Smoothing

L645 / B659
Dept. of Linguistics, Indiana University
Fall 2015
Smoothing

Definitions

- N-gram matrix for any given training corpus is **sparse**
  - i.e., not all n-grams will be present
  - MLE produces bad estimates when the counts are small

- **Smoothing** = re-evaluating zero & small probabilities, assigning very small probabilities for zero n-grams
  - If non-occurring n-grams receive small probabilities, the probability mass needs to be redistributed!
  - Smoothing also sometimes called **discounting**
Types vs. Tokens

- We’ve seen this before:
  - **Token**: single item
  - **Type**: abstract class of items
- Example: words in text
  - Token: each word
  - Type: each **different** word, i.e., wordform
- **sentence**: the man with the hat
  
  Tokens: the, man, with, the, hat  # of tokens = 5
  Types: the, man, with, hat         # of types = 4
Basic Techniques

An overview of what we’ll look at:

▶ Add-One Smoothing (& variations)
  ▶ Laplace’s, Lidstone’s, & Jeffreys-Perks laws
▶ Deleted estimation: validate estimates from one part of corpus with another part
▶ Witten-Bell smoothing: use probabilities of seeing events for the first time
▶ Good-Turing estimation: use ratios between \( n \)-grams occurring \( n + 1 \) and \( n \) times

Following Manning and Schütze, we’ll use \( n \)-gram language modeling as our example
Basics of $n$-grams

$n$-grams are used to model language, capturing some degree of grammatical properties

- The probability of a word based on its history:

  \[ P(w_n|w_1...w_{n-1}) \]

- $n$-gram probabilities are estimated as follows

  \[ P(w_n|w_1...w_{n-1}) = \frac{P(w_1...w_n)}{P(w_1...w_{n-1})} \]

- To avoid data sparsity issues, bigrams and trigrams are commonly used

- We can use maximum likelihood estimation (MLE) to obtain basic probabilities:

  \[ P(w_n|w_1...w_{n-1}) = \frac{C(w_1...w_n)}{C(w_1...w_{n-1})} \]

But MLE probabilities do nothing to handle unseen data
Add-One Smoothing

**Idea:** pretend that non-existent bigrams are there once

- To make the model more just: assume that for each bigram we also add one to the count

... This turns out not to be a very good estimator
Add-One Smoothing

Laplace’s Law

- **Unigram probabilities:**
  
  \[ N = \text{number of tokens} \]
  
  \[ C(x) = \text{frequency of } x \]
  
  \[ V = \text{vocabulary size; number of types} \]

- **Standard probability for word** \( w_x \): \( P(w_x) = \frac{C(w_x)}{N} \)

- **Adjusted count:** \( C^*(w_x) = (C(w_x) + 1) \frac{N}{N+V} \)

  \[ \frac{N}{N+V} : \text{normalizing factor; } N + V : \text{new “size” of text} \]

- **\( p^*(w_x) \): estimated probability**
  
  \[ \text{probability: } p^*(w_x) = \frac{(C(w_x) + 1) \frac{N}{N+V}}{N} = \frac{c(w_x) + 1}{N+V} \]
Test Corpus: Windows Haiku Corpus

- Corpus: 16 haikus, 253 tokens, 165 words
  - Haiku: Japanese poem with 17 syllables (mora); 5 in first line, 7 in second, 5 in third
- Windows NT crash’d.
  I am the Blue Screen of Death.
  No-one hears your screams.
- Yesterday it work’d.
  Today it is not working.
  Windows is like that.
- Three things are certain:
  Death, taxes and lost data.
  Guess which has occurred.
## Test Corpus: Add-One Smoothing

