Lecture 4:
Smoothing

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center
Last lecture’s key concepts

N-gram language models: unigram, bigram, trigram…
Independence assumptions
Relative frequency estimation
Unseen events
Zipf’s law
Start and End symbols $<s>\ldots <\backslash s>$

Why do we need a start-of-sentence symbol?
This is just a mathematical convenience, since it allows us to write e.g. $P(w_1 | <s>)$ for the probability of the first word in analogy to $P(w_{i+1} | w_i)$ for any other word.

Why do we need an end-of-sentence symbol?
This is necessary if we want to compare the probability of strings of different lengths (and actually define a probability distribution over $V^*$).
We include $<\backslash s>$ in the vocabulary $V$, require that each string ends in $<\backslash s>$ and that $<\backslash s>$ can only appear at the end of sentences, and estimate $P(w_{i+1} = <\backslash s> | w_i)$. 
Zipf’s law: the long tail

How many words occur once, twice, 100 times, 1000 times?

the $r$-th most common word $w_r$ has $P(w_r) \propto 1/r$

A few words are very frequent
Most words are very rare

In natural language:
A small number of events (e.g. words) occur with high frequency
A large number of events occur with very low frequency
Today’s lecture

How can we design language models* that can deal with previously unseen events?

*actually, probabilistic models in general
Parameter estimation (training)

Parameters: the actual probabilities
\[ P(w_i = 'the' \mid w_{i-1} = 'on') = ??? \]

We need (a large amount of) text as training data to estimate the parameters of a language model.

The most basic estimation technique: 
relative frequency estimation (= counts)
\[ P(w_i = 'the' \mid w_{i-1} = 'on') = \frac{C('on the')}{C('on')} \]
This assigns all probability mass to events in the training corpus.
Also called Maximum Likelihood Estimation (MLE)
How do we evaluate models?

Define an evaluation metric (scoring function).
We will want to measure how similar the predictions of the model are to real text.

Train the model on a ‘seen’ training set
Perhaps: tune some parameters based on held-out data (disjoint from the training data, meant to emulate unseen data)

Test the model on an unseen test set
(usually from the same source (e.g. WSJ) as the training data)
Test data must be disjoint from training and held-out data
Compare models by their scores (more on this next week).
Testing: unseen events will occur

Recall the Shakespeare example:

**Only 30,000 word types occurred.**
Any word that does not occur in the training data has zero probability!

**Only 0.04% of all possible bigrams occurred.**
Any bigram that does not occur in the training data has zero probability!
Dealing with unseen events

Relative frequency estimation assigns all probability mass to events in the training corpus.

But we need to reserve some probability mass to events that don’t occur in the training data.
   Unseen events = new words, new bigrams

Important questions:
   What possible events are there?
   How much probability mass should they get?
What unseen events may occur?

Simple distributions:

\[ P(X = x) \]

(e.g. unigram models)

Possibility:
The outcome \( x \) has not occurred during training (i.e. is unknown):

We need to reserve mass in \( P(X) \) for \( x \)

What outcomes \( x \) are possible?
How much mass should they get?
What unseen events may occur?

Simple _conditional_ distributions:

\[ P(X = x \mid Y = y) \]

(e.g. bigram models)

The outcome \( x \) has been seen, but not in the context of \( Y = y \):

We need to reserve mass in \( P(X \mid Y=y) \) for \( X = x \)

The conditioning variable \( y \) has not been seen:

We have no \( P(X \mid Y=y) \) distribution.

We need to drop the conditioning context \( Y=y \) and use \( P(X) \) instead.
What unseen events may occur?

Complex conditional distributions
\[ P( X = x \mid Y = y, Z = z) \]
(e.g. trigram models)

The outcome \( X = x \) was seen, but not in the context of \( (Y=y, Z=z) \):
- We need to reserve mass in \( P( X \mid Y=y, Z=z) \)

The joint conditioning event \( (Y=y, Z=z) \) has not been seen:
- We have no \( P( X \mid Y=y, Z=z) \) distribution.
- We need to drop \( z \) and use \( P( X \mid Y=y) \) instead.
Examples

Training data: The wolf is an endangered species
Test data: The wallaby is endangered

