Assignment 2 out
- bigram language modeling
- Java
- Can work with partners
  - Anyone looking for a partner?
- Due Wednesday 2/16 (but start working on it now!)
- HashMap

Our first quiz next Monday (2/14)
- In-class (~30 min.)
- Topics
  - corpus analysis
  - regular expressions
  - probability
  - language modeling
- Open book
  - we’ll try it out for this one
  - better to assume closed book (30 minutes goes by fast!)
- 5% of your grade
Take home ideas:
- Key idea of smoothing is to redistribute the probability to handle less see (or never seen) events
- Still must always maintain a true probability distribution
- Lots of ways of smoothing data
- Should take into account features in your data!
- For n-grams, backoff models and, in particular, Kneser-Ney smoothing work well

Smoothing

P(I think today is a good day to be me) =

P(I) \times P(\text{think} | I) \times P(\text{today} | \text{think}) \times P(\text{is} | \text{think today}) \times P(\text{a} | \text{today is}) \times P(\text{good} | \text{is a}) \times \ldots

If any of these has never been seen before, prob = 0!

These probability estimates may be inaccurate. Smoothing can help reduce some of the noise.
Add-lambda smoothing

- A large dictionary makes novel events too probable.
- add $\lambda = 0.01$ to all counts

<table>
<thead>
<tr>
<th>Count</th>
<th>Smoothed Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>see the abacus</td>
<td>1</td>
</tr>
<tr>
<td>see the abbot</td>
<td>0</td>
</tr>
<tr>
<td>see the abduct</td>
<td>0</td>
</tr>
<tr>
<td>see the above</td>
<td>2</td>
</tr>
<tr>
<td>see the Abram</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>see the zygote</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>3</td>
</tr>
</tbody>
</table>

Vocabulary

- n-gram language modeling assumes we have a fixed vocabulary
  - why?

- Whether implicit or explicit, an n-gram language model is defined over a finite, fixed vocabulary

- What happens when we encounter a word not in our vocabulary (Out Of Vocabulary)?
  - If we don’t do anything, prob = 0
  - Smoothing doesn’t really help us with this!

Vocabulary

- To make this explicit, smoothing helps us with…
  - all entries in our vocabulary

<table>
<thead>
<tr>
<th>Count</th>
<th>Smoothed Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>see the abacus</td>
<td>1</td>
</tr>
<tr>
<td>see the abbot</td>
<td>0</td>
</tr>
<tr>
<td>see the abduct</td>
<td>0</td>
</tr>
<tr>
<td>see the above</td>
<td>2</td>
</tr>
<tr>
<td>see the Abram</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>see the zygote</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>3</td>
</tr>
</tbody>
</table>

Vocabulary

- and…

<table>
<thead>
<tr>
<th>Vocabulary</th>
<th>Counts</th>
<th>Smoothed counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>10</td>
<td>10.01</td>
</tr>
<tr>
<td>able</td>
<td>1</td>
<td>1.01</td>
</tr>
<tr>
<td>about</td>
<td>2</td>
<td>2.01</td>
</tr>
<tr>
<td>account</td>
<td>0</td>
<td>0.01</td>
</tr>
<tr>
<td>acid</td>
<td>0</td>
<td>0.01</td>
</tr>
<tr>
<td>across</td>
<td>3</td>
<td>3.01</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>young</td>
<td>1</td>
<td>1.01</td>
</tr>
<tr>
<td>zebra</td>
<td>0</td>
<td>0.01</td>
</tr>
</tbody>
</table>

How can we have words in our vocabulary we’ve never seen before?
**Vocabulary**

- Choosing a vocabulary: ideas?
  - Grab a list of English words from somewhere
  - Use all of the words in your training data
  - Use some of the words in your training data
    - for example, all those the occur more than $k$ times
- Benefits/drawbacks?
  - Ideally your vocabulary should represents words your likely to see
  - Too many words, end up washing out your probability estimates (and getting poor estimates)
  - Too few, lots of out of vocabulary

**Out of vocabulary**

- Add an extra word in your vocabulary to denote OOV (<OOV>, <UNK>)
- Replace all words in your training corpus not in the vocabulary with <UNK>
  - You’ll get bigrams, trigrams, etc with <UNK>
    - $p(<UNK> | "I am")$
    - $p(\text{fast} | "I <UNK>")$
- During testing, similarly replace all OOV with <UNK>

**Vocabulary**

- No matter your chosen vocabulary, you’re still going to have out of vocabulary (OOV)
- How can we deal with this?
  - Ignore words we’ve never seen before
    - Somewhat unsatisfying, though can work depending on the application
    - Probability is then dependent on how many in vocabulary words are seen in a sentence/text
  - Use a special symbol for OOV words and estimate the probability of out of vocabulary

**Choosing a vocabulary**

- A common approach (and the one we’ll use for the assignment):
  - Replace the first occurrence of each word by <UNK> in a data set
  - Estimate probabilities normally
- Vocabulary then is all words that occurred two or more times
- This also discounts all word counts by 1 and gives that probability mass to <UNK>
Storing the table

How are we storing this table?
Should we store all entries?