<table>
<thead>
<tr>
<th>word</th>
<th>freq.</th>
<th>unsmoothed: ( \frac{C(w)}{N} )</th>
<th>add-one: ( \frac{C(w)+1}{N+V} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>35</td>
<td>0.1383</td>
<td>0.0860</td>
</tr>
<tr>
<td>,</td>
<td>8</td>
<td>0.0316</td>
<td>0.0215</td>
</tr>
<tr>
<td>the</td>
<td>7</td>
<td>0.0277</td>
<td>0.0191</td>
</tr>
<tr>
<td>The</td>
<td>4</td>
<td>0.0158</td>
<td>0.0119</td>
</tr>
<tr>
<td>that</td>
<td>3</td>
<td>0.0119</td>
<td>0.0095</td>
</tr>
<tr>
<td>on</td>
<td>2</td>
<td>0.0079</td>
<td>0.0072</td>
</tr>
<tr>
<td>We</td>
<td>1</td>
<td>0.0040</td>
<td>0.0048</td>
</tr>
<tr>
<td>operator</td>
<td>0</td>
<td>0.0000</td>
<td>0.0024</td>
</tr>
</tbody>
</table>
Add-One Smoothing: Bigrams

- \( P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \)

- \( p^*(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V} \)
## Bigrams Example

<table>
<thead>
<tr>
<th>bigram</th>
<th>freq.</th>
<th>freq. $w_{n-1}$</th>
<th><strong>unsmoothed:</strong> $\frac{C(w_{n-1}w_n)}{C(w_{n-1})}$</th>
<th><strong>add-one:</strong> $\frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>. END</td>
<td>35</td>
<td>35</td>
<td>1.0000</td>
<td>0.1800</td>
</tr>
<tr>
<td>START The</td>
<td>3</td>
<td>35</td>
<td>0.0857</td>
<td>0.0200</td>
</tr>
<tr>
<td>START You</td>
<td>2</td>
<td>35</td>
<td>0.0571</td>
<td>0.0150</td>
</tr>
<tr>
<td>is not</td>
<td>2</td>
<td>7</td>
<td>0.2857</td>
<td>0.0174</td>
</tr>
<tr>
<td>Your ire</td>
<td>1</td>
<td>2</td>
<td>0.5000</td>
<td>0.0120</td>
</tr>
<tr>
<td>You bring</td>
<td>1</td>
<td>3</td>
<td>0.3333</td>
<td>0.0119</td>
</tr>
<tr>
<td>not found</td>
<td>1</td>
<td>4</td>
<td>0.2500</td>
<td>0.0118</td>
</tr>
<tr>
<td>is the</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0.0058</td>
</tr>
<tr>
<td>This system</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0.0060</td>
</tr>
</tbody>
</table>
Lidstone’s & Jeffreys-Perks Laws

Because Laplace’s law overestimates zero events, variations were created:

▶ Lidstone’s law: instead of adding one, add some smaller value $\lambda$

$$P(w_1...w_n) = \frac{C(w_1...w_n) + \lambda}{N + V\lambda}$$

▶ Jeffreys-Perks law: set $\lambda$ to be $\frac{1}{2}$ (the expectation of maximized MLE):

$$P(w_1...w_n) = \frac{C(w_1...w_n) + \frac{1}{2}}{N + \frac{1}{2}V}$$

Problems: How do we guess $\lambda$? And still not good for low frequency $n$-grams . . .
Towards Deleted Estimation

Held-Out Estimation

To get an idea as to whether a smoothing technique is effective, we can use held-out estimation.

▶ Split the data into training data and held-out data
▶ Use the held-out data to see how good the training estimates are

Using bigrams as an example:

▶ $N_r$ bigrams with frequency $r$ in the training data
▶ $T_r$: how often all these bigrams together occur in the held-out data
▶ Average frequency in held-out data is thus $\frac{T_r}{N_r}$
Held-Out Estimation

Since $N$ is the number of training instances, the probability of one of these $n$-grams is $\frac{T_r}{N_rN}$

This re-estimate can provide one of two different things:

- A reality check on smoothing technique being used
- A better estimate to be used on the testing data
  - It is critical that the testing data be disjoint from both held-out & training data
Deleted Estimation

