LM: Smoothing and Backoff

CMSC 473/673 Spring 2017
Bryan Wilkinson
How to Deal with Zeros

And other Small Numbers too
Language Model Recap

- Language Models are a collection of counts that we use to estimate probability of a [character | word | linguistic unit] with.
- Last class we showed how they can be combined using the chain rule to estimate the probability of a phrase or sentence.
  - We used the Markov Assumption to simplify this calculation
- As we worked through some examples one major issue became clear:
  - Accepting an ngram has a count of 0 means we are saying that combination is impossible
  - We generally want to avoid saying any combination of words is impossible due to the creative nature of language
  - Ignoring it doesn’t feel right either, we lose out on knowledge that a pair (triplet, etc.) is rare
• The reason we have to be so concerned about 0s is because most of our counts will probably be 0.
• This is due to Zipf’s law, which states the rank of a word is proportional to its frequency
  ○ The most common word in a corpus should occur about twice as much as the second most common word
  ○ This means there is a lot of words, or n-grams, that occur 1 or 0 times.
The simplest methods to produce a better model are called smoothing. **Smoothing** refers to modifying probability of N-grams by redistributing the probability distribution so there are no zero probabilities. Jurafsky describes this as “shaving a little bit of probability mass from the higher counts and piling it instead on the zero counts” (SLP2, Pg 98).

Today we will go over 3 ways to do that:
- Laplace Smoothing
- Good-Turing Discounting
- Kesner-Ney Smoothing (Time-Permitting)
Laplace Smoothing

- Laplace Smoothing simply means adding one to every count.
- This is better than zeros, but not effective enough to actually ever be used.
  - We use it to teach because it provides a nice intuitive way to look at smoothing.
- We can look at Laplace Smoothing either in terms of probability or raw counts.

\[
P_L(w_i) = \frac{C(w_i) + 1}{N + V}
\]

\[
C^*(w_i) = (C(w_i) + 1) \frac{N}{N + V}
\]
Laplace Smoothing Example

\[ P_L(w_i) = \frac{C(w_i) + 1}{N + V} \]

\[ C^*(w_i) = \left( C(w_i) + 1 \right) \frac{N}{N + V} \]

- Given the following Unigram counts and probabilities

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>dog</th>
<th>bird</th>
<th>wug</th>
<th>the</th>
<th>ate</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>prob</td>
<td>0.04</td>
<td>0.16</td>
<td>0.16</td>
<td>0</td>
<td>0.36</td>
<td>0.28</td>
</tr>
</tbody>
</table>

- If we add 1 to everything, we would be increasing the number of words in our corpus, which isn’t the goal of smoothing
- In order to “steal” from the other words we need to renormalize using \( V \), the number of words in the vocabulary
Laplace Smoothing Example

\[ P_L(w_i) = \frac{C(w_i) + 1}{N + V} \]

\[ C^*(w_i) = (C(w_i) + 1) \frac{N}{N + V} \]

- The laplace smoothed counts and probabilities are

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>dog</th>
<th>bird</th>
<th>wug</th>
<th>the</th>
<th>ate</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>1.6129</td>
<td>4.0325</td>
<td>4.0325</td>
<td>0.8065</td>
<td>8.0650</td>
<td>6.452</td>
</tr>
<tr>
<td>P</td>
<td>0.0645</td>
<td>0.1613</td>
<td>0.1613</td>
<td>0.0323</td>
<td>0.3226</td>
<td>0.2580</td>
</tr>
</tbody>
</table>

- The counts still add up to 25 and the probabilities still add up to 1
- Notice that the probability of every word except for “the” increased
  - We effectively took some probability mass from “the” and “ate” and redistributed among everyone else
- Now let’s apply this to a bigram so we can see the effect on the probability of a sentence
The probability of a bigram in Laplace smoothing is:

\[ P^* \left( w_n \mid w_{n-1} \right) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V} \]

The adjusted count of a bigram in Laplace smoothing is:

\[ C^* (w_{n-1}w_n) = \frac{(C(w_{n-1}w_n) + 1) \times C(w_{n-1})}{C(w_{n-1}) + V} \]
Laplace Smoothing Bigram Example

\[
P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}
\]

\[
C^*(w_{n-1}w_n) = \frac{(C(w_{n-1}w_n) + 1) \times C(w_{n-1})}{C(w_{n-1}) + V}
\]

