Language acquisition

http://www.youtube.com/watch?v=RE4ce4mexrU
(4:30)

Assignments

Assignment 2 out
- bigram language modeling
- Java
- Can work with partners
  - Anyone looking for a partner?
- 2a: Due this Friday
- 2b: Due next Friday
- 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

Our first quiz (when?)
- In-class (~30 min.)
- Topics
  - corpus analysis
  - regular expressions
  - probability
  - language modeling
- Open book/notes
  - we’ll try it out for this one
  - better to assume closed book (30 minutes goes by fast!)
- 10% of your grade
Admin

Lab next class

Meet in Edmunds 105

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?

\[ P(\text{I think today is a good day to be me}) = P(\text{I}) \times P(\text{think} | \text{I}) \times P(\text{today} | \text{think}) \times P(\text{is} | \text{today}) \times P(\text{a} | \text{is}) \times P(\text{good} | \text{a}) \times \ldots \]

If any of these has never been seen before, \( P = 0! \)
Smoothing

\[ P(\text{I think today is a good day to be me}) = \]
\[ P(\text{I} | \text{<start> <start>}) \times P(\text{think} | \text{<start> I}) \times P(\text{today} | \text{I think}) \times P(\text{is} | \text{think today}) \times P(\text{a} | \text{today is}) \times P(\text{good} | \text{is a}) \times \ldots \]

These probability estimates may be inaccurate. Smoothing can help reduce some of the noise.

The general smoothing problem

<table>
<thead>
<tr>
<th>Modification</th>
<th>Probability</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>see the zygote</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>3</td>
</tr>
</tbody>
</table>

Add-lambda smoothing

A large dictionary makes novel events too probable.

\[ \text{add } \lambda = 0.01 \text{ to all counts} \]

<table>
<thead>
<tr>
<th>Modification</th>
<th>Probability</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>see the zygote</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>3</td>
</tr>
</tbody>
</table>

Add-lambda smoothing

How should we pick \( \lambda \)?

<table>
<thead>
<tr>
<th>Modification</th>
<th>Probability</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>see the zygote</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>3</td>
</tr>
</tbody>
</table>
Setting smoothing parameters

Idea 1: try many $\lambda$ values & report the one that gets the best results?

Is this fair/appropriate?

Test

Training

Is this fair/appropriate?

Setting smoothing parameters

Collect counts from 80% of the data

Pick $\lambda$ that gets best results on 20%

Now use that $\lambda$ to get smoothed counts from all 100%...

… and report results of that final model on test data.

Vocabulary

n-gram language modeling assumes we have a fixed vocabulary

Why?

Probability distributions are over finite events!

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>Word</th>
<th>Count</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>see the abacus</td>
<td>1</td>
<td>1.01</td>
</tr>
<tr>
<td>see the abbot</td>
<td>0</td>
<td>0.01</td>
</tr>
<tr>
<td>see the abduct</td>
<td>0</td>
<td>0.01</td>
</tr>
<tr>
<td>see the above</td>
<td>2</td>
<td>2.01</td>
</tr>
<tr>
<td>see the Abram</td>
<td>0</td>
<td>0.01</td>
</tr>
<tr>
<td>...</td>
<td>0</td>
<td>0.01</td>
</tr>
<tr>
<td>see the zygote</td>
<td>0</td>
<td>0.01</td>
</tr>
</tbody>
</table>
**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 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 how you chose your vocabulary, you’re still going to have out of vocabulary (OOV) words**

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 (<!--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(fast | “I <UNK>”)

During testing, similarly replace all OOV with <UNK>
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>Word</th>
<th>Count</th>
<th>Probability</th>
<th>Unsmoothed (MLE)</th>
<th>add-lambda smoothing</th>
</tr>
</thead>
<tbody>
<tr>
<td>see the abacus</td>
<td>1</td>
<td>1/3</td>
<td>1.01</td>
<td>1.01/203</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 abduct</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 Abram</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>Total</td>
<td>3</td>
<td>3/3</td>
<td>203</td>
<td></td>
</tr>
</tbody>
</table>

