Evaluating N-Gram Models

- Best evaluation for a language model
  - **Extrinsic** evaluation
    - Put model A into an application
      - For example, a speech recognizer
    - Evaluate the performance of the application with model A
    - Put model B into the application and evaluate
    - Compare performance of the application with the two models
Difficulty of extrinsic (in-vivo) evaluation of N-gram models

- **Extrinsic evaluation**
  - This is really time-consuming
  - Can take days to run an experiment

- **So**
  - As a temporary solution, in order to run experiments
  - To evaluate N-grams we often use an intrinsic evaluation, an approximation called perplexity
  - But perplexity is a poor approximation unless the test data looks just like the training data
  - So is generally only useful in pilot experiments (generally is not sufficient to publish)

- But is helpful to think about.
Evaluation

- Standard method
  - Train parameters of our model on a training set.
  - Look at the models performance on some new data
    - This is exactly what happens in the real world; we want to know how our model performs on data we haven’t seen
  - So use a test set. A dataset which is different than our training set, but is drawn from the same source
  - Then we need an evaluation metric to tell us how well our model is doing on the test set.
    - One such metric is **perplexity**
Intuition of Perplexity

- The Shannon Game:
  - How well can we predict the next word?
    - I always order pizza with cheese and ____
    - The 33rd President of the US was ____
    - I saw a ____
  - Unigrams are terrible at this game. (Why?)

- A better model of a text
  - is one which assigns a higher probability to the word that actually occurs

- Ask a speech recognizer to recognize digits: “0, 1, 2, 3, 4, 5, 6, 7, 8, 9” – easy – perplexity 10

- Perplexity is weighted equivalent branching factor.
Shannon’s Method

- Assigning probabilities to sentences is all well and good, but it’s not terribly illuminating. A more interesting task is to turn the model around and use it to generate random sentences that are like the sentences from which the model was derived.

- Generally attributed to Claude Shannon.
Generating Shakespeare

- Unigrams
  - To him swallowed confess hear both. Which. Of save on trial for are ay device and rote life have c
  - Hill he late speaks; or! A more or legless first you enter

- Bigrams
  - What means, sir. I confess she? Then all sorts, he is trim, captain.
  - Why doest stand forth they canopy, forsooth he is this palpable hit the King Henry. Live king. Follow.

- Trigrams
  - Sweet prince, Falstaff shall die. Harry of Monmouths grave
  - This shall forbid it should be branded, if renown made it empty

- Quadrigrams
  - King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv’d in;
  - Willyou not tell me who I am?
  - It cannot be but so.
Perplexity: Math

- Perplexity is geometric
- Average inverse probability
  - $N = \text{Normalized by length of the sentence}$
- Lower perplexity means a better model

\[
PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}
\]
More complex example

- Ask a speech recognizer to recognize names at Microsoft – hard – 30,000 – perplexity 30,000

- Imagine model:
  - “Operator” (1 in 4),
  - “Technical support” (1 in 4),
  - “sales” (1 in 4),
  - 30,000 names (1 in 120,000)

- Perplexity 53
Perplexity and Probability

- Minimizing perplexity is the same as maximizing probability
  - Higher probability means lower Perplexity
  - The more information, the lower perplexity
  - Lower perplexity means a better model
  - The lower the perplexity, the closer we are to the true model.

- Training 38 million words, test 1.5 million words, WSJ

<table>
<thead>
<tr>
<th>N-gram Order</th>
<th>Unigram</th>
<th>Bigram</th>
<th>Trigram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perplexity</td>
<td>962</td>
<td>170</td>
<td>109</td>
</tr>
</tbody>
</table>

- The best language model is one that best predicts an unseen test set
Perplexity:  Is lower really better?

