N-gram models

- Unsmoothed n-gram models (review)
- Smoothing
  - Add-one (Laplacian)
  - Good-Turing
- Unknown words
- Evaluating n-gram models
- Combining estimators
  - (Deleted) interpolation
  - Backoff

Goals

- Determine the next word in a sequence
  - Probability distribution across all words in the language
  - \( P \left( w_n \mid w_1 w_2 \ldots w_{n-1} \right) \)
- Determine the probability of a sequence of words
  - \( P \left( w_1 w_2 \ldots w_{n-1} w_n \right) \)

Probability of a word sequence

- \( P \left( w_1 w_2 \ldots w_{n-1} w_n \right) \)
  \[
  P(w_1^n) = P(w_1) \ P(w_2|w_1) \ P(w_3|w_2^2) \ldots \ P(w_n|w_{n-1}^{n-1})
  = \prod_{k=1}^{n} P(w_k|w_1^{k-1})
  \]
- Problem?
- Solution: *approximate* the probability of a word given all the previous words...

N-gram approximations

- Bigram model
  \[
  P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-1})
  \]
- Trigram model
  \[
  P(w_n \mid w_1^{n-2}) \approx P(w_n \mid w_{n-2} w_{n-1})
  \]
- Probability of a word sequence
  \[
  P(w_1^n) = P(w_1) \ P(w_2|w_1) \ P(w_3|w_2^2) \ldots \ P(w_n|w_1^{n-1})
  = \prod_{k=1}^{n} P(w_k|w_1^{k-1})
  \]
- General form
  \[
  P(w_1^n) \approx \prod_{k=1}^{n} P(w_k|w_1^{k-1-N+1})
  \]
Training N-gram models

- N-gram models can be trained by counting and normalizing
  - Bigrams
    \[ P(w_n | w_{n-1}) = \frac{\text{count}(w_{n-1}w_n)}{\text{count}(w_{n-1})} \]
  - General case
    \[ P(w_n | w_{n-N+1}) = \frac{\text{count}(w_{n-N+1}w_n)}{\text{count}(w_{n-N+1})} \]

  → An example of Maximum Likelihood Estimation (MLE)
    » Resulting parameter set is one in which the likelihood of the training set T given the model M (i.e. \( P(T|M) \)) is maximized.

Bigram counts: MLE

- Note the number of 0’s…

N-gram models

- Unsmoothed n-gram models (review)
- Smoothing
  - Add-one (Laplacian)
  - Good-Turing
- Unknown words
- Evaluating n-gram models
- Combining estimators
  - (Deleted) interpolation
  - Backoff

Smoothing

- Need better estimators than MLE for rare events
- Approach
  - Somewhat decrease the probability of previously seen events, so that there is a little bit of probability mass left over for previously unseen events
    » Smoothing
    » Discounting methods
Add-one smoothing

- Add one to all of the counts before normalizing into probabilities
- MLE unigram probabilities
  \[ P(w_i) = \frac{\text{count}(w_i)}{N} \]
- Smoothened unigram probabilities
  \[ P(w_i) = \frac{\text{count}(w_i) + 1}{N + V} \]
- Adjusted counts (unigrams)
  \[ c_i^* = (c_i + 1) \frac{N}{N + V} \]

Add-one smoothing: bigrams

- MLE bigram probabilities
  \[ P(w_n | w_{n-1}) = \frac{\text{count}(w_n | w_{n-1})}{\text{count}(w_{n-1})} \]
- Laplacian bigram probabilities
  \[ P(w_n | w_{n-1}) = \frac{\text{count}(w_n | w_{n-1}) + 1}{\text{count}(w_{n-1}) + V} \]
- Laplacian trigram probabilities

Add-one bigram counts

- Original counts
- New counts
Add-one smoothed bigram probabilities

- **Original**

<table>
<thead>
<tr>
<th>I</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>Chinese</th>
<th>food</th>
<th>lunch</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.0023</td>
<td>.32</td>
<td>.0038</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>want</td>
<td>.0025</td>
<td>.0092</td>
<td>.0031</td>
<td>.26</td>
<td>.0092</td>
<td>0</td>
</tr>
<tr>
<td>to</td>
<td>0</td>
<td>0</td>
<td>.0021</td>
<td>0</td>
<td>.20</td>
<td>.0021</td>
</tr>
<tr>
<td>Chinese</td>
<td>.0094</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.56</td>
<td>.0047</td>
</tr>
<tr>
<td>food</td>
<td>.013</td>
<td>0</td>
<td>.11</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lunch</td>
<td>.0087</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.0022</td>
</tr>
</tbody>
</table>