For those we’ve seen before:

Unsmoothed (MLE)  
\[ P(c|ab) = \frac{C(abc)}{C(ab)} \]

add-lambda smoothing  
\[ P(c|ab) = \frac{C(abc) + \lambda}{C(ab) + \lambda} \]

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:

Unsmoothed (MLE)   add-lambda smoothing
-------------------   -----------------------------------
P(c | ab) = \frac{C(abc)}{C(ab)}   P(c | ab) = \frac{C(abc) + \lambda}{C(ab) + \lambda V}

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
<th>Count</th>
<th>Count</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>see the abacus</td>
<td>1</td>
<td>1/3</td>
<td>1/3</td>
<td>1.01</td>
</tr>
<tr>
<td>see the abbot</td>
<td>0</td>
<td>0/3</td>
<td>0/3</td>
<td>0.01</td>
</tr>
<tr>
<td>see the abduct</td>
<td>0</td>
<td>0/3</td>
<td>0/3</td>
<td>0.01</td>
</tr>
<tr>
<td>see the above</td>
<td>2</td>
<td>2/3</td>
<td>2/3</td>
<td>2.01</td>
</tr>
<tr>
<td>see the abduct</td>
<td>0</td>
<td>0/3</td>
<td>0/3</td>
<td>0.01</td>
</tr>
<tr>
<td>see the abduct</td>
<td>0</td>
<td>0/3</td>
<td>0/3</td>
<td>0.01</td>
</tr>
<tr>
<td>see the abduct</td>
<td>0</td>
<td>0/3</td>
<td>0/3</td>
<td>0.01</td>
</tr>
<tr>
<td>Total</td>
<td>3</td>
<td>3/3</td>
<td>3/3</td>
<td>203</td>
</tr>
</tbody>
</table>

Problems with frequency based smoothing

The following bigrams have never been seen:

p(X | San)  \quad p(X | 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:

San  Diego  Francisco  Luis  Jose  Marcos  ate  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 \( w_k \).

Problems with frequency based smoothing

The following trigrams have never been seen:

| p(car | see the) | p(zygote | see the) | p(cumquat | see the) |
|--------|------------|-------------|

Which would add-lambda pick as most likely?

Witten-Bell?

Which would you pick?

Better smoothing approaches

Utilize information in lower-order models

Interpolation

\[
\text{Combine probabilities of lower-order models in some linear combination}
\]

Backoff

\[
P(z \mid xy) = \begin{cases} 
\frac{C^*(xyz)}{C(xy)} & \text{if } C(xy) > k \\
\frac{C^*(y)P(z \mid y)}{C^*(y)} & \text{otherwise}
\end{cases}
\]

- 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

\[
P(z \mid xy) \approx \frac{\lambda C(xy)C(z)}{C(xy)} + \mu \frac{C(y)C(z)}{C(y)} + (1 - \lambda - \mu) \frac{C(z)}{C(*)}
\]

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.

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

Backoff models: absolute discounting

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

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?

What is $\alpha(xy)$?
Backoff models: absolute discounting

Trigram model \( p(z|xy) \) (before discounting)
Trigram model \( p(z|xy) \) (after discounting)
Bigram model \( p(z|y) \)

"For \( z \) where \( xyz \) didn’t occur"

\[ P_{\text{absolute}}(z|xy) = C(\text{xyz}) - D \frac{C(\text{xy})}{\alpha(\text{xy})P_{\text{absolute}}(z|y)} \]

\( \text{if } C(\text{xyz}) > 0 \)

\( \alpha(\text{xy})P_{\text{absolute}}(z|y) \) otherwise

---

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

\( p(\text{cat} | \text{see the}) = ? \)
\( p(\text{puppy} | \text{see the}) = ? \)

