Language Modeling

Knersons-Ney Smoothness & KenLM

by Rong Zhou
Language Modeling review

-Why need it: e.g. “I am going to bed” and “I am going to bad”

-What it can do: sign probability for a sentence

\[ S = w_1, w_2, \ldots, w_m. \]

n-gram:

\[
P(S) = \prod_{t=1}^{m} P(w_t|w_{1:t-1})
\]

\[
P(w_t|w_{1:t-1}) = \frac{C(w_{1:t})}{C(w_{1:t-1})}
\]

\[
P(S) = \prod_{t=1}^{m} P(w_t|w_{t-n+1:t-1})
\]

\[
P(w_t|w_{t-n+1:t-1}) = \frac{C(w_{t-n+1:t})}{C(w_{t-n+1:t-1})}
\]

-Category: frequency based:
  - n-gram model

neural network based:
  - feedforward model (n-gram based)
  - recurrent neural network model
Absolute discounting smoothing:

\[ d \in [0, 1] \]

\[
P_{\text{abs}}(w_t|w_{t-n+1:t-1}) = \frac{\max(C(w_{t-n+1:t}) - d, 0)}{C'(w_{t-n+1:t-1})} + \lambda(w_{t-n+1:t-1})P_{\text{abs}}(w_t|w_{t-n+2:t-1})
\]

\( \lambda(w_{t-n+1:t-1}) \) is the penalty for the lower order ngram \( w_{t-n+2:t} \).
Knersy-Ney Smoothing (KN):

- backoff the bigram "reading _" to its unigram "_

eg. I cannot see without my reading _.

- continuation smoothing:

\[ P_{\text{con}}(w_t|w_{t-n+2:t-1}) = \frac{|\{w : C(w, w_{t-n+2:t}) > 0\}|}{|\{w_{t-n+1:t} : C(w_{t-n+1:t}) > 0\}|} \]  \hspace{1cm} (7)

- continuation probability:

\[ P(w_t|w_{t-n+1:t-1}) = \frac{\max(C(w_{t-n+1:t}) - d, 0)}{C(w_{t-n+1:t-1})} \]

\[ + \lambda(w_{t-n+1:t-1}) P_{\text{con}}(w_t|w_{t-n+2:t-1}) \]  \hspace{1cm} (8)
In real world scenario, $P_{con}(w_t|w_{t-n+2:t-1})$ may still be zero! So we can further improve Equation 8 and write it in a recursive fashion:

$$P_{KN}(w_t|w_{t-n+1:t-1}) = \frac{\max(C_{KN}(w_{t-n+1:t}) - d, 0)}{C_{KN}(w_{t-n+1:t-1})}$$

$$+ \lambda(w_{t-n+1:t-1}) P_{KN}(w_t|w_{t-n+2:t-1})$$

(9)

$$C_{KN}(\bullet) = \begin{cases} C(\bullet) & \text{for the highest order} \\ |\{w : C(w, \bullet) > 0\}| & \text{for the lower order} \end{cases}$$

(10)

Where

$$\lambda(w_{t-n+1:t-1}) = \frac{d}{C(w_{t-n+1:t-1})} |\{w : C(w_{t-n+1:t-1}, w) > 0\}|$$

(11)
KenLM

Hash tables and PROBING:

- $O\left(\frac{m}{m-1}\right)$imption:

$$(96m + 64)c_1 + 128m \sum_{n=2}^{N-1} c_n + 96mc_N. \quad (12)$$
Stored Arrays and TRIE:

-time consumption:

\[ O(\log \log |A|) \]

-memory consumption:

\[
(32 + 32 + 64 + 64)c_1 + \\
\sum_{n=2}^{N-1} (\left\lfloor \log_2 c_1 \right\rfloor + q + r + \left\lfloor \log_2 c_{n+1} \right\rfloor)c_n + \\
(\left\lfloor \log_2 c_1 \right\rfloor + q)c_N
\]
Experimental result consist with the hypothesis

<table>
<thead>
<tr>
<th>Package</th>
<th>Variant</th>
<th>Time (m)</th>
<th>RAM (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>CPU</td>
<td>Wall</td>
</tr>
<tr>
<td>Ken</td>
<td>PROBING-L</td>
<td>72.3</td>
<td>72.4</td>
</tr>
<tr>
<td></td>
<td>PROBING-P</td>
<td>73.6</td>
<td>74.7</td>
</tr>
<tr>
<td></td>
<td>TRIE-L</td>
<td>80.4</td>
<td>80.6</td>
</tr>
<tr>
<td></td>
<td>TRIE-P</td>
<td>80.1</td>
<td>80.1</td>
</tr>
<tr>
<td></td>
<td>TRIE-L 8(^a)</td>
<td>79.5</td>
<td>79.5</td>
</tr>
<tr>
<td></td>
<td>TRIE-P 8(^a)</td>
<td>79.9</td>
<td>79.9</td>
</tr>
<tr>
<td>SRI</td>
<td>Default</td>
<td>85.9</td>
<td>86.1</td>
</tr>
<tr>
<td></td>
<td>Compact</td>
<td>155.5</td>
<td>155.7</td>
</tr>
<tr>
<td></td>
<td>Cache-Invert-L</td>
<td>106.4</td>
<td>106.5</td>
</tr>
<tr>
<td></td>
<td>Cache-Invert-R</td>
<td>106.7</td>
<td>106.9</td>
</tr>
<tr>
<td>IRST</td>
<td>Invert-L</td>
<td>117.2</td>
<td>117.3</td>
</tr>
<tr>
<td></td>
<td>Invert-R</td>
<td>117.7</td>
<td>118.0</td>
</tr>
<tr>
<td></td>
<td>Default-L</td>
<td>126.3</td>
<td>126.4</td>
</tr>
<tr>
<td></td>
<td>Default-R</td>
<td>127.1</td>
<td>127.3</td>
</tr>
<tr>
<td>Rand</td>
<td>Backoff(^a)</td>
<td>277.9</td>
<td>278.0</td>
</tr>
<tr>
<td></td>
<td>Backoff(^b)</td>
<td>247.6</td>
<td>247.8</td>
</tr>
</tbody>
</table>

Ref. Kenneth Heafield
Summary

The performance of KN is very close to the state of the art. That is due to its continuation probability calculation; The contribution of KenLm mainly lies in the two kinds of data structure: Hash Table & PROBING and Sorted Arrays & TRIE. All of them laid the fundamention for language modeling.