- Remarkable fact: the true model for data has the lowest possible perplexity
- Lower the perplexity, the closer we are to true model
- Typically, perplexity correlates well with speech recognition word error rate
  - Correlates better when both models are trained on same data
  - Doesn’t correlate well when training data changes
- However, does not take into account acoustic difficulty
  - “A, B, C, D, E, F, G…Z”: perplexity is 26
  - “Alpha, bravo, charlie, delta…yankee, zulu”: perplexity is 26
### Functionalities of SRILM

**Three Main Functionalities**

1. Generate the n-gram count file from the corpus
2. Train the language model from the n-gram count file
3. Calculate the test data perplexity using the trained language model
What to do about Zero Counts

Back to Shakespeare

- Recall that Shakespeare produced 300,000 bigram types out of $V^2 = 844$ million possible bigrams...
- So, 99.96% of the possible bigrams were never seen (have zero entries in the table)
- Does that mean that any sentence that contains one of those bigrams should have a probability of 0?
Zipf’s Law

- Given the frequency $f$ of a word and its rank $r$ in the list of words ordered by their frequencies:

$$f \propto \frac{1}{r} \quad \text{or} \quad f \times r = k \text{ for a constant } k$$
Sparse Data Problem

- MLE is in general unsuitable for statistical inference in NLP because small parameters are hard to estimate.

- The problem is the sparseness of our data (even with the large corpus).
  - The vast majority of words are very uncommon
  - longer *n*-grams involving them are thus much rarer

- The MLE assigns a zero probability to unseen events
  - **Bad** ...because the probability of the whole sequences will be zero
    - computed by multiplying the probabilities of subparts
Solution

- **How do you handle unseen n-grams?**
  - Smoothing
    - Use some of the probability mass to cover unseen events
  - Backoff
    - Use counts from a smaller context
  - Interpolation
    - Combine multiple sources of information appropriately weighted

- **Try to differentiate cases**
  - Some of those zeros are really zeros...
    - Things that really can’t or shouldn’t happen.
  - Some of them are just rare events.
    - If the training corpus had been a little bigger they would have
      had a count (probably a count of 1!).
When we have sparse statistics:

\[ P(w \mid \text{denied the}) \]
3 allegations  
2 reports  
1 claims  
1 request  
7 total

Steal probability mass to generalize better

\[ P(w \mid \text{denied the}) \]
2.5 allegations  
1.5 reports  
0.5 claims  
0.5 request  
2 other  
7 total
Laplace Smoothing

- Also called add-one smoothing
- Just add one to all the counts!
- Very simple

**MLE estimate:**
\[
P(w_i) = \frac{c_i}{N}
\]

**Laplace estimate:**
\[
P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}
\]

**Reconstructed counts:**
\[
c_i^* = (c_i + 1) \frac{N}{N + V}
\]

- \(c_i\): Counts for word \(i\)
- \(N\): Number of words
- \(V\): Size of the vocabulary
### Unigram Smoothing Example

- Tiny Corpus, \( V = 4; N = 20 \)

\[
P_{\text{un}}(w_i) = \frac{c_i + 1}{N + V}
\]

<table>
<thead>
<tr>
<th>Word</th>
<th>True Ct</th>
<th>Unigram Prob</th>
<th>New Ct</th>
<th>Adjusted Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>eat</td>
<td>10</td>
<td>.5</td>
<td>11</td>
<td>.46</td>
</tr>
<tr>
<td>British</td>
<td>4</td>
<td>.2</td>
<td>5</td>
<td>.21</td>
</tr>
<tr>
<td>food</td>
<td>6</td>
<td>.3</td>
<td>7</td>
<td>.29</td>
</tr>
<tr>
<td>happily</td>
<td>0</td>
<td>.0</td>
<td>1</td>
<td>.04</td>
</tr>
<tr>
<td>20</td>
<td>1.0</td>
<td>~20</td>
<td>1.0</td>
<td></td>
</tr>
</tbody>
</table>
Reconstituted Counts From Berkeley

\[
c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n) + 1] \times C(w_{n-1})}{C(w_{n-1}) + V}
\]

<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>15</td>
<td>0.34</td>
</tr>
</tbody>
</table>

<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>
Big Change to the Counts!