\( P_{\text{absolute}}(z|1y) = \)
\( \frac{C(\text{xyz}) - D}{C(\text{xy}) \alpha(\text{xy})P_{\text{absolute}}(z|1y)} \text{ otherwise} \)

---

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

\( P_{\text{absolute}}(z|1y) = \)
\( \frac{C(\text{xyz}) - D}{C(\text{xy}) \alpha(\text{xy})P_{\text{absolute}}(z|1y)} \text{ otherwise} \)

---

```
P_{\text{absolute}}(z|1y) = \)
\( \frac{C(\text{xyz}) - D}{C(\text{xy}) \alpha(\text{xy})P_{\text{absolute}}(z|1y)} \text{ otherwise} \)
```
Backoff models: absolute discounting

| see the dog | 1 | p( puppy | see the ) = ? |
| see the cat | 2 | a(see the) = ? |
| see the banana | 4 | # of types starting with “see the” * D |
| see the man | 1 | count(“see the”) |
| see the woman | 1 | |
| see the car | 1 | |

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

\[ P_{\text{absolute}}(z | xy) = C(xyz) - D \frac{C(xy)}{C(see the)} \] if \( C(xyz) > 0 \)

\[ \alpha(see the) = \frac{\text{reserved \_ mass}(see the)}{\sum_{X \in \text{see the X} > 0} p(X | \text{the})} \]

Calculating \( \alpha \)

We can calculate \( \alpha \) two ways:

- Based on those we haven’t seen:
  \[ \alpha(\text{see the}) = \frac{\text{reserved \_ mass}(\text{see the})}{\sum_{X \in \text{see the X} > 0} p(X | \text{the})} \]

- Or, more often, based on those we do see:
  \[ \alpha(\text{see the}) = \frac{\text{reserved \_ mass}(\text{see the})}{1 - \sum_{X \in \text{see the X} > 0} p(X | \text{the})} \]
Calculating $\alpha$ in general: trigrams

$$p( C \mid A B)$$

$\text{Calculate the reserved mass}$

$$\text{reserved\_mass}(\text{bigram}--A B) = \frac{\# \text{ of types starting with bigram } \times D}{\text{count(bigram)}}$$

$\text{Calculate the sum of the backed off probability. For bigram "A B"}$

$$1 - \sum_{X \in C \mid A X > 0} p(X) \quad \text{either is fine, in practice the left is easier} \quad \sum_{X \in C \mid A X > 0} p(X)$$

$$\text{Calculate } \alpha$$

$$\alpha(A B) = \frac{\text{reserved\_mass}(A B)}{1 - \sum_{X \in C \mid A X > 0} p(X)}$$

$$1 - \text{the sum of the bigram probabilities of those trigrams that we saw starting with bigram A B}$$

Calculating $\alpha$ in general: bigrams

$$p( B \mid A)$$

$\text{Calculate the reserved mass}$

$$\text{reserved\_mass}(\text{unigram}--A) = \frac{\# \text{ of types starting with unigram } \times D}{\text{count(unigram)}}$$

$\text{Calculate the sum of the backed off probability. For bigram "A B"}$

$$1 - \sum_{X \in C \mid A X > 0} p(X) \quad \text{either is fine in practice, the left is easier} \quad \sum_{X \in C \mid A X > 0} p(X)$$

$$\text{Calculate } \alpha$$

$$\alpha(A) = \frac{\text{reserved\_mass}(A)}{1 - \sum_{X \in C \mid A X > 0} p(X)}$$

$$1 - \text{the sum of the unigram probabilities of those bigrams that we saw starting with word A}$$

Calculating backoff models in practice

$\text{Store the } \alpha$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

$\text{Compute the } \alpha$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( jumped | the Dow ) = ?
the Dow rose  5  What is the reserved mass?
the Dow fell  5
```

$\text{Compute the } \alpha$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

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

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