Language Models

Advanced Machine Learning for NLP
Jordan Boyd-Graber
FOUNDATIONS
After this class, you’ll be able to:

• Give examples of where we need language models
• Evaluate language models
• Connection between Bayesian nonparametrics and backoff
Language models

- **Language models** answer the question: *How likely is a string of English words good English?*
- Autocomplete on phones and websearch
- Creating English-looking documents
- Very common in machine translation systems
  - Help with reordering / style
    \[ p_{\text{lm}}(\text{the house is small}) > p_{\text{lm}}(\text{small the is house}) \]
  - Help with word choice
    \[ p_{\text{lm}}(\text{I am going home}) > p_{\text{lm}}(\text{I am going house}) \]
Why language models?

- Like sorting for computer science
- Language models essential for many NLP applications
- Optimized for performance and runtime
N-Gram Language Models

- Given: a string of English words $W = w_1, w_2, w_3, ..., w_n$
- Question: what is $p(W)$?
- Sparse data: Many good English sentences will not have been seen before
  - Decomposing $p(W)$ using the chain rule:

$$p(w_1, w_2, w_3, ..., w_n) = p(w_1) p(w_2|w_1) p(w_3|w_1, w_2) ... p(w_n|w_1, w_2, ... w_{n-1})$$

(not much gained yet, $p(w_n|w_1, w_2, ... w_{n-1})$ is equally sparse)
Markov Chain

- **Markov independence assumption:**
  - only previous history matters
  - limited memory: only last $k$ words are included in history (older words less relevant)
  \[
  \rightarrow \text{ } k\text{-th order Markov model}
  \]

- For instance 2-gram language model:
  \[
  p(w_1, w_2, w_3, \ldots, w_n) \approx p(w_1) p(w_2|w_1) p(w_3|w_2) \ldots p(w_n|w_{n-1})
  \]

- What is conditioned on, here $w_{i-1}$ is called the **history**
How good is the LM?

- A good model assigns a text of real English $W$ a high probability
- This can be also measured with **perplexity**

\[
\text{perplexity}(W) = P(w_1, \ldots w_N)^{-\frac{1}{N}}
\]

\[
= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}
\]
## Comparison 1–4-Gram

<table>
<thead>
<tr>
<th>word</th>
<th>unigram</th>
<th>bigram</th>
<th>trigram</th>
<th>4-gram</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>6.684</td>
<td>3.197</td>
<td>3.197</td>
<td>3.197</td>
</tr>
<tr>
<td>would</td>
<td>8.342</td>
<td>2.884</td>
<td>2.791</td>
<td>2.791</td>
</tr>
<tr>
<td>like</td>
<td>9.129</td>
<td>2.026</td>
<td>1.031</td>
<td>1.290</td>
</tr>
<tr>
<td>to</td>
<td>5.081</td>
<td>0.402</td>
<td>0.144</td>
<td>0.113</td>
</tr>
<tr>
<td>commend</td>
<td>15.487</td>
<td>12.335</td>
<td>8.794</td>
<td>8.633</td>
</tr>
<tr>
<td>the</td>
<td>3.885</td>
<td>1.402</td>
<td>1.084</td>
<td>0.880</td>
</tr>
<tr>
<td>reporter</td>
<td>10.840</td>
<td>7.319</td>
<td>2.763</td>
<td>2.350</td>
</tr>
<tr>
<td>.</td>
<td>4.896</td>
<td>3.020</td>
<td>1.785</td>
<td>1.510</td>
</tr>
<tr>
<td>&lt;/s&gt;</td>
<td>4.828</td>
<td>0.005</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>average</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>perplexity</td>
<td>265.136</td>
<td>16.817</td>
<td>6.206</td>
<td>4.758</td>
</tr>
</tbody>
</table>
Example: 3-Gram