- **Add-one smoothing**

<table>
<thead>
<tr>
<th>I</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>Chinese</th>
<th>food</th>
<th>lunch</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.0018</td>
<td>.22</td>
<td>.0020</td>
<td>.0028</td>
<td>.0020</td>
<td>.0020</td>
</tr>
<tr>
<td>want</td>
<td>.0014</td>
<td>.0035</td>
<td>.28</td>
<td>.0035</td>
<td>.0025</td>
<td>.0032</td>
</tr>
<tr>
<td>to</td>
<td>.0082</td>
<td>.0021</td>
<td>.0023</td>
<td>.18</td>
<td>.0082</td>
<td>.0021</td>
</tr>
<tr>
<td>eat</td>
<td>.0039</td>
<td>.0039</td>
<td>.012</td>
<td>.0039</td>
<td>.0078</td>
<td>.0012</td>
</tr>
<tr>
<td>Chinese</td>
<td>.016</td>
<td>.0055</td>
<td>.0055</td>
<td>.0055</td>
<td>.0055</td>
<td>.066</td>
</tr>
<tr>
<td>food</td>
<td>.0064</td>
<td>.0032</td>
<td>.0058</td>
<td>.0032</td>
<td>.0032</td>
<td>.0032</td>
</tr>
<tr>
<td>lunch</td>
<td>.0024</td>
<td>.0048</td>
<td>.0048</td>
<td>.0048</td>
<td>.0048</td>
<td>.0048</td>
</tr>
</tbody>
</table>

Too much probability mass is moved

- **Estimated bigram frequencies**
  - AP data, 44 million words
    - Church and Gale (1991)
  - In general, add-one smoothing is a poor method of smoothing
  - Often much worse than other methods in predicting the actual probability for unseen bigrams

<table>
<thead>
<tr>
<th>r = f_{MLE}</th>
<th>f_{emp}</th>
<th>f_{add-1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000027</td>
<td>0.000137</td>
</tr>
<tr>
<td>1</td>
<td>0.448</td>
<td>0.000274</td>
</tr>
<tr>
<td>2</td>
<td>1.25</td>
<td>0.000411</td>
</tr>
<tr>
<td>3</td>
<td>2.24</td>
<td>0.000548</td>
</tr>
<tr>
<td>4</td>
<td>3.23</td>
<td>0.000685</td>
</tr>
<tr>
<td>5</td>
<td>4.21</td>
<td>0.000822</td>
</tr>
<tr>
<td>6</td>
<td>5.23</td>
<td>0.000959</td>
</tr>
<tr>
<td>7</td>
<td>6.21</td>
<td>0.00109</td>
</tr>
<tr>
<td>8</td>
<td>7.21</td>
<td>0.00123</td>
</tr>
<tr>
<td>9</td>
<td>8.26</td>
<td>0.00137</td>
</tr>
</tbody>
</table>

Methodology

- **Cardinal sin: test on the training corpus**
- **Cardinal sin: train on the test corpus**
- **Divide data into training set and test set**
  - Train the statistical parameters on the training set; use them to compute probabilities on the test set
  - Test set: 5%-20% of the total data, but large enough for reliable results
- **Divide training into training and validation set**
  - Validation set might be ~10% of original training set
  - Obtain counts from training set
  - Tune smoothing parameters on the validation set
- **Divide test set into development and final test set**
  - Do all algorithm development by testing on the dev set
  - Save the final test set for the very end…use for reported results

Good-Turing discounting

- Re-estimates the amount of probability mass to assign to N-grams with zero or low counts by looking at the number of N-grams with higher counts.
- Let $N_c$ be the number of N-grams that occur $c$ times.
  - For bigrams, $N_0$ is the number of bigrams of count 0, $N_1$ is the number of bigrams with count 1, etc.
- **Revised counts:**
  $$c^* = (c + 1) \frac{N_{c+1}}{N_c}$$
Good-Turing discounting results

- Works very well in practice
- Usually, the GT discounted estimate $c^*$ is used only for unreliable counts (e.g. < 5)
- As with other discounting methods, it is the norm to treat N-grams with low counts (e.g. counts of 1) as if the count was 0

