Ch. 2 Math Preliminaries for Lossless Compression

Section 2.2 Info Theory

Section 2.3 Models
Motivation for Info Theory & Models

- Gives Mathematical Foundation for Compression
  - How do we mathematically describe how much “information” is in “data”?
  - How do we model information and data?
- Provides Theoretical Limits for Compression
  - For a given type of data, what is the smallest number of bits that can be used to represent it?
  - What aspects of the data impact this lower bound?
- Motivates Practical Algorithms for Compression
  - Theoretically, what aspects of data can we exploit?
  - What kinds of processing is best?
  - How close do the real-world algorithms come to the theoretical limits
Section 2.2
Information Theory
Defining Information

Needed Characteristics:

• Probability is involved...
  – Rare events… convey large amounts of info
  – Common events… convey small amounts of info

• Info is additive for independent events
  – \( i(A \text{ and } B) = i(A) + i(B) \)

⇒ Define Info of an Event \( A \):

\[
i(A) = \log_2 \left( \frac{1}{P(A)} \right) \quad \text{(bits)}
\]

\[
= -\log_2 [P(A)]
\]

It is easy to verify that this definition satisfies the two “needed characteristics” above
Simplest Form of Avg. Info of a Source (Entropy)

Each source symbol conveys some amount of info – generally, not all symbols convey the same amount of info

Q: On average, how much info does a source put out per symbol

A: First consider a source $S$ putting out a stream of symbols (i.e., RVs) that are Independent & Identically Distributed (iid)…

Assume source “alphabet” consists of symbol set $\{A_1, A_2, \ldots, A_N\}$

Recall that the info of the $k$th symbol is $i(A_k) = -\log_2[P(A_k)]$

Then… the source’s average info is just the average of the symbol informations…

\[
\text{Source's Avg. Info } = E\{i(A_i)\} = \sum_{i=1}^{N} i(A_i)P(A_i)
\]

For iid source:

\[
H(S) = -\sum_{i=1}^{N} P(A_i)\log_2 [P(A_i)] \quad \text{(bits/symbol)}
\]

Entropy of the source = Avg. Info conveyed per symbol (in bits)
More General Form of Entropy of a Source

But… most real sources are **NOT** independent

(Recall: English text… prob. of next letter depends on current letter)

So… need to capture the (possibly infinite-order) dependence between subsequent symbols emitted by a source

We’ll use an asymptotic (or limiting) approach…

**Define 2\textsuperscript{nd}-Order Entropy**… “Avg. Info/Pair of Symbols”

\[
G_2(S) = -\sum_{i=1}^{N} \sum_{j=1}^{N} P(A_i, A_j) \log_2 \left[ P(A_i, A_j) \right] \quad \text{(bits/symbol-pair)}
\]

**Define 3\textsuperscript{rd}-Order Entropy**… “Avg. Info/Triple of Symbols”

\[
G_3(S) = -\sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{N} P(A_i, A_j, A_k) \log_2 \left[ P(A_i, A_j, A_k) \right] \quad \text{(bits/symbol-triple)}
\]

- Etc. 4\textsuperscript{th}, 5\textsuperscript{th}, 6\textsuperscript{th}, …
-
Now… Need to convert each of these into a “per symbol” form:

Define: \( H_n(S) = \frac{1}{n} G_n(S) \) (bits/symbol)

As we increase \( n \) we capture more and more of the inter-symbol structure

“\( n^{\text{th}} \)-order Entropy”

In the limit we capture all the structure…

Define the Entropy of the Source \( S \) to be:

\[
H(S) = \lim_{n \to \infty} H_n(S) = \lim_{n \to \infty} \frac{1}{n} G_n(S) \quad \text{(bits/symbol)}
\]

(For General Source)

Can verify that if the source is iid, then this general entropy collapses to

\[
H(S) = \lim_{n \to \infty} H_n(S) = -\sum_{i=1}^{N} P(A_i) \log_2 [P(A_i)]
\]

Same as we defined before for iid!
Typical Behavior of General Form of Entropy

For non-independent source:
At first… goes up slower than linear
Later… Goes up linearly with slope of $H(s)$

$H_n(S) = G_n(S)/n$

Asymptotically approaches $H(S)$ from above

A decreasing “return” as the order increases!
Two Views of Entropy $H(S)$

- Expresses the Avg Info per symbol conveyed by source
  - Although each symbol conveys a different amount of information, $H(S)$ gives the overall average amount
  - Sources with the same number of symbols can have different entropy values
- $H(S)$ gives a Lower Bound on the average # of bits/symbol needed to code the source
  - This provides a measure of what the best level of compression we can expect for a given source

Recall “Lossless Example to Motivate…”
If the source is assumed to be iid, with symbol probabilities of

\[
P(A) = 0.5 \quad P(B) = 0.25 \quad P(C) = 0.125 \quad P(D) = 0.125
\]

Then $H(S) = 1.75$ bits… which is exactly what our best decodable code achieved!!!