- Counts for trigrams and estimated word probabilities

<table>
<thead>
<tr>
<th>word</th>
<th>c.</th>
<th>prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>cross</td>
<td>123</td>
<td>0.547</td>
</tr>
<tr>
<td>tape</td>
<td>31</td>
<td>0.138</td>
</tr>
<tr>
<td>army</td>
<td>9</td>
<td>0.040</td>
</tr>
<tr>
<td>card</td>
<td>7</td>
<td>0.031</td>
</tr>
<tr>
<td>,</td>
<td>5</td>
<td>0.022</td>
</tr>
</tbody>
</table>

- 225 trigrams in the Europarl corpus start with **the red**
- 123 of them end with **cross**

→ maximum likelihood probability is \( \frac{123}{225} = 0.547 \).
### Example: 3-Gram

- Counts for trigrams and estimated word probabilities

<table>
<thead>
<tr>
<th>word</th>
<th>c.</th>
<th>prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>cross</td>
<td>123</td>
<td>0.547</td>
</tr>
<tr>
<td>tape</td>
<td>31</td>
<td>0.138</td>
</tr>
<tr>
<td>army</td>
<td>9</td>
<td>0.040</td>
</tr>
<tr>
<td>card</td>
<td>7</td>
<td>0.031</td>
</tr>
<tr>
<td>,</td>
<td>5</td>
<td>0.022</td>
</tr>
</tbody>
</table>

- 225 trigrams in the Europarl corpus start with **the red**
- 123 of them end with **cross**

→ maximum likelihood probability is \( \frac{123}{225} = 0.547 \).

- Can’t use ML estimate
How do we estimate a probability?

• Assuming a **sparse Dirichlet** prior, \( \alpha < 1 \) to each count

\[
\theta_i = \frac{n_i + \alpha_i}{\sum_k n_k + \alpha_k}
\]  

(1)

• \( \alpha_i \) is called a smoothing factor, a pseudocount, etc.
How do we estimate a probability?

- Assuming a **sparse Dirichlet** prior, $\alpha < 1$ to each count
  
  $$\theta_i = \frac{n_i + \alpha_i}{\sum_k n_k + \alpha_k} \quad (1)$$

- $\alpha_i$ is called a smoothing factor, a pseudocount, etc.
- When $\alpha_i = 1$ for all $i$, it’s called “Laplace smoothing”
How do we estimate a probability?

- Assuming a **sparse Dirichlet** prior, $\alpha < 1$ to each count

$$\theta_i = \frac{n_i + \alpha_i}{\sum_k n_k + \alpha_k}$$  \hspace{1cm} (1)

- $\alpha_i$ is called a smoothing factor, a pseudocount, etc.
- When $\alpha_i = 1$ for all $i$, it’s called “Laplace smoothing”
- What is a good value for $\alpha$?
- Could be optimized on held-out set to find the “best” language model
Example: 2-Grams in Europarl

<table>
<thead>
<tr>
<th>Count</th>
<th>Adjusted count</th>
<th>Test count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>((c + 1))</td>
<td>((c + \alpha))</td>
</tr>
<tr>
<td>0</td>
<td>0.00378</td>
<td>0.00016</td>
</tr>
<tr>
<td>1</td>
<td>0.00755</td>
<td>0.95725</td>
</tr>
<tr>
<td>2</td>
<td>0.01133</td>
<td>1.91433</td>
</tr>
<tr>
<td>3</td>
<td>0.01511</td>
<td>2.87141</td>
</tr>
<tr>
<td>4</td>
<td>0.01888</td>
<td>3.82850</td>
</tr>
<tr>
<td>5</td>
<td>0.02266</td>
<td>4.78558</td>
</tr>
<tr>
<td>6</td>
<td>0.02644</td>
<td>5.74266</td>
</tr>
<tr>
<td>8</td>
<td>0.03399</td>
<td>7.65683</td>
</tr>
<tr>
<td>10</td>
<td>0.04155</td>
<td>9.57100</td>
</tr>
<tr>
<td>20</td>
<td>0.07931</td>
<td>19.14183</td>
</tr>
</tbody>
</table>
Example: 2-Grams in Europarl