<table>
<thead>
<tr>
<th>Unigram</th>
<th>Bigram</th>
<th>Trigram</th>
</tr>
</thead>
<tbody>
<tr>
<td>P(the)</td>
<td>P(the</td>
<td>&lt;s&gt;)</td>
</tr>
<tr>
<td>× P(wallaby)</td>
<td>× P( wallaby</td>
<td>the)</td>
</tr>
<tr>
<td>× P(is)</td>
<td>× P(is</td>
<td>wallaby)</td>
</tr>
<tr>
<td>× P(endedangered)</td>
<td>× P(endedangered</td>
<td>is)</td>
</tr>
</tbody>
</table>

- **Case 1:** P(wallaby), P(wallaby | the), P( wallaby | the, <s>):
  What is the probability of an unknown word (in any context)?
- **Case 2:** P(endedangered | is)
  What is the probability of a known word in a known context, if that word hasn’t been seen in that context?
- **Case 3:** P(is | wallaby) P(is | wallaby, the) P(endedangered | is, wallaby):
  What is the probability of a known word in an unseen context?
Smoothing: Reserving mass in $P(X)$ for unseen events
Dealing with unknown words: The simple solution

Training:
- Assume a fixed vocabulary
  (e.g. all words that occur at least twice (or n times) in the corpus)
- Replace all other words by a token <UNK>
- Estimate the model on this corpus.

Testing:
- Replace all unknown words by <UNK>
- Run the model.

This requires a large training corpus to work well.
Dealing with unknown events

Use a different estimation technique:
- Add-1 (Laplace) Smoothing
- Good-Turing Discounting

Idea: Replace MLE estimate \( P(w) = \frac{C(w)}{N} \)

Combine a complex model with a simpler model:
- Linear Interpolation
- Modified Kneser-Ney smoothing

Idea: use bigram probabilities of \( w_i \) \( P(w_i | w_{i-1}) \) to calculate trigram probabilities of \( w_i \) \( P(w_i | w_{i-n} \ldots w_{i-1}) \)
Add-1 (Laplace) smoothing

Assume every (seen or unseen) event occurred once more than it did in the training data.

**Example: unigram probabilities**
Estimated from a corpus with $N$ tokens and a vocabulary (number of word types) of size $V$.

\[
P(w_i) = \frac{C(w_i)}{\sum_j C(w_j)} = \frac{C(w_i)}{N}
\]

**MLE**

\[
P(w_i) = \frac{C(w_i)+1}{\sum_j (C(w_j)+1)} = \frac{C(w_i)+1}{N+V}
\]

**Add One**
## Bigram counts

### Original:

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>5</td>
<td>827</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>want</td>
<td>2</td>
<td>0</td>
<td>608</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>to</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>686</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>211</td>
</tr>
<tr>
<td>eat</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>16</td>
<td>2</td>
<td>42</td>
<td>0</td>
</tr>
<tr>
<td>chinese</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>82</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>food</td>
<td>15</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lunch</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>spend</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Smoothed:

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>6</td>
<td>828</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>want</td>
<td>3</td>
<td>1</td>
<td>609</td>
<td>2</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>to</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>687</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>212</td>
</tr>
<tr>
<td>eat</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>17</td>
<td>3</td>
<td>43</td>
<td>1</td>
</tr>
<tr>
<td>chinese</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>83</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>food</td>
<td>16</td>
<td>1</td>
<td>16</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>lunch</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>spend</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
### Bigram probabilities

#### Original:

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0.002</td>
<td>0.33</td>
<td>0</td>
<td>0.0036</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00079</td>
</tr>
<tr>
<td>want</td>
<td>0.0022</td>
<td>0</td>
<td>0.66</td>
<td>0.0011</td>
<td>0.0065</td>
<td>0.0065</td>
<td>0.0054</td>
<td>0.0011</td>
</tr>
<tr>
<td>to</td>
<td>0.00083</td>
<td>0</td>
<td>0.0017</td>
<td>0.28</td>
<td>0.00083</td>
<td>0.0027</td>
<td>0.0025</td>
<td>0.087</td>
</tr>
<tr>
<td>eat</td>
<td>0</td>
<td>0</td>
<td>0.0027</td>
<td>0</td>
<td>0.021</td>
<td>0.52</td>
<td>0.056</td>
<td>0</td>
</tr>
<tr>
<td>chinese</td>
<td>0.0063</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0092</td>
<td>0.0037</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>food</td>
<td>0.014</td>
<td>0</td>
<td>0.014</td>
<td>0</td>
<td>0</td>
<td>0.0029</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lunch</td>
<td>0.0059</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>spend</td>
<td>0.0036</td>
<td>0</td>
<td>0.0036</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Smoothed:

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0.0015</td>
<td>0.21</td>
<td>0.00025</td>
<td>0.0025</td>
<td>0.00025</td>
<td>0.00025</td>
<td>0.00025</td>
<td>0.00075</td>
</tr>
<tr>
<td>want</td>
<td>0.0013</td>
<td>0.00042</td>
<td>0.26</td>
<td>0.00084</td>
<td>0.0029</td>
<td>0.0029</td>
<td>0.0025</td>
<td>0.00084</td>
</tr>
<tr>
<td>to</td>
<td>0.00078</td>
<td>0.00026</td>
<td>0.0013</td>
<td>0.18</td>
<td>0.00078</td>
<td>0.00026</td>
<td>0.0018</td>
<td>0.055</td>
</tr>
<tr>
<td>eat</td>
<td>0.00046</td>
<td>0.00046</td>
<td>0.0014</td>
<td>0.00046</td>
<td>0.0078</td>
<td>0.0014</td>
<td>0.02</td>
<td>0.00046</td>
</tr>
<tr>
<td>chinese</td>
<td>0.0012</td>
<td>0.00062</td>
<td>0.0062</td>
<td>0.00062</td>
<td>0.00062</td>
<td>0.052</td>
<td>0.0012</td>
<td>0.00062</td>
</tr>
<tr>
<td>food</td>
<td>0.0063</td>
<td>0.00039</td>
<td>0.0063</td>
<td>0.00039</td>
<td>0.00079</td>
<td>0.002</td>
<td>0.00039</td>
<td>0.00039</td>
</tr>
<tr>
<td>lunch</td>
<td>0.0017</td>
<td>0.00056</td>
<td>0.00056</td>
<td>0.00056</td>
<td>0.00056</td>
<td>0.0011</td>
<td>0.00056</td>
<td>0.00056</td>
</tr>
<tr>
<td>spend</td>
<td>0.0012</td>
<td>0.00058</td>
<td>0.0012</td>
<td>0.00058</td>
<td>0.00058</td>
<td>0.00058</td>
<td>0.00058</td>
<td>0.00058</td>
</tr>
</tbody>
</table>

### Problem:

Add-one moves too much probability mass from seen to unseen events!
Reconstituting the counts

We can “reconstitute” pseudo-counts $c^*$ for our training set of size $N$ from our estimate:

Unigrams:

$$c_i^* = P(w_i) \cdot N$$

$$= \frac{C(w_i) + 1}{N + V} \cdot N$$

$$= (C(w_i) + 1) \cdot \frac{N}{N + V}$$

Bigrams:

$$c^*_{(w_i|w_{i-1})} = P(w_i|w_{i-1}) \cdot C(w_{i-1})$$

$$= \frac{C(w_{i-1}w_i) + 1}{C(w_{i-1}) + V} \cdot C(w_{i-1})$$

$P(w_{i-1}w_i)$: probability of bigram “$w_{i-1}w_i$”.

$C(w_{i-1})$: frequency of $w_{i-1}$ (in training data)

$P(w_i)$: probability that the next word is $w_i$.

$N$: number of word tokens we generate

$V$: size of vocabulary

Plug in the model definition of $P(w_i)$

Rearrange (to see dependence on $N$ and $V$)

Plug in the model definition of $P(w_i|w_{i-1})$
## Reconstituted Bigram counts

### Original:

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>5</td>
<td>827</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>want</td>
<td>2</td>
<td>0</td>
<td>608</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>to</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>668</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>211</td>
</tr>
<tr>
<td>eat</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>16</td>
<td>2</td>
<td>42</td>
<td>0</td>
</tr>
<tr>
<td>chinese</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>82</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>food</td>
<td>15</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lunch</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>spend</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Reconstituted:

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>3.8</td>
<td>527</td>
<td>0.64</td>
<td>6.4</td>
<td>0.64</td>
<td>0.64</td>
<td>0.64</td>
<td>1.9</td>
</tr>
<tr>
<td>want</td>
<td>1.2</td>
<td>0.39</td>
<td>238</td>
<td>0.78</td>
<td>2.7</td>
<td>2.7</td>
<td>2.3</td>
<td>0.78</td>
</tr>
<tr>
<td>to</td>
<td>1.9</td>
<td>0.63</td>
<td>3.1</td>
<td>430</td>
<td>1.9</td>
<td>0.63</td>
<td>4.4</td>
<td>133</td>
</tr>
<tr>
<td>eat</td>
<td>0.34</td>
<td>0.34</td>
<td>1</td>
<td>0.34</td>
<td>5.8</td>
<td>1</td>
<td>0.2</td>
<td>0.098</td>
</tr>
<tr>
<td>chinese</td>
<td>0.2</td>
<td>0.098</td>
<td>0.098</td>
<td>0.098</td>
<td>0.098</td>
<td>8.2</td>
<td>0.38</td>
<td>0.19</td>
</tr>
<tr>
<td>food</td>
<td>6.9</td>
<td>0.43</td>
<td>6.9</td>
<td>0.43</td>
<td>0.86</td>
<td>2.2</td>
<td>0.43</td>
<td>0.43</td>
</tr>
<tr>
<td>lunch</td>
<td>0.57</td>
<td>0.19</td>
<td>0.19</td>
<td>0.19</td>
<td>0.19</td>
<td>0.38</td>
<td>0.19</td>
<td>0.19</td>
</tr>
<tr>
<td>spend</td>
<td>0.32</td>
<td>0.16</td>
<td>0.32</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
</tr>
</tbody>
</table>
Summary: Add-One smoothing

Advantage:
Very simple to implement

Disadvantage:
Takes away too much probability mass from seen events.
Assigns too much total probability mass to unseen events.

The Shakespeare example
(V = 30,000 word types; ‘the’ occurs 25,545 times)
Bigram probabilities for ‘the …’:

\[
P(w_i|w_{i-1} = \text{the}) = \frac{C(\text{the } w_i) + 1}{25,545 + 30,000}\]
Add-K smoothing

Variant of Add-One smoothing:
For any $k > 0$ (typically, $k < 1$)

\[
\text{Add K} \quad P(w_i) = \frac{C(w_i) + k}{N + kV}
\]

This is still too simplistic to work well.
Good-Turing smoothing

Basic idea: Use total frequency of events that occur only once to estimate how much mass to shift to unseen events.

Relative Frequency Estimate  Good Turing Estimate
Good-Turing smoothing

MLE

$$P(\text{seen}) + P(\text{unseen}) = 1$$

$$\frac{N}{N} + 0 = 1$$

Good Turing

$$\frac{2 \cdot N_2 + \ldots + m \cdot N_m}{\sum_{i=1}^{m} i \cdot N_i} + \frac{1 \cdot N_1}{\sum_{i=1}^{m} i \cdot N_i} = \frac{\sum_{i=1}^{m} i \cdot N_i}{\sum_{i=1}^{m} i \cdot N_i}$$

$$N_c: \text{number of event types that occur } c \text{ times (can be counted)}$$

$$N_1: \text{number of event types that occur once}$$

$$N = 1N_1 + \ldots + mN_m: \text{total number of observed event tokens}$$
Good-Turing Smoothing

General principle:
Reassign the probability mass of all events that occur \( k \) times in the training data to all events that occur \( k - 1 \) times.

\( N_k \) events occur \( k \) times, with a total frequency of \( k \cdot N_k \)

The probability mass of all words that appear \( k - 1 \) times becomes:

\[
\sum_{w: C(w) = k - 1} P_{GT}(w) = \sum_{w': C(w') = k} P_{MLE}(w') = \sum_{w': C(w') = k} \frac{k}{N} = \frac{k \cdot N_k}{N}
\]

There are \( N_{k-1} \) words \( w \) that occur \( k - 1 \) times in the training data. Good-Turing replaces the original count \( c_{k-1} \) of \( w \) with a new count \( c^*_{k-1} \):

\[
c^*_{k-1} = \frac{k \cdot N_k}{N_{k-1}}
\]
Good-Turing smoothing

The Maximum Likelihood estimate of the probability of a word $w$ that occurs $k-1$ times $P_{MLE}(w) = \frac{C(w)}{N}$

$$P_{MLE}(w) = \frac{c_{k-1}}{N} = \frac{k-1}{N}$$

The Good-Turing estimate of the probability of a word $w$ that occurs $k-1$ times: $P_{GT}(w) = c^*_{k-1} / N$:

$$P_{GT}(w) = \frac{c^*_{k-1}}{N} = \frac{k \cdot N_k}{N_{k-1}} = \frac{k \cdot N_k}{N \cdot N_{k-1}}$$
Problems with Good-Turing

Problem 1:
What happens to the most frequent event?