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>dog</th>
<th>bird</th>
<th>wug</th>
<th>the</th>
<th>ate</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>dog</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>bird</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>wug</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>the</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ate</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
### Laplace Smoothing Bigram Example

\[
P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}
\]

\[
C^*(w_{n-1}w_n) = \frac{(C(w_{n-1}w_n) + 1) \times C(w_{n-1})}{C(w_{n-1}) + V}
\]

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>dog</th>
<th>bird</th>
<th>wug</th>
<th>the</th>
<th>ate</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>0.14 (0)</td>
<td>0.80 (1)</td>
<td>0.40 (0)</td>
<td>0 (0)</td>
<td>1.2 (1)</td>
<td>1.07 (1)</td>
</tr>
<tr>
<td>dog</td>
<td>0.14 (0)</td>
<td>0.40 (0)</td>
<td>1.60 (3)</td>
<td>0 (0)</td>
<td>2.4 (3)</td>
<td>1.61 (2)</td>
</tr>
<tr>
<td>bird</td>
<td>0.28 (1)</td>
<td>1.20 (2)</td>
<td>0.40 (0)</td>
<td>0 (0)</td>
<td>2.4 (3)</td>
<td>2.15 (3)</td>
</tr>
<tr>
<td>wug</td>
<td>0.14 (0)</td>
<td>0.40 (0)</td>
<td>0.40 (0)</td>
<td>0 (0)</td>
<td>0.6 (0)</td>
<td>0.53 (0)</td>
</tr>
<tr>
<td>the</td>
<td>0.14 (0)</td>
<td>0.40 (0)</td>
<td>0.80 (1)</td>
<td>0 (0)</td>
<td>0.6 (0)</td>
<td>1.07 (1)</td>
</tr>
<tr>
<td>ate</td>
<td>0.14 (0)</td>
<td>0.80 (1)</td>
<td>0.40 (0)</td>
<td>0 (0)</td>
<td>1.8 (2)</td>
<td>0.53 (0)</td>
</tr>
</tbody>
</table>
Calculate the probability of the sentence “the dog ate the wug” using the two language models.

Using the chain rule, the formula for this sentence is

\[ P(\text{dog | the}) \times P(\text{ate | dog}) \times P(\text{the | ate}) \times P(\text{wug | the}) \]
Issues with Laplace Smoothing

\[ P_L(w_i) = \frac{C(w_i) + 1}{N + V} \]

\[ C^*(w_i) = (C(w_i) + 1) \frac{N}{N + V} \]

- Why add 1?
  - We could add another number, but how would we choose it?
  - This exists and is called add-\(\delta\) smoothing

- Does Laplace Smoothing manipulate the distribution in a way that makes sense?
  - In a 370M sample of the ukWaC corpus, 99% of bigrams have a zero count!
  - If we increase all these by one, we are actually moving most of the mass to the zero counts!

- Laplace Smoothing doesn’t use any outside knowledge that might produce a better smoothed distribution
Good-Turing Discounting

- Instead of adding one, what if we use the other counts to estimate the 0s
  - We use the counts of things that occur once to estimate the counts of things we haven’t seen
  - After you do this you should re-estimate the counts for things you have only seen once, and so on.
    - If we don’t do this the new “counts” won’t add up to the same number as before

- Good-Turing counts the number of N-grams that have count c
  - We are finding the frequency of frequency c
  - Let $N_c = \text{the number of N-grams that have a count of } c$

\[
P^* = \frac{C^*}{N} \quad C^* = (C + 1) \frac{N_{c+1}}{N_c}
\]
Good-Turing Discounting Example

- Let’s look at the same unigram counts as before:

<table>
<thead>
<tr>
<th>word</th>
<th>cat</th>
<th>dog</th>
<th>bird</th>
<th>wug</th>
<th>the</th>
<th>ate</th>
</tr>
</thead>
<tbody>
<tr>
<td>freq</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>prob</td>
<td>2.66</td>
<td>5.825</td>
<td>5.825</td>
<td>1</td>
<td>10</td>
<td>8</td>
</tr>
</tbody>
</table>

\[
C^* = (C + 1) \frac{N_{c+1}}{N_c}
\]
Let’s look at the same unigram counts as before:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>dog</th>
<th>bird</th>
<th>wug</th>
<th>the</th>
<th>ate</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>P</td>
<td>0.1065</td>
<td>0.233</td>
<td>0.233</td>
<td>.04</td>
<td>.4</td>
<td>.32</td>
</tr>
</tbody>
</table>

But these don’t add up to 1....
- In practice Good-Turing isn’t usually used on its own.
- For this class, you can just sum up the probabilities and divide by that number so they do add to 1.
- Normally there isn’t so many 0s in the $N_c$ counts for low values of c.
Absolute Discounting

- It turns out that other than 0 and 1, the effect of Good-Turing discounting on counts is close to constant. (From SLP2, AP Newswire Corpus)

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>C*</td>
<td>.0000270</td>
<td>0.446</td>
<td>1.26</td>
<td>2.24</td>
<td>3.24</td>
<td>4.22</td>
</tr>
</tbody>
</table>

- We can save on computation by just assuming it is constant and defining the probability as

\[
P^*(w) = \frac{C(w) - D}{N}
\]

\[
P^*(w | w_{n-1}) = \frac{C(w_{n-1}w) - D}{C(w_{n-1})}
\]
Backoff and Interpolation

- Good-Turing gets better estimates but
  - Still gives every unseen N-gram an equal chance of occurring
- Unless we are looking at unigrams we have additional information that we can use.
  - We might not know the count of \((\text{ate}, \text{wug})\), but we probably know the count of just \text{ate} and just \text{wug}
- Interpolation means we estimate an ngram probability by a weighted sum of the ngram probability \(\times\) (n-1)gram probability \(\times\) …. \(\times\) unigram probability
  - We do this for all ngram probabilities, not just those that are 0
- Backoff is a technique used only when the counts of a particular ngram are 0
  - We just use the (n-1) gram count instead, or if it is 0 continue until we get to the unigram count
Interpolation

- The formula for interpolation of a bigram is

\[ P^*(w_n|w_{n-1}) = \lambda_1 P(w_n|w_{n-1}) + \lambda_2 P(w_n) \]

Where the lambdas add to 1
Backoff

- When using backoff, it is important to have already calculated the discounted probabilities
  - This is important to make sure everything still adds up to 1
  - We decide whether or not to use backoff based on the original counts
- If the original count of an N-gram is 0, we let the probability of that N-gram be the undiscounted probability of the (N-1)-gram, weighted by a function of the (N-1)-gram

\[
P_k(w_n|w_{n-1}) = \begin{cases} 
P^*(w_n|w_{n-1}^+) & C(w_{n-1}w_n) > 0 \\
\alpha(w_{n-1})P^*(w_n) & C(w_{n-1}w_n) = 0 \end{cases}
\]

- For simple models, Backoff > Interpolation, but......
Kneser-Ney discounting

- Originally a backoff-model
- If the count is not zero, use the absolute discounted probability
- If it is zero, backoff to an estimate based on the previous word
- Uses the intuition that the more ngrams a word appears in the more likely it is to be in an ngram with count 0 too.
  - Instead of $P(\text{the wug}) = P(\text{mauve wug})$, we assume $P(\text{the wug}) \gg P(\text{mauve wug})$ because \textit{the} occurs in many more N-grams
  - We calculate this \textbf{continuation probability function} as

$$P_C(w) = \frac{|| \{ w_{n-1} : C(w_{n-1}w) > 0 \} ||}{|| \{ (w_{n-1}, w) : C(w_{n-1}w) > 0 \} ||}$$
It turns out that interpolation works better when we use a method like Kneser-Ney.

Rather than interpolate between for example, trigrams, bigrams and unigrams, we interpolate between the N-gram probability and the continuation probability.

For bigrams this becomes:

$$P_{KN}(w|w_{n-1}) = \frac{C(w_{n-1}w) - D}{C(w_{n-1})} + \beta(w_{n-1})P_C(w_n)$$

$$\beta(w_{n-1}) = \frac{D}{C(w_{n-1})} \left\| \{w : C(w_{n-1}w) > 0\} \right\|$$