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1/3</th>
<th>1.01</th>
<th>1.01/203</th>
</tr>
</thead>
<tbody>
<tr>
<td>see the abacus</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>see the abbot</td>
<td>0</td>
<td>0/3</td>
<td>0.01</td>
<td>0.01/203</td>
</tr>
<tr>
<td>see the absc</td>
<td>0</td>
<td>0/3</td>
<td>0.01</td>
<td>0.01/203</td>
</tr>
<tr>
<td>see the above</td>
<td>2</td>
<td>2/3</td>
<td>2.01</td>
<td>2.01/203</td>
</tr>
<tr>
<td>see the Abr in</td>
<td>0</td>
<td>0/3</td>
<td>0.01</td>
<td>0.01/203</td>
</tr>
<tr>
<td>see the zygote</td>
<td>0</td>
<td>0/3</td>
<td>0.01</td>
<td>0.01/203</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>3</td>
<td>3/3</td>
<td>203</td>
<td></td>
</tr>
</tbody>
</table>

Storing the table

- Hashtable
  - fast retrieval
  - fairly good memory usage
- Only store entries of things we’ve seen
  - for example, we don’t store V^2 trigrams
- For trigrams we can:
  - Store one hashtable with bigrams as keys
  - Store a hashtable of hashtables (I’m recommending this)

Storing the table: add-lambda smoothing

- For those we’ve seen before:
  \[ P(c\mid ab) = \frac{C(abc) + \lambda}{C(ab) + \lambda V} \]
- Unseen n-grams: \( p(z\mid ab) = ? \)
  \[ P(z\mid ab) = \frac{\lambda}{C(ab) + \lambda V} \]

How common are novel events?

How likely are novel/unseen events?

Store the lower order counts
(or probabilities)
How common are novel events?

If we follow the pattern, something like this…

Good-Turing estimation

- $N_c = \text{number of words/bigrams occurring } c \text{ times}$
- Replace MLE counts for things with count $c$:
  
  $$
c^* = (c + 1) \frac{N_{c+1}}{N_c}
  $$

  scale down the next frequency up

- Estimate the probability of novel events as:
  
  $$
p(\text{unseen}) = \frac{N_c}{\text{Total words}}
  $$

Good-Turing (classic example)

- Imagine you are fishing
  - 8 species: carp, perch, whitefish, trout, salmon, eel, catfish, bass
- You have caught
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish
- How likely is it that the next fish caught is from a new species (one not seen in our previous catch)?
  
  $$
p(\text{unseen}) = \frac{N_c}{\text{Total words}} = \frac{3}{18}
  $$
Good-Turing (classic example)

- Imagine you are fishing
  - 8 species: carp, perch, whitefish, trout, salmon, eel, catfish, bass
- You have caught
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish
- How likely is it that next species is trout?

\[
\hat{c} = \frac{(c + 1) \frac{N_{x+1}}{N}}{c} = \frac{2 \times \frac{1}{3}}{18} = 0.67
\]

Problems with frequency based smoothing

- The following bigrams have never been seen:

\[
p(X \mid \text{San}) \quad \text{and} \quad p(X \mid \text{ate})
\]

Which would add-a-lambda pick as most likely?

Which would you pick?

Witten-Bell Discounting

- Some words are more likely to be followed by new words

Diego
Francisco
San
José
Marcos

food
apples
bananas
hamburgers
ate
a lot
for two
grapes
…
Witten-Bell Discounting

- Probability mass is shifted around, depending on the context of words
- If $P(w_i \mid w_{i-1}, \ldots, w_{i-m}) = 0$, then the smoothed probability $P_{WB}(w_i \mid w_{i-1}, \ldots, w_{i-m})$ is higher if the sequence $w_{i-1}, \ldots, w_{i-m}$ occurs with many different words $w_i$

Witten-Bell Smoothing

- For bigrams
  - $T(w_{i-1})$ is the number of different words (types) that occur to the right of $w_{i-1}$
  - $N(w_{i-1})$ is the number of times $w_{i-1}$ occurred
  - $Z(w_{i-1})$ is the number of bigrams in the current data set starting with $w_{i-1}$ that do not occur in the training data

Witten-Bell Smoothing

- If $c(w_{i-1}, w_i) > 0$
  \[ P_{WB}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{N(w_{i-1}) + T(w_{i-1})} \]
  \# times we saw the bigram
  \# times $w_{i-1}$ occurred + \# of types to the right of $w_{i-1}$

- If $c(w_{i-1}, w_i) = 0$
  \[ P_{WB}(w_i \mid w_{i-1}) = \frac{T(w_{i-1})}{Z(w_{i-1})(N + T(w_{i-1}))} \]
Problems with frequency based smoothing

- The following trigrams have never been seen:
  
  \[ p(\text{car} \mid \text{see the}) \quad p(\text{zygote} \mid \text{see the}) \quad p(\text{cumquat} \mid \text{see the}) \]

Which would add-lambda pick as most likely? Good-Turing? Witten-Bell?
Which would you pick?

