Language acquisition

- [http://www.youtube.com/watch?v=RE4ce4mexrU](http://www.youtube.com/watch?v=RE4ce4mexrU)

---

**LANGUAGE MODELING: SMOOTHING**

David Kauchak
CS457 – Fall 2011

Some slides adapted from Jason Eisner

---

**Admin**

- Assignment 2 out
  - bigram language modeling
  - Java
  - Can work with partners
  - Due Wednesday 10/5
  - Style/commenting (JavaDoc)
  - Some advice
    - Start now!
    - Spend 1-2 hours working out an example by hand (you can check your answers with me)
    - HashMap

---

**Admin**

- Our first quiz next Tuesday (10/4)
  - 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!)
  - 7.5% of your grade
Today

Take home ideas:
- Key idea of smoothing is to redistribute the probability to handle less seen (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!

Smoothing

What if our test set contains the following sentence, but one of the trigrams never occurred in our training data?

\[
P(I \text{ think today is a good day to be me}) =
\]
\[
P(I | \langle\text{start}\rangle \langle\text{start}\rangle) \times
\]
\[
P(\text{think} | \langle\text{start}\rangle I) \times
\]
\[
P(\text{today} | I \text{ think}) \times
\]
\[
P(a | \text{today is}) \times
\]
\[
P(\text{good} | is a) \times
\]
... If any of these has never been seen before, prob = 0!

Smoothing

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></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>
</tr>
<tr>
<td>see the abbot</td>
<td>0</td>
<td>0.01</td>
<td>0.01/203</td>
</tr>
<tr>
<td>see the abduct</td>
<td>2</td>
<td>2.01</td>
<td>2.01/203</td>
</tr>
<tr>
<td>see the above</td>
<td>0</td>
<td>0.01</td>
<td>0.01/203</td>
</tr>
<tr>
<td>see the Abram</td>
<td>0</td>
<td>0.01</td>
<td>0.01/203</td>
</tr>
<tr>
<td>see the zygo</td>
<td>0</td>
<td>0.01</td>
<td>0.01/203</td>
</tr>
<tr>
<td>Total</td>
<td>3</td>
<td>3/3</td>
<td>203</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></th>
<th>1.01</th>
</tr>
</thead>
<tbody>
<tr>
<td>see the abacus</td>
<td>1.01</td>
</tr>
<tr>
<td>see the abbot</td>
<td></td>
</tr>
<tr>
<td>see the abduct</td>
<td></td>
</tr>
<tr>
<td>see the above</td>
<td></td>
</tr>
<tr>
<td>see the Abram</td>
<td></td>
</tr>
<tr>
<td>see the zygo</td>
<td></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?
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 that occur more than \( k \) times

Benefits/drawbacks?
- Ideally your vocabulary should represent words you’re likely to see
- Too many words: end up washing out your probability estimates (and getting poor estimates)
- Too few: lots of out of 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

Add an extra word in your vocabulary to denote OOV \(<\text{OOV}>, <\text{UNK}>\)

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

A common approach (and the one we’ll use for the assignment):
- Replace the first occurrence of each word by \(<\text{UNK}>\) 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 \(<\text{UNK}>\)
Storing the table

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

<table>
<thead>
<tr>
<th>Entry</th>
<th>Count</th>
<th>Count/3</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>see the abacus</td>
<td>1</td>
<td>1/3</td>
<td>1.00</td>
</tr>
<tr>
<td>see the abbot.</td>
<td>0</td>
<td>0/3</td>
<td>0.01</td>
</tr>
<tr>
<td>see the abduct.</td>
<td>0</td>
<td>0/3</td>
<td>0.01</td>
</tr>
<tr>
<td>see the above.</td>
<td>2</td>
<td>2/3</td>
<td>2.01</td>
</tr>
<tr>
<td>see the Abram.</td>
<td>0</td>
<td>0/3</td>
<td>0.01</td>
</tr>
<tr>
<td>see the zygot.</td>
<td>0</td>
<td>0/3</td>
<td>0.01</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>3</td>
<td>3/3</td>
<td>203</td>
</tr>
</tbody>
</table>

Hashtable
- fast retrieval
- fairly good memory usage

Only store those 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:

\[ P(c|ab) = \frac{C(ab)c + \lambda}{C(ab) + \lambda V} \]

Unseen n-grams: \( p(z|ab) = ? \)

\[ P(z|ab) = \frac{\lambda}{C(ab) + \lambda V} \]

Store the lower order counts (or probabilities)

How common are novel events?