- C(want to) went from 608 to 238!
- P(to|want) from .66 to .26!
- Discount \( d = \frac{c^*}{c} \)
  - \( d \) for “chinese food” = .10!!! A 10x reduction
  - So in general, Laplace is a blunt instrument
  - Could use more fine-grained method (add-k)
- But Laplace smoothing not used for N-grams, as we have much better methods
- Despite its flaws Laplace (add-k) is however still used to smooth other probabilistic models in NLP, especially
  - For pilot studies
  - in domains where the number of zeros isn’t so huge.
Problem with Laplace Smoothing

- Problem: give too much probability mass to unseen n-grams.
- For sparse sets of data over large vocabularies, such as n-grams, Laplace's law actually gives far too much of the probability space to unseen events.
- Can we smooth more usefully?
Additive vs. Discounting approaches

- LaPlace is additive (add 1 to everything)
  - Can be improved by adding to everything

- Discounting (absolute discounting)
  - Subtracts $\varepsilon$ from everything
  - Distributes $\varepsilon$ across the unseen event
Intuition for Absolute Discounting

- Bigrams from AP Newswire corpus (Church & Gale, 1991)
- It turns out, 5 4.22 after all the calculation,
  - $c^* \approx c - D$
  - where $D = .75$
- Combine this with Back-off (interpolation is also possible)

<table>
<thead>
<tr>
<th>$C_{\text{unsmoothed}}$</th>
<th>$C^*_{\text{(GT)}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>.000027</td>
</tr>
<tr>
<td>1</td>
<td>.446</td>
</tr>
<tr>
<td>2</td>
<td>1.26</td>
</tr>
<tr>
<td>3</td>
<td>2.24</td>
</tr>
<tr>
<td>4</td>
<td>3.24</td>
</tr>
<tr>
<td>5</td>
<td>4.22</td>
</tr>
<tr>
<td>6</td>
<td>5.19</td>
</tr>
<tr>
<td>7</td>
<td>6.21</td>
</tr>
<tr>
<td>8</td>
<td>7.24</td>
</tr>
<tr>
<td>9</td>
<td>8.25</td>
</tr>
</tbody>
</table>
Better Smoothing

- Intuition used by many smoothing algorithms
  - Good-Turing
  - Kneser-Ney
  - Witten-Bell

- Is to use the count of things we’ve seen once to help estimate the count of things we’ve never seen
Imagine you are fishing
- There are 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)?
- 3/18

Assuming so, how likely is it that next species is trout?
- Must be less than 1/18
Good-Turing

- Notation: $N_x$ is the frequency-of-frequency-$x$
  - So $N_{10}=1$
    - Number of fish species seen 10 times is 1 (carp)
  - $N_1=3$
    - Number of fish species seen 1 is 3 (trout, salmon, eel)

- To estimate total number of unseen species
  - Use number of species (words) we’ve seen once
  - $c_0^* = c_1$, $p_0 = N_1/N$

- All other estimates are adjusted (down) to give probabilities for unseen

$$c^* = (c + 1) \frac{N_{c+1}}{N_c}$$
Could just spread 1s over 0s

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>10</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carp</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perch</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>WF</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Trout</td>
<td>1</td>
<td>1</td>
<td>.6</td>
</tr>
<tr>
<td>Salmon</td>
<td>1</td>
<td>1</td>
<td>.6</td>
</tr>
<tr>
<td>Eel</td>
<td>1</td>
<td>1</td>
<td>.6</td>
</tr>
<tr>
<td>Catfish</td>
<td>0</td>
<td>1</td>
<td>.6</td>
</tr>
<tr>
<td>Bass</td>
<td>0</td>
<td>1</td>
<td>.6</td>
</tr>
<tr>
<td>TOTAL</td>
<td>18</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Prob of things that occurred once
  \[ \frac{1}{18} + \frac{1}{18} + \frac{1}{18} = \frac{3}{18} \]
- Add one to zero counts
- Spread probability over 1s and 0s
  \[ \frac{3/18}{5} = .033 \]
GT Smoothed Bigram Probabilities

