High order entropy

Ben Langmead

Please sign guestbook (www.langmead-lab.org/teaching-materials) to tell me briefly how you are using the slides. For original Keynote files, email me (ben.langmead@gmail.com).
High-order entropy

Zero order empirical entropy seems insufficient when context matters

<table>
<thead>
<tr>
<th>Bigram frequency per 40,000 words</th>
</tr>
</thead>
<tbody>
<tr>
<td>th</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>1.52</td>
</tr>
<tr>
<td>0.94</td>
</tr>
<tr>
<td>0.82</td>
</tr>
<tr>
<td>0.63</td>
</tr>
<tr>
<td>0.57</td>
</tr>
<tr>
<td>0.56</td>
</tr>
<tr>
<td>0.55</td>
</tr>
</tbody>
</table>

High-order entropy

Can compress better if we consider \textit{context}

Let $C$ change depending on surrounding symbols

For $k$ symbols of context, we have codes $C_i \in \{ C_{\Sigma^k} \}$
High-order entropy

Could consider context to the right →

$$C_{\text{tcg}}(a) \quad C_{\text{g cg}}(a) \quad C_{\text{tcg}}(a) \quad C_{\text{tcg}}(a) \quad C_{\text{tct}}(a)$$

Or context to the left ←

$$C_{\text{tct}}(a) \quad C_{\text{c gg}}(a) \quad C_{\text{gtt}}(a) \quad C_{\text{tcg}}(a) \quad C_{\text{gc g}}(a)$$
High-order entropy

How should we build each code \( \{ C_{\Sigma^k} \} \)?

Same as before, but with frequencies *conditioned on context*

E.g. \( C_{\text{gca}} \) is built considering the number of times each symbol occurs just after gca.
High-order entropy

\texttt{abracadabraabraabraabraabra}

Let $S_a$ be the substring we get by concatenating characters just after the a's

$$S_a = \text{bcdbabcdb}$$

Build code $C_a$ using frequencies in $S_a$:

\{a : 1, b : 4, c : 2, d : 2, r : 0\}

(r won't get a code)
High-order entropy

\{a : 1, b : 4, c : 2, d : 2, r : 0\}
High-order entropy

\{a : 1, b : 4, c : 2, d : 2, r : 0\}
High-order entropy

\{a : 1, b : 4, c : 2, d : 2, r : 0\}
High-order entropy

\{ a : 1, b : 4, c : 2, d : 2, r : 0 \}
High-order entropy

\{a : 1, b : 4, c : 2, d : 2, r : 0\}

\[ C_a(a) = 000 \]
\[ C_a(d) = 001 \]
\[ C_a(c) = 01 \]
\[ C_a(b) = 1 \]
High-order entropy

bracadabraabracadabra

\[ S_a = bcdbabcdb \]
\[ \{ a : 1, b : 4, c : 2, d : 2 \} \]
\[ C_a(a) = 000 \]
\[ C_a(d) = 001 \]
\[ C_a(c) = 01 \]
\[ C_a(b) = 1 \]

\[ S_b = rrrr \quad \{ r : 4 \} \]
\[ \text{(no code)} \]
\[ S_c = aa \quad \{ a : 2 \} \]
\[ \text{(no code)} \]
\[ S_d = aa \quad \{ a : 2 \} \]
\[ \text{(no code)} \]
\[ S_r = aaaa \quad \{ a : 4 \} \]
\[ \text{(no code)} \]
High-order entropy

abra(cadabra)abra(cadabra)

Context:

\[ C_a(a) = 000 \]
\[ C_a(b) = 001 \]
\[ C_a(c) = 01 \]
\[ C_a(d) = 1 \]

Codes based on \( H_0 \)

(no codes required)
High-order entropy

\[ S_i = \text{sspmssp} \]
\[ \{s: 4, p: 2, m: 1\} \]
\[ C_i(p) = 00 \]
\[ C_i(m) = 01 \]
\[ C_i(s) = 1 \]

\[ S_m = \text{ii} \quad \{i: 2\} \]
\[ \text{(no code)} \]

\[ S_p = \text{pipi} \]
\[ \{p: 2, i: 2\} \]
\[ C_p(p) = 0 \]
\[ C_p(i) = 1 \]

\[ S_s = \text{sisisisi} \]
\[ \{s: 4, i: 4\} \]
\[ C_s(s) = 0 \]
\[ C_s(i) = 1 \]
High-order entropy

mississippimississippi

\[ C_i(p) = 00 \]
\[ C_i(m) = 01 \]
\[ C_i(s) = 1 \]

\[ C_p(p) = 0 \quad C_s(s) = 0 \]
\[ C_p(i) = 1 \quad C_s(i) = 1 \]
High-order entropy

Def'n of high-order empirical entropy $H_k$ is similarly hierarchical

Contexts:

Codes achieving near-$H_0$ given context
High-order entropy

$H_k$ of a length-$n$ string $S$ is a weighted sum over all contexts of the zero order empirical entropy of symbols having that context

$$H_k(S) = \sum_{t \in \Sigma^k} \frac{|S_t|}{n} \cdot H_0(S_t) \quad \text{for } k > 0$$

$S$ is the entire string, $S_t$ is the concatenation of symbols having context $t$
High-order entropy

Right-context: a

\[ C_\text{a}(\text{a}) = 000 \]
\[ C_\text{a}(\text{d}) = 001 \]
\[ C_\text{a}(\text{c}) = 01 \]
\[ C_\text{a}(\text{b}) = 1 \]

Schemes like this can compress to \( \leq n(H_k(S) + 1) \) bits

With added overhead of switching between many codes
High-order entropy

<table>
<thead>
<tr>
<th>Collection</th>
<th>H0</th>
<th>H1</th>
<th>H2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE SOURCES</td>
<td>5.537 (69.21%)</td>
<td>4.038 (50.48%)</td>
<td>3.012 (37.65%)</td>
</tr>
<tr>
<td>MIDI</td>
<td>5.633 (70.41%)</td>
<td>4.734 (59.18%)</td>
<td>4.139 (51.74%)</td>
</tr>
<tr>
<td>PROTEINS</td>
<td>4.195 (52.44%)</td>
<td>4.173 (52.16%)</td>
<td>4.146 (51.82%)</td>
</tr>
<tr>
<td>DNA</td>
<td>1.982 (24.78%)</td>
<td>1.935 (24.19%)</td>
<td>1.925 (24.06%)</td>
</tr>
<tr>
<td>ENGLISH</td>
<td>4.529 (56.61%)</td>
<td>3.606 (45.08%)</td>
<td>2.922 (36.53%)</td>
</tr>
<tr>
<td>XML</td>
<td>5.230 (65.37%)</td>
<td>3.294 (41.17%)</td>
<td>2.007 (25.09%)</td>
</tr>
</tbody>
</table>

Empirical entropies for 6 texts. Values are bits-per-symbol, percentages are ratios compared to ASCII.

High-order entropy

\(H_k\) encoding reaches into the string, extracting "structure" needed to compress well

\(k\) balances compression with overhead

**Grouping** principle at play

\(H_0\)-based methods are simpler, faster, require less memory, but can't find as much structure as \(H_k\)

...or...can they? **Order** to the rescue