = \log \frac{1}{p(x)}$.
- Construction is similar for $D > 2$, in such case we might use dummy symbols in alphabet.