<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>0.0014</td>
<td>0.326</td>
<td>0.00248</td>
<td>0.00355</td>
<td>0.000205</td>
<td>0.0017</td>
<td>0.00073</td>
<td>0.000489</td>
</tr>
<tr>
<td>want</td>
<td>0.00134</td>
<td>0.00152</td>
<td>0.656</td>
<td>0.000483</td>
<td>0.00455</td>
<td>0.00455</td>
<td>0.00384</td>
<td>0.000483</td>
</tr>
<tr>
<td>to</td>
<td>0.000512</td>
<td>0.00152</td>
<td>0.00165</td>
<td>0.284</td>
<td>0.000512</td>
<td>0.0017</td>
<td>0.00175</td>
<td>0.0873</td>
</tr>
<tr>
<td>eat</td>
<td>0.00101</td>
<td>0.00152</td>
<td>0.00166</td>
<td>0.00189</td>
<td>0.0214</td>
<td>0.00166</td>
<td>0.0563</td>
<td>0.000585</td>
</tr>
<tr>
<td>chinese</td>
<td>0.00283</td>
<td>0.00152</td>
<td>0.00248</td>
<td>0.00189</td>
<td>0.000205</td>
<td>0.519</td>
<td>0.00283</td>
<td>0.000585</td>
</tr>
<tr>
<td>food</td>
<td>0.0137</td>
<td>0.00152</td>
<td>0.0137</td>
<td>0.00189</td>
<td>0.000409</td>
<td>0.00366</td>
<td>0.00073</td>
<td>0.000585</td>
</tr>
<tr>
<td>lunch</td>
<td>0.00363</td>
<td>0.00152</td>
<td>0.00248</td>
<td>0.00189</td>
<td>0.000205</td>
<td>0.00131</td>
<td>0.00073</td>
<td>0.000585</td>
</tr>
<tr>
<td>spend</td>
<td>0.00161</td>
<td>0.00152</td>
<td>0.00161</td>
<td>0.00189</td>
<td>0.000205</td>
<td>0.0017</td>
<td>0.00073</td>
<td>0.000585</td>
</tr>
</tbody>
</table>

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>0.0083</td>
<td>0</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>0.027</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>0.0029</td>
<td>0.0027</td>
<td>0.0063</td>
<td>0</td>
</tr>
<tr>
<td>food</td>
<td>0.014</td>
<td>0</td>
<td>0.014</td>
<td>0</td>
<td>0.0092</td>
<td>0.027</td>
<td>0.056</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>0.0029</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>
### Bigram Frequencies of Frequencies and GT Re-estimates

<table>
<thead>
<tr>
<th></th>
<th>AP Newswire</th>
<th>Berkeley Restaurant —</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>c (MLE)</strong></td>
<td><strong>$N_c$</strong></td>
<td><em><em>c</em> (GT)</em>*</td>
</tr>
<tr>
<td>0</td>
<td>74,671,100,000</td>
<td>0.0000270</td>
</tr>
<tr>
<td>1</td>
<td>2,018,046</td>
<td>0.446</td>
</tr>
<tr>
<td>2</td>
<td>449,721</td>
<td>1.26</td>
</tr>
<tr>
<td>3</td>
<td>188,933</td>
<td>2.24</td>
</tr>
<tr>
<td>4</td>
<td>105,668</td>
<td>3.24</td>
</tr>
<tr>
<td>5</td>
<td>68,379</td>
<td>4.22</td>
</tr>
<tr>
<td>6</td>
<td>48,190</td>
<td>5.19</td>
</tr>
</tbody>
</table>
Practical considerations for Good-Turing