Better smoothing approaches

- Utilize information in lower-order models
- Interpolation
  \[ p(x \mid y) = \lambda p(x \mid y) + \mu p(x \mid y) + (1 - \lambda - \mu) p(x) \]
  Combine the probabilities in some linear combination
- Backoff
  \[ p(z \mid xy) = \begin{cases} 
  \frac{C(xyz)}{C(xy)} & \text{if } C(xyz) \geq k \\
  \frac{C(y)}{C(y)} & \text{otherwise}
  \end{cases} \]
  Often \( k = 0 \) (or 1)
  Combine the probabilities by “backing off” to lower models only when we don’t have enough information

Smoothing: Simple Interpolation

\[ P(z \mid xy) = \lambda C(xyz) C(xy) + \mu C(yz) C(y) + (1 - \lambda - \mu) C(z) C(\cdot) \]

- Trigram is very context specific, very noisy
- Unigram is context-independent, smooth
- Interpolate Trigram, Bigram, Unigram for best combination
- How should we determine \( \lambda \) and \( \mu \)?

Smoothing: Finding parameter values

- Just like we talked about before, split training data into training and development
  - can use cross-validation, leave-one-out, etc.
- Try lots of different values for \( \lambda, \mu \) on heldout data, pick best
- Two approaches for finding these efficiently
  - EM (expectation maximization)
  - “Powell search” — see Numerical Recipes in C
Smoothing: Jelinek-Mercer

- Simple interpolation:
  \[ P_{\text{smooth}}(z \mid xy) = \lambda \frac{C(xyz)}{C(xy)} + (1 - \lambda)P_{\text{smooth}}(z \mid y) \]
- Should all bigrams be smoothed equally? Which of these is it more likely to start an unseen trigram?

Smoothing: Jelinek-Mercer continued

- Bin counts by frequency and assign \( \lambda \)s for each bin
- Find \( \lambda \)s by cross-validation on held-out data

Backoff models: absolute discounting

- Subtract some absolute number from each of the counts (e.g., 0.75)
- will have a large effect on low counts
- will have a small effect on large counts

\[ P_{\text{absolute}}(z \mid xy) = \begin{cases} C(xyz) - D & \text{if } C(xyz) > 0 \\ \frac{C(xy)}{C(x)} & \text{otherwise} \end{cases} \]
Backoff models: absolute discounting

\[ P_{\text{absolute}}(z | xy) = \begin{cases} C(xyz) & \text{if } C(xyz) > 0 \\ \alpha(xy) P_{\text{absolute}}(z | y) & \text{otherwise} \end{cases} \]

What is \( \alpha(xy) \)?

Backoff models: absolute discounting

<table>
<thead>
<tr>
<th>Event</th>
<th>Count</th>
<th>Event</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>see the dog</td>
<td>1</td>
<td>see the cat</td>
<td>2</td>
</tr>
<tr>
<td>see the banana</td>
<td>4</td>
<td>see the man</td>
<td>1</td>
</tr>
<tr>
<td>see the woman</td>
<td>1</td>
<td>see the woman</td>
<td>1</td>
</tr>
<tr>
<td>see the car</td>
<td>1</td>
<td>see the car</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ P_{\text{absolute}}(z | xy) = \begin{cases} \frac{C(xyz) - D C(xy)}{C(xy)} & \text{if } C(xyz) > 0 \\ \alpha(xy) P_{\text{absolute}}(z | y) & \text{otherwise} \end{cases} \]

How much probability mass did we reserve/discount for the bigram model?
Calculating $\alpha$

- We have some number of bigrams we’re going to backoff to, i.e. those $X$ where $C(\text{see the } X) = 0$, that is unseen trigrams starting with “see the”.
- When we backoff, for each of these, we’ll be including their probability in the model: $P(X | \text{the})$.
- $\alpha$ is the normalizing constant so that the sum of these probabilities equals the reserved probability mass.

\[ \sum_{X:C(\text{see the } X) = 0} p(X | \text{the}) = \text{reserved mass(see the)} \]
Calculating $\alpha$ in general: trigrams