Held-Out Estimation keeps the held-out data separate from the training data

- But what if we split the training data in half?
  1. Train on one half and validate on the other
  2. Then, switch the training and validation portions
- With both of these estimates, we average them to obtain more reliable estimates

\[ p_{del}(w_1 w_2) = \frac{T_r^1 + T_r^2}{N(N_r^1 + N_r^2)} \]
Deleted Estimation turns out to be quite good ... but not for low frequency $n$-grams

- Overestimates unseen objects & underestimates one-time objects
  - The number of unseen objects is not linear, but deleted estimation assumes it is
    - As the size of the data increases, there are generally less unseen $n$-grams
    - Smaller training sets lead to more unseen events in the held-out data
  - Deleted estimation assumes the probability of an object seen $r$ times in $\frac{N}{2}$ data is double one seen $r$ times in $N$ data
Witten-Bell Discounting

Problems with Add-One Smoothing:

▶ Add-one smoothing leads to sharp changes in probabilities
▶ Too much probability mass goes to unseen events

Intuition between Witten-Bell: unseen events are ones that have not happened yet
▶ Probability of this event can be modeled by probability of seeing it for the first time
First Time Probability

How do we estimate probability of an N-gram occurring for the first time?

- Count number of times of seeing an N-gram for the first time in training corpus
- Think of corpus as series of events: one event for each token and one event for each new type
- e.g. unigrams:
  corpus: a man with a hat
  event: a **new** man **new** with **new** a hat...

- Number of events: $N + T$
Witten-Bell Probabilities

- **Total** probability mass for unseen events:
  \[ \sum_{x: C(w_x) = 0} p^*(w_x) = \frac{T}{N + T} \]

- Probability for one unseen unigram:
  \[ p^*(w_x) = \frac{T}{Z(N+T)} \]
  - Divide total prob. mass for all unseen events by number of all unseen unigrams: \( Z = \sum_{x: C(w_x) = 0} 1 \)

- Discount total probability mass for unseen events from other events:
  \[ p^*(w_x) = \frac{C(w_x)}{N + T} \] for \( C(w_x) > 0 \)

- Alternatively: **smoothed counts**:
  \[
  C^*(w_x) = \begin{cases} 
  \frac{T}{Z(N+T)} \frac{N}{N + T} & \text{if } C(w_x) = 0 \\
  C(w_x) \frac{N}{N + T} & \text{if } C(w_x) > 0 
  \end{cases}
  \]
Witten-Bell Smoothed Bigrams

Type counts are conditioned on previous word: use probability of bigram **starting with previous word**

- \( T(w_x) = \) number of bigrams starting with \( w_x \)

Zero-count events:

- Total prob. mass:
  \[
  \sum_{i: C(w_{i-1}w_i) = 0} p^*(w_i \mid w_{i-1}) = \frac{T(w_{i-1})}{N + T(w_{i-1})}
  \]

- \( p^*(w_i \mid w_{i-1}) = \frac{T(w_{i-1})}{Z(w_{i-1})(N + T(w_{i-1}))} \) if \( C(w_{i-1}w_i) = 0 \)
  
  - \( Z(w_{i-1}) = \sum_{i: C(w_{i-1}w_i) = 0} 1 \)

Non-zero-count events:

- \( p^*(w_i \mid w_{i-1}) = \frac{C(w_{i-1}w_i)}{N + T(w_{i-1})} \) if \( C(w_{i-1}w_i) > 0 \)
  
  - \( N = C(w_{i-1}) \)
### $T(w)$ And $Z(w)$ from Haikus

$Z(w)$: number of unseen bigrams starting with $w$

$V$: complete number of bigrams (types)