- The new estimations of c (c*) are dependent on the counts of c+1
  - But some can be 0
    - First use linear regression to smooth counts
  - Higher counts are more likely to be right
    - Only apply this to counts <=5
  - Low counts can be noise
    - Treat counts of 1 as if they were 0

- Good-Turing is never used alone—always combined with back-off and interpolation
Backoff and Interpolation

■ Smaller context can be a useful source of knowledge

■ If we are estimating:
  ■ trigram $p(z|x,y)$
  ■ but count(xyz) is zero

■ Use info from:
  ■ Bigram $p(z|y)$

■ Or even:
  ■ Unigram $p(z)$

■ How to combine this trigram, bigram, unigram info in a valid fashion?
Backoff Vs. Interpolation

**Backoff:**
- If you don’t have the full context, use partial context
- use trigram if you have it, otherwise bigram, otherwise unigram

**Interpolation:**
- Use multiple sources of information
- Weight them based on how much information they contribute
  - More context is weighted higher
  - Better match (higher perplexity) is weighted higher
**Interpolation**

- Simple interpolation

\[ \hat{P}(w_n|w_{n-1}w_{n-2}) = \lambda_1 P(w_n|w_{n-1}w_{n-2}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n) \]

\[ \sum_i \lambda_i = 1 \]

- Lambdas conditional on context:

\[ \hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1 (w_{n-2}^{n-1})P(w_n|w_{n-2}w_{n-1}) + \lambda_2 (w_{n-2}^{n-1})P(w_n|w_{n-1}) + \lambda_3 (w_{n-2}^{n-1})P(w_n) \]
How to Set the Lambdas?

- Use a held-out, or development, corpus

- Choose lambdas which maximize the probability on dev
  - i.e. fix the $N$-gram probabilities
  - Then search for lambda values
  - That when plugged into previous equation
  - Give largest probability for held-out set
  - Can use EM to do this search
Katz Backoff N-gram model

- If we’ve seen the n-gram, use it
  - But “discount it” by a normalizing factor
  - Have to account for “borrowing” for other unseen n-grams

- Otherwise: Recursively back off the the (N-1)-gram until there are some counts

Thanks to Dan Jurafsky for these slides
Katz Backoff

\[
P_{\text{katz}}(w_n|w_{n-N+1}^{n-1}) = \begin{cases} 
  P^*(w_n|w_{n-N+1}^{n-1}), & \text{if } C(w_{n-N+1}^n) > 0 \\
  \alpha(w_{n-N+1}^{n-1})P_{\text{katz}}(w_n|w_{n-N+2}^{n-1}), & \text{otherwise.}
\end{cases}
\]

\[
P_{\text{katz}}(z|x,y) = \begin{cases} 
  P^*(z|x,y), & \text{if } C(x,y,z) > 0 \\
  \alpha(x,y)P_{\text{katz}}(z|y), & \text{else if } C(x,y) > 0 \\
  P^*(z), & \text{otherwise.}
\end{cases}
\]

\[
P_{\text{katz}}(z|y) = \begin{cases} 
  P^*(z|y), & \text{if } C(y,z) > 0 \\
  \alpha(y)P^*(z), & \text{otherwise.}
\end{cases}
\]
Why discounts $P^*$ and alpha?

- MLE probabilities sum to 1

$$\sum_i P(w_i | w_j w_k) = 1$$

- So if we used MLE probabilities but backed off to lower order model when MLE prob is zero

- We would be adding extra probability mass

- And total probability would be greater than 1

$$P^*(w_n | w_{n-N+1}^n) = \frac{c^*(w_{n-N+1}^n)}{c(w_{n-N+1}^{n-1})}$$
Intuition of Backoff+Discounting

- How much probability to assign to all the zero trigrams?
  - Use GT or other discounting algorithm to tell us

- How to divide that probability mass among different contexts?
  - Use the N-1 gram estimates to tell us

- What do we do for the unigram words not seen in training?
  - Out Of Vocabulary = OOV words
Google NGrams

All Our N-gram are Belong to You
By Peter Norvig - 8/03/2006 11:26:00 AM

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word n-gram models for a variety of R&D projects, such as statistical machine translation, speech recognition, spelling correction, entity detection, information extraction, and others. While such models have usually been estimated from training...