So… $H(S)$ tells us exactly the lowest rate the best lossless compression scheme can compress source $S$ to!!!
Section 2.3
Models
Type of Models

What kind of models can capture the prob. structure of a source?

1. **Prob. Model**... Can’t capture symbol-to-symbol dependence
   - Symbols: $A_1, A_2, \ldots, A_N$
   - Probabilities: $P(A_1), P(A_2), \ldots, P(A_N)$

2. **Joint Prob. Model**... Can capture sym-to-sym dependence
   - 2nd-Order Model:
     $P(A_1, A_1), P(A_1, A_2), \ldots, P(A_1, A_N), P(A_2, A_1), \ldots, P(A_2, A_N), \ldots, P(A_N, A_N)$
   - 3rd-Order Model:
     $P(A_1, A_1, A_1), P(A_1, A_1, A_2), \ldots, P(A_1, A_1, A_N), \ldots, P(A_N, A_N, A_N)$
   - Etc., Etc., Etc…

1st-Order Model? Just a Prob. Model…
… Called 1st-Order Prob Model
3. **Cond. Prob. Model**... Can capture sym-to-sym dependence

Called a “Context Model”… Most useful model of the three…

A special Cond. Prob. Model is called a “Markov Model” (MM) which is also called a “Discrete-Time Markov Chain”

Suppose that the source output sequence is $x_1, x_2, x_3, \ldots$ and each $x_i$ can take on any symbol $A_1, A_2, \ldots, A_N$.

- **1st-Order MM** – Context beyond one symbol has no effect:
  
  \[
  P(x_n|x_{n-1}, x_{n-2}, x_{n-3}, x_{n-4}, \ldots) = P(x_n|x_{n-1})
  \]

- **k**th-Order MM:
  \[
  P(x_n|x_{n-1}, x_{n-2}, \ldots, x_{n-k}, x_{n-(k+1)}, \ldots) = P(x_n|x_{n-1}, x_{n-2}, \ldots, x_{n-k})
  \]

Note: A 0th-Order MM is just a 1st-Order Prob. Model

If source really does have only finite context, what is its entropy? For 1st-Order MM the result of finding $H(S)$ via the limiting approach is:

\[
H(S) = - \sum_{i=1}^{N} \sum_{j=1}^{N} P(A_i, A_j) \log_2 \left[ P(A_i \mid A_j) \right]
\]
Using MMs for Compression

How does one use a $k^{th}$-Order MM for compression?

For an $N$ symbol alphabet… build a set of $N$ codewords for each $k$-symbol context… Called “Context Coding”

There will be $N^k$ contexts… so there will be $N^{k+1}$ codewords!

In building compression algorithms… there is a trade-off: model complexity vs. model accuracy

Higher-order MM more accurately captures source structure… … BUT… increases the number of codewords
Example: Context Coding via 1\textsuperscript{st}–Order MM

Consider in English text... how should one code the letter \( u \)?

It depends on the context...

If current letter = \( q \)… then next letter is very likely \( u \)…

\( \Rightarrow \) in this context the codeword for \( u \) should be short

But… if current letter = \( u \)… then next letter is unlikely \( u \)...

\( \Rightarrow \) in this context the codeword for \( u \) should be long

Note that the decoder – once it decodes the current symbol – can choose the correct set of codewords to decode the next symbol

Example: 3-symbol alphabet, source model is 1\textsuperscript{st}-Order MM

\( \text{(Assume that repeated symbols are not likely to occur)} \)

<table>
<thead>
<tr>
<th>Current Symbol / Next Symbol</th>
<th>( A_1 )</th>
<th>( A_2 )</th>
<th>( A_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A_1 )</td>
<td>1</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>( A_2 )</td>
<td>00</td>
<td>1</td>
<td>01</td>
</tr>
<tr>
<td>( A_3 )</td>
<td>00</td>
<td>01</td>
<td>1</td>
</tr>
</tbody>
</table>

Codewords for next symbol conditioned on current symbol
Demo of Text Generation Via Models

A model that accurately captures a source’s conditional probability structure is useful for compression... a way to visualize how well a model captures this structure is to use the model to generate a symbol sequence and see if it “looks right”

This is most easily visually checked for the case of English text...