<table>
<thead>
<tr>
<th>$r$</th>
<th>$f_{\text{MLE}}$</th>
<th>$f_{\text{emp}}$</th>
<th>$f_{\text{add}}$</th>
<th>$f_{\text{GT}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000027</td>
<td>0.000137</td>
<td>0.000027</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.448</td>
<td>0.000274</td>
<td>0.446</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.25</td>
<td>0.000411</td>
<td>1.26</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2.24</td>
<td>0.000548</td>
<td>2.24</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3.23</td>
<td>0.000685</td>
<td>3.24</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4.21</td>
<td>0.000822</td>
<td>4.22</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5.23</td>
<td>0.000959</td>
<td>5.19</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6.21</td>
<td>0.00109</td>
<td>6.21</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7.21</td>
<td>0.00123</td>
<td>7.24</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>8.26</td>
<td>0.00137</td>
<td>8.25</td>
<td></td>
</tr>
</tbody>
</table>

N-gram models

- Unsmoothed n-gram models (review)
- Smoothing
  - Add-one (Laplacian)
  - Good-Turing
- Unknown words
- Evaluating n-gram models
- Combining estimators
  - (Deleted) interpolation
  - Backoff

Unknown words

- Closed vocabulary
  - Vocabulary is known in advance
  - Test set will contain only these words
- Open vocabulary
  - Unknown, out of vocabulary words can occur
  - Add a pseudo-word <UNK>
- Training the model???

Evaluating n-gram models

- Best way: extrinsic evaluation
  - Embed in an application and measure the total performance of the application
  - End-to-end evaluation
- Intrinsic evaluation
  - Measure quality of the model independent of any application
  - Perplexity
    » Intuition: the better model is the one that has a tighter fit to the test data or that better predicts the test data
Perplexity

For a test set $W = w_1 w_2 \ldots w_N$,

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-1/N} = \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

The higher the (estimated) probability of the word sequence, the lower the perplexity.

Must be computed with models that have no knowledge of the test set.

N-gram models

- Unsmoothed n-gram models (review)
- Smoothing
  - Add-one (Laplacian)
  - Good-Turing
- Unknown words
- Evaluating n-gram models
  - (Deleted) interpolation
  - Backoff

Combining estimators

- Smoothing methods
  - Provide the same estimate for all unseen (or rare) n-grams with the same prefix
  - Make use only of the raw frequency of an n-gram
- But there is an additional source of knowledge we can draw on --- the n-gram “hierarchy”
  - If there are no examples of a particular trigram, $w_{n-2}w_{n-1}w_n$, to compute $P(w_n|w_{n-2}w_{n-1})$, we can estimate its probability by using the bigram probability $P(w_n|w_{n-1})$.
  - If there are no examples of the bigram to compute $P(w_n|w_{n-1})$, we can use the unigram probability $P(w_n)$.
- For n-gram models, suitably combining various models of different orders is the secret to success.

Simple linear interpolation

- Construct a linear combination of the multiple probability estimates.
  - Weight each contribution so that the result is another probability function.
  \[
P(w_n | w_{n-2}w_{n-1}) = \lambda_3 P(w_n | w_{n-2}w_{n-1}) + \lambda_2 P(w_n | w_{n-1}) + \lambda_1 P(w_n)
\]
  - Lambda’s sum to 1.
- Also known as (finite) mixture models
- Deleted interpolation
  - Each lambda is a function of the most discriminating context
Backoff (Katz 1987)

- Non-linear method
- The estimate for an n-gram is allowed to back off through progressively shorter histories.
- The most detailed model that can provide sufficiently reliable information about the current context is used.
- Trigram version (high-level):

\[
\hat{P}(w_i | w_{i-2}w_{i-1}) = \begin{cases} 
  P(w_i | w_{i-2}w_{i-1}), & \text{if } C(w_{i-2}w_{i-1}w_i) > 0 \\
  \alpha_1 P(w_i | w_{i-1}), & \text{if } C(w_{i-2}w_{i-1}w_i) = 0 \\
  \quad \text{and } C(w_{i-1}w_i) > 0 \\
  \alpha_2 P(w_i), & \text{otherwise.}
\end{cases}
\]

Final words…

- Problems with backoff?
  - Probability estimates can change suddenly on adding more data when the back-off algorithm selects a different order of n-gram model on which to base the estimate.
  - Works well in practice in combination with smoothing.
- Good option: simple linear interpolation with MLE n-gram estimates plus some allowance for unseen words (e.g. Good-Turing discounting)