- Calculate the reserved mass
  \[ \text{reserved}_\text{mass}(\text{bigram}) = \frac{\# \text{ of types starting with bigram } \times D}{\text{count(bigram)}} \]
- Calculate the sum of the backed off probability. For bigram "A B":
  \[ 1 - \sum_{X \in \text{C(A X) > 0}} p(X) \]
- Calculate $\alpha$ (bigram):
  \[ \alpha(A B) = \frac{\text{reserved}_\text{mass}(A B)}{1 - \sum_{X \in \text{C(A X) > 0}} p(X)} \]

Calculating $\alpha$ in general: bigrams

- Calculate the reserved mass
  \[ \text{reserved}_\text{mass}(\text{unigram}) = \frac{\# \text{ of types starting with unigram } \times D}{\text{count(unigram)}} \]
- Calculate the sum of the backed off probability. For bigram "A B":
  \[ 1 - \sum_{X \in \text{C(A X) > 0}} p(X) \]
- Calculate $\alpha$ (unigram):
  \[ \alpha(A) = \frac{\text{reserved}_\text{mass}(A)}{1 - \sum_{X \in \text{C(A X) > 0}} p(X)} \]

Calculating backoff models in practice

- Store the $\alpha$'s in another table
  - If it's a trigram backed off to a bigram, it's a table keyed by the bigrams
  - If it's a bigram backed off to a unigram, it's a table keyed by the unigrams
- Compute the $\alpha$'s during training
  - After calculating all the probabilities of seen unigrams/bigrams/trigrams
  - Go back through and calculate the $\alpha$'s (you should have all of the information you need)
  - During testing, it should then be easy to apply the backoff model with the $\alpha$'s pre-calculated

Backoff models: absolute discounting

\[ p(\text{jumped \mid the Dow}) = \frac{20}{20} = 1 \]
\[ \alpha(\text{the Dow}) = \frac{\# \text{ of types starting with "see the" } \times D}{\text{count("see the")}} \]
\[ \text{reserved}_\text{mass}(\text{the Dow}) = \frac{3 \times D}{20} = \frac{3 \times 0.75}{20} = 0.115 \]
\[ \alpha(\text{the Dow}) = \frac{\text{reserved}_\text{mass(see the)}}{1 - \sum_{X \in \text{C(see the X) > 0}} p(X)} \]
Backoff models: absolute discounting

\[
\text{reserved_mass} = \frac{\text{# of types starting with bigram} \times D}{\text{count(bigram)}}
\]

- Two nice attributes:
  - decreases if we've seen more bigrams
    - should be more confident that the unseen trigram is no good
  - increases if the bigram tends to be followed by lots of other words
    - will be more likely to see an unseen trigram

Kneser-Ney

- Idea: not all counts should be discounted with the same value
  
  \[P(\text{Francisco} \mid \text{eggplant}) \text{ vs } P(\text{stew} \mid \text{eggplant})\]

  - If we've never seen either, which should be more likely? why?
  - What would an normal discounted backoff model say?
  - What is the problem?

Kneser-Ney

- Idea: not all counts should be discounted with the same value
  
  \[P(\text{Francisco} \mid \text{eggplant}) \text{ vs } P(\text{stew} \mid \text{eggplant})\]

  - “Francisco” is common, so backoff/interpolated methods say it is likely
    - But it only occurs in context of “San”
    - “Stew” is common in many contexts
  - Weight backoff by number of contexts word occurs in
    
    \[P(\text{Francisco} \mid \text{eggplant}) \text{ vs } P(\text{stew} \mid \text{eggplant})\]
Kneser-Ney

\[
P_{\text{absolute}}(z|xy) = \begin{cases} 
\frac{C(xyz) - D}{C(xy)} & \text{if } C(xyz) > 0 \\
\alpha(xy)P_{\text{absolute}}(z|y) & \text{otherwise}
\end{cases}
\]

Instead of the probability of the word/bigram occurring, use the probability of the

\[
P_{\text{CONTINUATION}}(z|1, y) = \begin{cases} 
\frac{C(xyz) - D}{C(xy)} & \text{if } C(xyz) > 0 \\
\alpha(xy)P_{\text{CONTINUATION}}(z|y) & \text{otherwise}
\end{cases}
\]

Other language model ideas?

- Skipping models: rather than just the previous 2 words, condition on the previous word and the 3rd word back, etc.
- Caching models: phrases seen are more likely to be seen again (helps deal with new domains)
- Clustering:
  - some words fall into categories (e.g. Monday, Tuesday, Wednesday…)
  - smooth probabilities with category probabilities
- Domain adaptation:
  - interpolate between a general model and a domain specific model

Smoothing results
Language Modeling Toolkits

- SRI
- CMU
  - [http://www.speech.cs.cmu.edu/SLM_info.html](http://www.speech.cs.cmu.edu/SLM_info.html)