$Z(w) = V - T(w) = 165 - T(w)$

<table>
<thead>
<tr>
<th>word</th>
<th>$T(w)$</th>
<th>$Z(w)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>1</td>
<td>164</td>
</tr>
<tr>
<td>START</td>
<td>13</td>
<td>152</td>
</tr>
<tr>
<td>is</td>
<td>6</td>
<td>159</td>
</tr>
<tr>
<td>Your</td>
<td>2</td>
<td>163</td>
</tr>
<tr>
<td>You</td>
<td>2</td>
<td>163</td>
</tr>
<tr>
<td>not</td>
<td>4</td>
<td>161</td>
</tr>
<tr>
<td>This</td>
<td>2</td>
<td>163</td>
</tr>
<tr>
<td>bigram</td>
<td>unsmoothed</td>
<td>add-one</td>
</tr>
<tr>
<td>----------------</td>
<td>------------</td>
<td>---------</td>
</tr>
<tr>
<td>. END</td>
<td>1.0000</td>
<td>0.1800</td>
</tr>
<tr>
<td>START The</td>
<td>0.0857</td>
<td>0.0200</td>
</tr>
<tr>
<td>START You</td>
<td>0.0571</td>
<td>0.0150</td>
</tr>
<tr>
<td>is not</td>
<td>0.2857</td>
<td>0.0174</td>
</tr>
<tr>
<td>Your ire</td>
<td>0.5000</td>
<td>0.0120</td>
</tr>
<tr>
<td>You bring</td>
<td>0.3333</td>
<td>0.0119</td>
</tr>
<tr>
<td>not found</td>
<td>0.2500</td>
<td>0.0118</td>
</tr>
<tr>
<td>is the</td>
<td>0</td>
<td>0.0058</td>
</tr>
<tr>
<td>This system</td>
<td>0</td>
<td>0.0060</td>
</tr>
</tbody>
</table>
Good-Turing Smoothing

**Idea:** re-estimate probability *mass* assigned to *n*-grams with zero counts
- Done by looking at probability mass of all *n*-grams with count 1
- Based on assumption of binomial distribution

Idea broken down:
- Create classes $N_c$ of *n*-grams which occur *c* times
- Size of class $N_c$ is the frequency of frequency *c*

This works well for *n*-grams
Good-Turing Smoothing (2)

- Smoothed count \( c^* = (c + 1) \frac{N_{c+1}}{N_c} \)
  - \( N_c = \sum_{b: c(b) = c} 1 \)
- Smoothed count for unseen events: \( c^* = \frac{N_1}{N_0} \)

Haiku counts:

<table>
<thead>
<tr>
<th>c</th>
<th>( N_c )</th>
<th>( c^* )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>26980</td>
<td>0.0087</td>
</tr>
<tr>
<td>1</td>
<td>236</td>
<td>0.0593</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>0.4286</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.0000</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0.0000</td>
</tr>
</tbody>
</table>
Good-Turing Smoothing (3)

- **Problem:** for highest count \( c \), \( N_{c+1} = 0 \)!!!
  - i.e. \( c^* = (c + 1) \frac{N_{c+1}}{N_c} = (c + 1) \frac{0}{N_c} = 0 \)
- **Solution:** discount only for small counts \( c \leq k \) (e.g. \( k = 5 \))
  - \( c^* = c \) for \( c > k \)
- **New discounting:**
  \[
  c^* = \frac{(c+1) \frac{N_{c+1}}{N_c} - c \frac{(k+1)N_{k+1}}{N_1}}{1 - \frac{(k+1)N_{k+1}}{N_1}} \quad \text{for } 1 \leq c \leq k
  \]
Haiku Bigrams