- For 1\textsuperscript{st}-Order Prob. Model
  - Use computer to analyze large sample of text to estimate prob. of each symbol: \( P(a) = \) (# of a’s)/(total # of characters)
  - Then… generate stream of symbols drawn according to measured prob.
- For \( k \textsuperscript{th}\)-Order MM
  - Use computer to analyze large sample of text to estimate prob. of each symbol under each k-symbol context: for example, for \( k = 1 \)
    \[
    P(a|b) = \frac{(# \text{ of } a \text{'s preceded by } b)}{(\text{total } \# \text{ of characters preceded by } b)}
    \]
"0th-Order" Probability Model
(i.e., Assume equal probabilities for a-z & space)

```
>> char(gen_text_0(70))

tgpgmmvi lhaedxyx wzgkpsufeflmcqsjehhantxymmwzwxuxghmimiixdbmrsviionp

>> char(gen_text_0(70))

xpcnv gvaelshgvbprabbdzcvrve qmganfcdqdrzdiycxdbhkyrkjdklq vbtd jhrcw

>> char(gen_text_0(70))

dsxuwnqjvnywokiaxtjmtjwcmotln oqcxbkzllwshhifwkvjs lsoazhazmlrzshjfd

>> char(gen_text_0(70))

andjmaxgxkphlqitpjwugpt ahfqfklahakcwmzmzpyrgfwglnrsicojmthixmtjxtmrt

>> char(gen_text_0(70))

dwndroffhdpbjfxukvvpqwodrsaqwiquylyxvntkoupisqlcamrjxcstrpkzqupkpnxyo
```
1st-Order Probability Model
(i.e., estimated probabilities for a-z, A-Z & space)

```python
>>> char(gen_text(probs,70))
oe pciA otwcrfa art bb fw iufnktl uibldt is slPe  d sto faeb  Lldeo st
```

```python
>>> char(gen_text(probs,70))
Ika  lo ekosa eocekdptbcfy manoAaomrxe riu c cs eieucl Za cs impiakoj
```

```python
>>> char(gen_text(probs,70))
pjd dot stlAa o ue gil  oe k  sl ourls islllaelsadu m g  snb  yh e
```

```python
>>> char(gen_text(probs,70))
pjd dot stlAa o ue gil  oe k  sl ourls islllaelsadu m g  snb  yh e
```

```python
>>> char(gen_text(probs,70))
x Hcnd  ceh eelacoulsbrke atmitr kun i ekes tal eaddp  nootltiNot iac
```
1st-Order Markov Model – Conditioned on Single Letter
(i.e., estimated cond. probs. for a-z, A-Z & space)

>>> char(gen_text_2(cond_probs_2,70))

ho lenaf b me t chontha ffes ntarub fulep eot Errol sibl uxiritha fte

>>> char(gen_text_2(cond_probs_2,70))

flug cee Roly nararme sp grirme sork d ak focamase asnolot bartan hov

>>> char(gen_text_2(cond_probs_2,70))

se fu slovelop t ta d gi Morn puea schel be friilgud a d pe Prilabe b

>>> char(gen_text_2(cond_probs_2,70))

cmel wntoi aga f bube mewil ceng g Dave gilar peedeazy st h wl cist m

>>> char(gen_text_2(cond_probs_2,70))

lly Munebest hn rte leara scujowh Kuram t Honologri b Lithy cr acaval
2\textsuperscript{nd}-Order Markov Model – Conditioned on Letter Pairs
(i.e., estimated cond. probs. for a-z, A-Z & space)

\begin{verbatim}
>> char(gen_text_3(cond_probs_2,cond_probs_3,70,MAX))

loft dick patace preend buff Cob Rit Goga co Kan brack hast kinc fieu

>> char(gen_text_3(cond_probs_2,cond_probs_3,70,MAX))

te dump flip cam boup hoodeem tald twilt Hoyawn warchiz Boin Mact tid

>> char(gen_text_3(cond_probs_2,cond_probs_3,70,MAX))

mock lubeate clavy dary ick sivarsart fown solavere dod gi Nage Luke

>> char(gen_text_3(cond_probs_2,cond_probs_3,70,MAX))

wearn la sk Haass Ohin Chowl west se deub Guada molext heap dierk sai

>> char(gen_text_3(cond_probs_2,cond_probs_3,70,MAX))

walift Peny lue it cork faile gly wass Age jaxon Gaink hoy aiteck rit
\end{verbatim}