File sizes: approx. 24 GB compressed (gzip'ed) text files

Number of tokens: 1,024,908,267,229
Number of sentences: 95,119,665,584
Number of unigrams: 13,588,391
Number of bigrams: 314,843,401
Number of trigrams: 977,069,902
Number of fourgrams: 1,313,818,354
Number of fivegrams: 1,176,470,663

http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html
Smoothing: Kneser-Ney

P(Francisco | eggplant) vs P(stew | eggplant)

- “Francisco” is common, so backoff & interpolated methods say it is likely
- But it only occurs in context of “San”
- “Stew” is common, and in many contexts

- Weight backoff by number of contexts word occurs in
  - $C = \text{number of different Contexts}$
  - $D = \text{absolute discount (see textbook)}$

$$P_{IKN}(w_i | w_{i-1}) = \frac{C(w_{i-1}w_i) - D}{C(w_{i-1})} + \beta(w_i) \frac{| \{ w_{i-1} : C(w_{i-1}w_i) > 0 \} |}{\sum_{w_i} | \{ w_{i-1} : C(w_{i-1}w_i) > 0 \} |}$$
Kneser-Ney smoothing

- Still state-of-the-art n-gram LM
- Absolute discounting

<table>
<thead>
<tr>
<th>Bigram count in training set</th>
<th>Bigram count in heldout set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0000270</td>
</tr>
<tr>
<td>1</td>
<td>0.448</td>
</tr>
<tr>
<td>2</td>
<td>1.25</td>
</tr>
<tr>
<td>3</td>
<td>2.24</td>
</tr>
<tr>
<td>4</td>
<td>3.23</td>
</tr>
<tr>
<td>5</td>
<td>4.21</td>
</tr>
<tr>
<td>6</td>
<td>5.23</td>
</tr>
<tr>
<td>7</td>
<td>6.21</td>
</tr>
<tr>
<td>8</td>
<td>7.21</td>
</tr>
<tr>
<td>9</td>
<td>8.26</td>
</tr>
</tbody>
</table>

\[
P_{\text{AbsoluteDiscounting}}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) - d}{C(w_{i-1})} + \lambda(w_{i-1})P(w_i)
\]
Kneser-Ney smoothing

I’m going to San __________

I can’t see without my reading _____.

- Francisco ONLY follows San
- glasses follows many words
- Probability that Francisco is in other bigrams

\[ P_{\text{CONTINUATION}}(w_i) = \frac{|\{w_{i-1} : c(w_{i-1}w_i) > 0\}|}{|\{(w_{j-1}, w_j) : c(w_{j-1}w_j) > 0\}|} \]

\[ P_{\text{KN}}(w_i|w_{i-1}) = \frac{\max(c(w_{i-1}w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1})P_{\text{CONTINUATION}}(w_i) \]
Out Of Vocabulary = OOV words

We don’t use GT smoothing for these
  Because GT assumes we know the number of unseen events

Instead: create an unknown word token <UNK>
  Training of <UNK> probabilities
    Create a fixed lexicon L of size V
    At text normalization phase, any training word not in L changed to <UNK>
    Now we train its probabilities like a normal word

At decoding time
  If text input: Use UNK probabilities for any word not in training
+ Practical Issues

- We do everything in log space
  - Avoid underflow
  - (also adding is faster than multiplying)

\[ p_1 \times p_2 \times p_3 \times p_4 = \exp(\log p_1 + \log p_2 + \log p_3 + \log p_4) \]