Problem 2:
We don’t observe events for every $k$.

Variant: Simple Good-Turing
Replace $N_n$ with a fitted function $f(n)$:

$$f(n) = a + b \log(n)$$

Requires parameter tuning (on held-out data):
Set $a, b$ so that $f(n) \approx N_n$ for known values.
Use $c_n*$ only for small $n$
Smoothing:
Reserving mass in $P(X | Y)$ for unseen events
Linear Interpolation (1)

We don’t see “Bob was reading”, but we see “__ was reading”. We estimate $P(\text{reading} | 'Bob was') = 0$ but $P(\text{reading} | 'was') > 0$

Use $(n - 1)$-gram probabilities to smooth $n$-gram probabilities:

$$\hat{P}_\text{LI}(w_i|w_{i-n}w_{i-n+1}\ldots w_{i-2}w_{i-1}) = \lambda \hat{P}(w_i|w_{i-n}w_{i-n+1}\ldots w_{i-2}w_{i-1}) + (1 - \lambda) \hat{P}_\text{LI}(w_i|w_{i-n+1}\ldots w_{i-2}w_{i-1})$$
Linear Interpolation (2)

We’ve never seen “Bob was reading”,
but we might have seen “__ was reading”,
and we’ve certainly seen “__ reading” (or <UNK>)

\[
\tilde{P}(w_i|w_{i-1}, w_{i-2}) = \lambda_3 \cdot \hat{P}(w_i|w_{i-1}, w_{i-2}) \\
+ \lambda_2 \cdot \hat{P}(w_i|w_{i-1}) \\
+ \lambda_1 \cdot \hat{P}(w_i) \\
\text{for } \lambda_1 + \lambda_2 + \lambda_3 = 1
\]
Interpolation: Setting the $\lambda$s

Method A: Held-out estimation
Divide data into training and held-out data. Estimate models on training data. Use held-out data (and some optimization technique) to find the $\lambda$ that gives best model performance.
(We’ll talk about evaluation later)

$\lambda$ can also depend on $w_{i-n}...w_{i-1}$

Method B:
$\lambda$ is some function of the frequencies of $w_{i-n}...w_{i-1}$
Absolute discounting

Subtract a constant factor $D < 1$ from each nonzero $n$-gram count, and interpolate with $P_{AD}(w_i \mid w_{i-1})$:

$$P_{AD}(w_i \mid w_{i-1}, w_{i-2}) = \frac{\max(C(w_{i-2}w_{i-1}w_i) - D, 0)}{C(w_{i-2}w_{i-1})} + (1 - \lambda)P_{AD}(w_i \mid w_{i-1})$$

If $S$ seen word types occur after $w_{i-2}w_{i-1}$ in the training data, this reserves the probability mass $P(U) = (S \times D)/C(w_{i-2}w_{i-1})$ to be computed according to $P(w_i \mid w_{i-1})$. Set:

$$(1 - \lambda) = P(U) = \frac{S \cdot D}{C(w_{i-2}w_{i-1})}$$

N.B.: with $N_1, N_2$ the number of $n$-grams that occur once or twice, $D = N_1/(N_1+2N_2)$ works well in practice
Kneser-Ney smoothing

Observation: “San Francisco” is frequent, but “Francisco” only occurs after “San”.

Solution: the unigram probability $P(w)$ should not depend on the frequency of $w$, but on the number of contexts in which $w$ appears

\[
N_{+1}(\bullet w): \text{number of contexts in which } w \text{ appears} = \text{number of word types } w' \text{ which precede } w
\]

\[
N_{+1}(\bullet \bullet) = \sum_{w'} N_{+1}(\bullet w')
\]

Kneser-Ney smoothing: Use absolute discounting, but use $P(w) = N_{+1}(\bullet w)/N_{+1}(\bullet \bullet)$

Modified Kneser-Ney smoothing: Use different $D$ for bigrams and trigrams (Chen & Goodman ’98)
To recap....
Today’s key concepts

Dealing with unknown words
Dealing with unseen events
Good-Turing smoothing
Linear Interpolation
Absolute Discounting
Kneser-Ney smoothing

Today’s reading:
Jurafsky and Martin, Chapter 4, sections 1-4