- G-T = \frac{c^*(w_{n-1}w_n)}{C(w_{n-1})}
- k = 3

<table>
<thead>
<tr>
<th>bigram</th>
<th>count</th>
<th>orig.</th>
<th>add-1</th>
<th>W-B</th>
<th>G-T</th>
</tr>
</thead>
<tbody>
<tr>
<td>. END</td>
<td>35</td>
<td>1.0000</td>
<td>0.1800</td>
<td>0.9722</td>
<td>1.0000</td>
</tr>
<tr>
<td>START The</td>
<td>3</td>
<td>0.0857</td>
<td>0.0200</td>
<td>0.0625</td>
<td>0.0857</td>
</tr>
<tr>
<td>START You</td>
<td>2</td>
<td>0.0571</td>
<td>0.0150</td>
<td>0.0417</td>
<td>0.0122</td>
</tr>
<tr>
<td>is not</td>
<td>2</td>
<td>0.2857</td>
<td>0.0174</td>
<td>0.1538</td>
<td>0.1837</td>
</tr>
<tr>
<td>Your ire</td>
<td>1</td>
<td>0.5000</td>
<td>0.0120</td>
<td>0.2500</td>
<td>0.0297</td>
</tr>
<tr>
<td>You bring</td>
<td>1</td>
<td>0.3333</td>
<td>0.0119</td>
<td>0.2000</td>
<td>0.0198</td>
</tr>
<tr>
<td>not found</td>
<td>1</td>
<td>0.2500</td>
<td>0.0118</td>
<td>0.1250</td>
<td>0.0148</td>
</tr>
<tr>
<td>is the</td>
<td>0</td>
<td>0</td>
<td>0.0058</td>
<td>0.0029</td>
<td>0.0012</td>
</tr>
<tr>
<td>This system</td>
<td>0</td>
<td>0</td>
<td>0.0060</td>
<td>0.0031</td>
<td>0.0044</td>
</tr>
</tbody>
</table>
**Backoff**

**Idea:** go back to “smaller” N-grams

- If no trigram is found, use bigram probability; if no bigram is found, use unigram

This can be used instead of smoothing

- Need to weight contribution of specific \( n \)-gram:

\[
P^*(w_i \mid w_{i-2} w_{i-1}) = \begin{cases} 
P(w_i \mid w_{i-2} w_{i-1}) & \text{if } C(w_{i-2} w_{i-1} w_i) > 0 \\
\alpha_1 P(w_i \mid w_{i-1}) & \text{if } C(w_{i-2} w_{i-1} w_i) = 0 \text{ and } C(w_{i-1} w_i) > 0 \\
\alpha_2 P(w_i) & \text{otherwise} 
\end{cases}
\]
Linear Interpolation

Simple linear interpolation

Simple linear interpolation involves mixing different pieces of information to derive a probability

- Called deleted interpolation with subset relations (e.g., bigrams and unigrams are subsets of trigrams)

\[ \hat{P}(w_i|w_{i-2}w_{i-1}) = \lambda_1 P(w_i|w_{i-2}w_{i-1}) + \lambda_2 P(w_i|w_{i-1}) + \lambda_3 P(w_i) \]

- \( \sum \lambda_i = 1 \)
- \( 0 \leq \lambda_i \leq 1 \)

Every trigram probability is a linear combination of the focus word’s trigram, bigram, and unigram.

- Use EM algorithm on held-out data to calculate \( \lambda \) values
More generally, we can condition the word on its history and each $\lambda$ can be based on the history, too

\begin{equation}
P(w|h) = \sum_i \lambda_i(h)P_i(w|h)
\end{equation}

\[= \lambda_1(h)P_1(w|h) + \lambda_2(h)P_2(w|h) + \lambda_3(h)P_3(w|h)\]

▪ $P_1$ may focus on the trigram history, while $P_2$ uses the bigram, and so forth.

▪ Instead of having one $\lambda_1$ for all trigrams, we have individualized it for each unique trigram
  ▪ Every trigram potentially behaves differently
  ▪ But sparse data is an issue
Equivalence bins

To overcome the sparse data problem, \(\lambda\)'s are calculated by putting them into equivalence bins

- One method (Chen and Goodman 1996) bases the bins on the number of different words which an \(n-1\)-gram has following it

\[
\frac{C(w_1...w_{i-1})}{|w_i: C(w_1...w_i) > 0|}
\]

- \(w_i : C(w_1...w_i) > 0\): the set of \(w_i\) such that the trigram exists

- *great deal* occurs 178 times, with 36 different words after it: average count = 4.94

- *of that* occurs 178 times, with 115 different words after it: 1.55

- These histories will thus prompt different \(\lambda\) values