LANGUAGE MODELING: SMOOTHING

Admin

- Assignment 2
  - bigram language modelling
  - Java
  - Can work with partners
    - Anyone looking for a partner?
  - 2a: Due this Thursday
  - 2b: Due next Wednesday
  - 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

- Lab next class

Today

- Same time, but will be an interactive session

smoothing techniques
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 characteristics of your data!

Smoothing

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

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

If any of these has never been seen before, \( \text{prob} = 0! \)

Smoothing can help reduce some of the noise.

The general smoothing problem

<table>
<thead>
<tr>
<th>Event</th>
<th>Probability modification</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>see the abacus</td>
<td>1 1/3</td>
<td>?</td>
</tr>
<tr>
<td>see the abbot</td>
<td>0 0/3</td>
<td>?</td>
</tr>
<tr>
<td>see the abduct</td>
<td>0 0/3</td>
<td>?</td>
</tr>
<tr>
<td>see the above</td>
<td>2 2/3</td>
<td>?</td>
</tr>
<tr>
<td>see the Abram</td>
<td>0 0/3</td>
<td>?</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>?</td>
</tr>
<tr>
<td>see the zygote</td>
<td>0 0/3</td>
<td>?</td>
</tr>
<tr>
<td>Total</td>
<td>3 3/3</td>
<td>?</td>
</tr>
</tbody>
</table>
Add-lambda smoothing

A large dictionary makes novel events too probable.

add $\lambda = 0.01$ to all counts

<table>
<thead>
<tr>
<th>Event</th>
<th>Count</th>
<th>Probability</th>
<th>Smoothed Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>see the abacus</td>
<td>1</td>
<td>1/3</td>
<td>1.01</td>
</tr>
<tr>
<td>see the abbot</td>
<td>0</td>
<td>0/3</td>
<td>0.01</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 zygote</td>
<td>0</td>
<td>0/3</td>
<td>0.01</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>3</strong></td>
<td><strong>3/3</strong></td>
<td><strong>203</strong></td>
</tr>
</tbody>
</table>

How should we pick lambda?

<table>
<thead>
<tr>
<th>Event</th>
<th>Count</th>
<th>Probability</th>
<th>Smoothed Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>see the abacus</td>
<td>1</td>
<td>1/3</td>
<td>1.01</td>
</tr>
<tr>
<td>see the abbot</td>
<td>0</td>
<td>0/3</td>
<td>0.01</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 zygote</td>
<td>0</td>
<td>0/3</td>
<td>0.01</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>3</strong></td>
<td><strong>3/3</strong></td>
<td><strong>203</strong></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?
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 (or it’s not defined)
- Smoothing doesn’t really help us with this!

To make this explicit, smoothing helps us with…

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></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>...</td>
<td>...</td>
</tr>
<tr>
<td>see the zygote</td>
<td>0</td>
<td>0.01</td>
</tr>
</tbody>
</table>

 Counts

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></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>

Smoothed counts

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 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

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

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>
  - \( p(<\text{UNK}>) \), \( p(\text{fast} | <\text{UNK}>) \)
  - \( p(\text{i} | <\text{UNK}>) \)

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></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 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>...</td>
<td></td>
<td></td>
<td></td>
<td></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>
</tbody>
</table>

Total: 3 | 3/3 | 203 |
Storing the table

Hashtable (e.g. HashMap)
- fast retrieval
- fairly good memory usage

Only store those entries of things we’ve seen
- for example, we don’t store $|V|^3$ 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:

Unsmoothed (MLE) vs. add-lambda smoothing

$$P(c \mid ab) = \frac{C(abc)}{C(ab)}$$

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

$$P(c \mid ab) = \frac{C(abc) + \lambda}{C(ab) + \lambda V}$$

Storing the table: add-lambda smoothing

For those we’ve seen before:

Unsmoothed (MLE) vs. add-lambda smoothing

$$P(c \mid ab) = \frac{C(abc)}{C(ab)}$$

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

$$P(c \mid ab) = \frac{\lambda}{C(ab) + \lambda V}$$
Problems with frequency based smoothing

The following bigrams have never been seen:

- p( X | San )
- 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
- 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 \( w_k \).