<table>
<thead>
<tr>
<th>Number of words occurring ( \lambda ) times in the corpus</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>

How likely are novel/unseen events?
How common are novel events?

If we follow the pattern, something like this...

Good-Turing estimation

- \( N_c \) = number of words/bigrams occurring \( c \) times
- Estimate the probability of novel events as:
  \[
  p(\text{unseen}) = \frac{N_c}{N_c + 1}
  \]
  \( N_c \) = number of words/bigrams occurring \( c \) times
  \( N_c + 1 \) = total words

- Replace MLE counts for things with count \( c \):
  \[
  c^* = (c + 1) \frac{N_c + 1}{N_c}
  \]
  scale down the next frequency up

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)?
  \[
  p(\text{unseen}) = \frac{N_c}{N_c + 1} = \frac{3}{18}
  \]
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?

\[ c^* = \frac{(c + 1) N_{c+1}}{N_c} = \frac{2(1)}{3} = 0.67 \]

Problems with frequency based smoothing

- The following bigrams have never been seen:
  - \( p(X \mid \text{San}) \)
  - \( p(X \mid \text{ate}) \)

Which would add-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
- Luis
- Jose
- Marcos

- food
- apples
- bananas
- hamburgers
- 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 \( w_k \)

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

- 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:
  \[ p(\text{ car | see the }) \quad p(\text{ zygote | see the }) \]
  \[ p(\text{ cumquat | see the }) \]

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
  \[ p^*(x | x,y) = \lambda p(x | x,y) + \mu p(x | y) + (1-\lambda - \mu)p(x) \]
  Combine the probabilities in some linear combination
- Backoff
  \[ P_{\text{Backoff}}(z | x,y) = \begin{cases} 
  \frac{C^*(z | x,y)}{C^*(x,y)} & \text{if } C^*(z | x,y) > k \\
  0 & \text{otherwise}
  \end{cases} \]
  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

\[ P(z | x,y) = \frac{C(z | x,y)}{C(x,y)} + \mu \frac{C(z | y)}{C(y)} + (1-\lambda - \mu) \frac{C(z)}{C(*)} \]

- 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 more likely to start an unseen trigram?

\[ P_{\text{smooth}}(z \mid xy) = \lambda \frac{C(xyz)}{C(xy)} + (1 - \lambda)P_{\text{smooth}}(z \mid y) \]

Multiple parameters based on frequency bins: smooth a little after “The Dow”, more after “Adobe acquired”

\[ P_{\text{smooth}}(z \mid xy) = \lambda(C(xy)) \frac{C(xyz)}{C(xy)} + (1 - \lambda(C(xy)))P_{\text{smooth}}(z \mid y) \]

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

\[ P_{\text{smooth}}(z \mid xy) = \lambda(C(xy)) \frac{C(xyz)}{C(xy)} + (1 - \lambda(C(xy)))P_{\text{smooth}}(z \mid y) \]

Backoff models: absolute discounting

\[ P_{\text{absolute}}(z \mid xy) = \begin{cases} \frac{C(xyz)}{C(xy)} - D & \text{if } C(xyz) > 0 \\ \frac{C(xy)}{C(xy)}(1 - P_{\text{smooth}}(z \mid y)) & \text{otherwise} \end{cases} \]

- Subtract some absolute number from each of the counts (e.g. 0.75)
- How will this affect rare words?
- How will this affect common words?
Backoff models: absolute discounting

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

- Subtract some absolute number from each of the counts (e.g. 0.75)
- Will have a large effect on low counts (rare words)
- Will have a small effect on large counts (common words)

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

| See the dog | 1 | See the cat | 2 | See the banana | 4 | See the man | 1 | See the woman | 1 | See the car | 1 |
|---|---|---|---|---|---|---|---|---|---|---|
| the Dow Jones | 10 | the Dow rose | 5 | the Dow fell | 5 | |
| p( rose | the Dow ) = \( \frac{4}{10} = 0.4 \) | p( jumped | the Dow ) = \( \frac{4}{10} = 0.4 \) |
| p( cat | see the ) = \( \frac{1}{10} = 0.1 \) | p( puppy | see the ) = \( \frac{1}{10} = 0.1 \) |

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

- see the dog 1
- see the cat 2
- see the banana 4
- see the man 1
- see the woman 1
- see the car 1

\[ p(\text{puppy} | \text{see the}) = \] ?
\[ \alpha(\text{see the}) = \] ?