<table>
<thead>
<tr>
<th>Count</th>
<th>Adjusted count</th>
<th>Test count</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c)</td>
<td>((c + 1))</td>
<td>((c + \alpha))</td>
</tr>
<tr>
<td>0</td>
<td>0.00378</td>
<td>0.00016</td>
</tr>
<tr>
<td>1</td>
<td>0.00755</td>
<td>0.95725</td>
</tr>
<tr>
<td>2</td>
<td>0.01133</td>
<td>1.91433</td>
</tr>
<tr>
<td>3</td>
<td>0.01511</td>
<td>2.87141</td>
</tr>
<tr>
<td>4</td>
<td>0.01888</td>
<td>3.82850</td>
</tr>
</tbody>
</table>

Can we do better?
In higher-order models, we can learn from similar contexts!

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.03399</td>
<td>7.65683</td>
</tr>
<tr>
<td>10</td>
<td>0.04155</td>
<td>9.57100</td>
</tr>
<tr>
<td>20</td>
<td>0.07931</td>
<td>19.14183</td>
</tr>
</tbody>
</table>
Back-Off

• In given corpus, we may never observe
  ◦ Scottish beer drinkers
  ◦ Scottish beer eaters
• Both have count 0
  → our smoothing methods will assign them same probability
• Better: backoff to bigrams:
  ◦ beer drinkers
  ◦ beer eaters
Interpolation

• Higher and lower order n-gram models have different strengths and weaknesses
  ◦ high-order n-grams are sensitive to more context, but have sparse counts
  ◦ low-order n-grams consider only very limited context, but have robust counts
• Combine them

\[
p_I(w_3|w_1, w_2) = \lambda_1 p_1(w_3) \\
+ \lambda_2 p_2(w_3|w_2) \\
+ \lambda_3 p_3(w_3|w_1, w_2)
\]
Back-Off

- Trust the highest order language model that contains n-gram

\[
p^{BO}_n(w_i|w_{i-n+1},...,w_{i-1}) =
\begin{cases}
  \alpha_n(w_i|w_{i-n+1},...,w_{i-1}) & \text{if } \text{count}_n(w_{i-n+1},...,w_i) > 0 \\
  d_n(w_{i-n+1},...,w_{i-1}) p^{BO}_{n-1}(w_i|w_{i-n+2},...,w_{i-1}) & \text{else}
\end{cases}
\]

- Requires
  - adjusted prediction model \( \alpha_n(w_i|w_{i-n+1},...,w_{i-1}) \)
  - discounting function \( d_n(w_1,\ldots,w_{n-1}) \)
What’s a word?

- There are an infinite number of words
  - Possible to develop generative story of how new words are created
  - Bayesian non-parametrics
What's a word?

- There are an infinite number of words
  - Possible to develop generative story of how new words are created
  - Bayesian non-parametrics
- Defining a vocabulary (the event space)
- But how do you handle words outside of your vocabulary?
What’s a word?

• There are an infinite number of words
  ○ Possible to develop generative story of how new words are created
  ○ Bayesian non-parametrics

• Defining a vocabulary (the event space)

• But how do you handle words outside of your vocabulary?
  ○ Ignore? You could win just by ignoring everything
  ○ Standard: replace with $<$UNK$>$ token

• Next week: word representations!
Reducing Vocabulary Size

- For instance: each number is treated as a separate token
- Replace them with a number token num
  - but: we want our language model to prefer

\[
p_{lm}(I \text{ pay } 950.00 \text{ in May } 2007) > p_{lm}(I \text{ pay } 2007 \text{ in May } 950.00)
\]
  - not possible with number token

\[
p_{lm}(I \text{ pay num in May num}) = p_{lm}(I \text{ pay num in May num})
\]

- Replace each digit (with unique symbol, e.g., @ or 5), retain some distinctions

\[
p_{lm}(I \text{ pay } 555.55 \text{ in May } 5555) > p_{lm}(I \text{ pay } 5555 \text{ in May } 555.55)
\]