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
- Combine probabilities of lower-order models in some linear combination

Backoff
- 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

$$P(z | xy) = \frac{C(xy)}{C(y)} P(z | y)$$

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

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

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

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

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 \( \alpha(xy) \)?

| Trigram model: \( p(z|xy) \) | Trigram model: \( p(z|xy) \) (after discounting) | Bigram model: \( p(z|y) \) * | Unseen words |
|---|---|---|---|
| \[ \text{before discounting} \] | \[ \text{after discounting} \] | \[ \text{for } z \text{ where } xyz \text{ didn't occur} \] | \[ \text{xyz occurred} \] |
| see the dog | 1 | see the cat | 2 |
| see the banana | 4 | see the man | 1 |
| see the woman | 1 | see the car | 1 |

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

\[ P_{\text{absolute}}(z|y) = \frac{C(xy)}{a(xy)P_{\text{absolute}}(z|y)} \quad \text{otherwise} \]
### Backoff models: absolute discounting

<table>
<thead>
<tr>
<th>Event</th>
<th>Count</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>see the dog</td>
<td>1</td>
<td>$p(\text{cat} \mid \text{see the}) = ?$</td>
</tr>
<tr>
<td>see the cat</td>
<td>2</td>
<td>$p(\text{puppy} \mid \text{see the}) = ?$</td>
</tr>
<tr>
<td>see the banana</td>
<td>4</td>
<td>$a(\text{see the}) = ?$</td>
</tr>
<tr>
<td>see the man</td>
<td>1</td>
<td>$\frac{2-D}{10} = \frac{2-0.75}{10} = .125$</td>
</tr>
<tr>
<td>see the woman</td>
<td>1</td>
<td>$\alpha(\text{see the}) = ?$</td>
</tr>
<tr>
<td>see the car</td>
<td>1</td>
<td>$\frac{6 \times D}{10} = \frac{6 	imes 0.75}{10} = 0.45$</td>
</tr>
</tbody>
</table>

For each of the unique trigrams, we subtracted $D/\text{count(“see the”)}$ from the probability distribution and distributed this probability mass to all bigrams that we are backing off to.
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: $P(X \mid \text{the})$.

$\alpha$ is the normalizing constant so that the sum of these probabilities equals the reserved probability mass $\alpha(\text{see the}) = \text{reserved mass(see the)}$.

$\alpha(\text{see the}) \sum_{X : C(\text{see the } X) = 0} P(X \mid \text{the}) = \text{reserved mass(see the)}$

Calculating $\alpha$ in general: trigrams

$p(\text{C} \mid A \ B)$

Calculate the reserved mass

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

Calculate the sum of the backed off probability. For bigram “A B”,

$1 - \sum_{X : C(A \ B \ X) > 0} p(X \mid B)$

either is fine in practice, the left is easier

$\sum_{X : C(A \ B \ X) = 0} p(X \mid B)$

Calculate $\alpha$

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

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

Calculating $\alpha$ in general: bigrams

$p(\text{B} \mid A)$

Calculate the reserved mass

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

Calculate the sum of the backed off probability. For bigram “A”:

$1 - \sum_{X : C(A \ X) > 0} p(X)$

either is fine in practice, the left is easier

$\sum_{X : C(A \ X) = 0} p(X)$

Calculate $\alpha$

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

$1 = \text{the sum of the unigram probabilities of those bigrams that we saw starting with word A}$
Calculating backoff models in practice

- Store the αs 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 αs during training
  - After calculating all of the probabilities of seen unigrams/bigrams/trigrams
  - Go back through and calculate the αs (you should have all of the information you need)

During testing, it should then be easy to apply the backoff model with the αs pre-calculated

Backoff models: absolute discounting

where:
- 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