How much probability mass did we reserve/discount for the bigram model?

\[ \text{reserved}_\text{mass}(\text{see the}) = \frac{6 \times 0.25}{10} = 0.45 \]

\[ \text{distribute this probability mass to all bigrams that we are backing off to} \]

Backoff models: absolute discounting

- see the dog 1
- see the cat 2
- see the banana 4
- see the man 1
- see the woman 1
- see the car 1

\[ p(\text{puppy} | \text{see the}) = \] ?
\[ \alpha(\text{see the}) = \] ?

For each of the unique trigrams, we subtracted \( D / \text{count(“see the”)} \) from the probability distribution.

\[ P(\text{absolute}) (z | xy) = \]
\[ \text{if } C(xy) > 0 \]
\[ \text{count}(“see the”) \]
\[ \text{otherwise} \]

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

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

- Calculate the reserved mass
  \[ \text{reserved\_mass}(\text{bigram}) = \frac{\# \text{ of types starting with bigram}}{D} \cdot \text{count(\text{bigram})} \]

- Calculate the sum of the backed off probability for bigram "A B":
  \[ 1 - \sum_{X : (A X) > 0} p(X B) \]

- Or, if it's fine in practice, the left is easier:
  \[ \sum_{X : (A X) > 0} p(X B) \]

- Calculate $\alpha$
  \[ \alpha(A B) = \frac{\text{reserved\_mass}(A B)}{1 - \sum_{X : (A X) > 0} p(X B)} \]

  1 – the sum of the bigram probabilities of those trigrams that we saw starting with bigram A B

Calculating $\alpha$ in general: bigrams

- Calculate the reserved mass
  \[ \text{reserved\_mass}(\text{unigram}) = \frac{\# \text{ of types starting with unigram}}{D} \cdot \text{count(\text{unigram})} \]

- Calculate the sum of the backed off probability for bigram "A B":
  \[ 1 - \sum_{X : (A X) > 0} p(X) \]

- Or, if it's fine in practice, the left is easier:
  \[ \sum_{X : (A X) > 0} p(X) \]

- Calculate $\alpha$
  \[ \alpha(A) = \frac{\text{reserved\_mass}(A)}{1 - \sum_{X : (A X) > 0} p(X)} \]

  1 – the sum of the unigram probabilities of those bigrams that we saw starting with word A

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 of 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

| the Dow Jones  | 10 | $p(\text{jumped} \mid \text{the Dow}) =$ |
| the Dow rose   | 5  | What is the reserved mass? |
| the Dow fell   | 5  | |

$\text{reserved\_mass}(\text{the Dow}) = \frac{3 \cdot D}{20} = \frac{3 \cdot 0.75}{20} = 0.115$

$\alpha(\text{the Dow}) = \frac{\text{reserved\_mass}(\text{see the})}{1 - \sum p(X \mid \text{the})}$

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

$P(\text{Francisco} \mid \text{eggplant})$ vs $P(\text{stew} \mid \text{eggplant})$

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

Problem:
- Both of these would have the same backoff parameter since they’re both conditioning on eggplant
- If we then would end up picking based on which was most frequent
- However, even though Francisco tends to only be preceded by a small number of words
Kneser-Ney

- Idea: not all counts should be discounted with the same value
- “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

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

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

**P_{\text{continuation}}**

- Relative to other words, how likely is this word to continue (i.e. follow) many other words

\[ P_{\text{continuation}}(z|y) = \frac{\# \text{ types ending with } yc}{\sum_{h \in \text{bigrams}} \# \text{ types ending with bigram } hc} \]

= \frac{\# \{ \text{types ending with } yc \}}{\sum_{h \in \text{bigrams}} \# \{ \text{types ending with bigram } hc \}}

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

- Key idea of smoothing is to redistribute the probability to handle less seen (or never seen) events
  - 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

Take home ideas

Language Modeling Toolkits

- SRI
- CMU
  - http://www.speech.cs.cmu.edu/SLM_info.html