Probabilistic Language Models

- Assign a probability to a sentence
- Machine Translation:
  - $P(\text{high winds tonight}) > P(\text{large winds tonight})$
- Spell Correction
  - The office is about fifteen minutes from my house
  - $P(\text{about fifteen minutes from}) > P(\text{about fifteen minuets from})$
- Speech Recognition
  - $P(\text{I saw a van}) > P(\text{eyes awe of an})$
- Text Generation in general:
  - Summarization, question-answering ...

Probabilistic Language Modeling

- Goal: compute the probability of a sentence or sequence of words:
  - $P(W) = P(w_1,w_2,w_3,w_4,...,w_n)$
- Related task: probability of an upcoming word:
  - $P(w_{n+1}|w_1,w_2,w_3,...,w_n)$
- A model that computes either of these:
  - $P(W)$ or $P(w_{n+1}|w_1,w_2,...,w_n)$ is called a language model.
- Better: the grammar
- But language model or LM is standard

How to compute $P(W)$

- How to compute this joint probability:
  - $P(\text{its, water, is, so, transparent, that})$
  - Intuition: let’s rely on the Chain Rule of Probability

Quick Review: Probability

- Recall the definition of conditional probabilities
  - $P(B|A) = \frac{P(A,B)}{P(A)}$  
  - Rewriting: $P(A,B) = P(A)P(B|A)$
- More variables:
  - $P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$
- The Chain Rule in General
  - $P(x_1,x_2,x_3,...,x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)...P(x_n|x_1,...,x_{n-1})$
The Chain Rule applied to compute joint probability of words in sentence

\[ P(w_1w_2...w_n) = \prod_i P(w_i \mid w_1w_2...w_{i-1}) \]

\[ P("its water is so transparent") = \\
P(its) \times P(water \mid its) \times P(is \mid its water) \\
\times P(so \mid its water is) \times P(transparent \mid its water is so) \]

How to estimate these probabilities

- Could we just count and divide?

\[ P(\text{the its water is so transparent that}) = \frac{\text{Count(its water is so transparent that)}}{\text{Count(its water is so transparent that) plus other possibilities}} \]

- No! Too many possible sentences!

- We'll never see enough data for estimating these

Markov Assumption

- Simplifying assumption:

\[ P(\text{the its water is so transparent that}) \approx P(\text{the that}) \]

- Or maybe

\[ P(\text{the its water is so transparent that}) \approx P(\text{the transparent that}) \]

Markov Assumption

\[ P(w_1w_2...w_n) = \prod_i P(w_i \mid w_{i-k}...w_{i-1}) \]

• In other words, we approximate each component in the product

\[ P(w_i \mid w_1w_2...w_{i-1}) \approx P(w_i \mid w_{i-k}...w_{i-1}) \]

Simplest case: Unigram model

\[ P(w_1w_2...w_n) = \prod_i P(w_i) \]

Some automatically generated sentences from a unigram model

fifth, an, of, futures, the, an, incorporated, a, a, the, inflation, most, dollars, quarter, in, is, mass, thrift, did, eighty, said, hard, 'm, july, bullish, that, or, limited, the

Bigram model

- Condition on the previous word:

\[ P(w_i \mid w_1w_2...w_{i-1}) = P(w_i \mid w_{i-1}) \]

texaco, rose, one, in, this, issue, is, pursuing, growth, in, a, boiler, house, said, mr., gurria, mexico, 's, motion, control, proposal, without, permission, from, five, hundred, fifty, five, yen, outside, new, car, parking, lot, of, the, agreement, reached, this, would, be, a, record, november
N-gram models

• We can extend to trigrams, 4-grams, 5-grams
• In general this is an insufficient model of language
  • because language has long-distance dependencies:
    “The computer(s) which I had just put into the machine room on the fifth floor is (are) crashing.”
• But we can often get away with N-gram models

Today’s Outline

• Probabilistic language model and n-grams
  • Estimating n-gram probabilities
• Language model evaluation and perplexity
  • Generalization and zeros
• Smoothing: add-one
  • Interpolation, backoff, and web-scale LMs
• Smoothing: Kneser-Ney Smoothing

Estimating bigram probabilities

• The Maximum Likelihood Estimate

\[
P(w_i | w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}
\]

\[
P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}
\]

An example

\[
P(\text{I} | \text{am}) = \frac{c(\text{am}), \text{am}}{c(\text{am})} = 0.67
\]

<table>
<thead>
<tr>
<th>Raw bigram counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{Out of 9222 sentences}</td>
</tr>
</tbody>
</table>
| \begin{tabular}{|c|c|c|c|c|c|c|}
<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>want</td>
<td>2</td>
<td>0</td>
<td>606</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>5</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>
</tr>
<tr>
<td>eat</td>
<td>0</td>
<td>2</td>
<td>16</td>
<td>2</td>
<td>42</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>chinese</td>
<td>1</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>
</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>
</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>
</tr>
</tbody>
</table>

More examples: Berkeley Restaurant Project sentences

• can you tell me about any good cantonese restaurants close by
• mid priced thai food is what i’m looking for
• tell me about chez panisse
• can you give me a listing of the kinds of food that are available
• i’m looking for a good place to eat breakfast
• when is caffe venezia open during the day
Raw bigram probabilities

• Normalize by unigrams:

<table>
<thead>
<tr>
<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>2533</td>
<td>.002</td>
<td>.33</td>
<td>0</td>
<td>0.0065</td>
<td>0</td>
<td>0</td>
<td>0.0079</td>
</tr>
<tr>
<td>927</td>
<td>.0022</td>
<td>.0017</td>
<td>0</td>
<td>0.0021</td>
<td>0.003</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2417</td>
<td>.0083</td>
<td>0</td>
<td>0.28</td>
<td>0.0083</td>
<td>0.0054</td>
<td>0.0011</td>
<td>0.0025</td>
</tr>
<tr>
<td>746</td>
<td>.0063</td>
<td>0</td>
<td>0</td>
<td>0.0021</td>
<td>0.0056</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>158</td>
<td>0</td>
<td>0</td>
<td>0.52</td>
<td>0.0003</td>
<td>.0056</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1093</td>
<td>0</td>
<td>0</td>
<td>0.52</td>
<td>0.0003</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>341</td>
<td>0</td>
<td>0.0036</td>
<td>0</td>
<td>0.0036</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>278</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0029</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Result:

<table>
<thead>
<tr>
<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>2533</td>
<td>.002</td>
<td>.33</td>
<td>0</td>
<td>0.0065</td>
<td>0</td>
<td>0</td>
<td>0.0079</td>
</tr>
<tr>
<td>927</td>
<td>.0022</td>
<td>.0017</td>
<td>0</td>
<td>0.0021</td>
<td>0.003</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2417</td>
<td>.0083</td>
<td>0</td>
<td>0.28</td>
<td>0.0083</td>
<td>0.0054</td>
<td>0.0011</td>
<td>0.0025</td>
</tr>
<tr>
<td>746</td>
<td>.0063</td>
<td>0</td>
<td>0</td>
<td>0.0021</td>
<td>0.0056</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>158</td>
<td>0</td>
<td>0</td>
<td>0.52</td>
<td>0.0003</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1093</td>
<td>0</td>
<td>0</td>
<td>0.52</td>
<td>0.0003</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>341</td>
<td>0</td>
<td>0.0036</td>
<td>0</td>
<td>0.0036</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>278</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0029</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Bigram estimates of sentence probabilities

\[
P(\text{<s> I want english food </s>}) = \ P(\text{I | <s>}) \times \ P(\text{want | I}) \times \ P(\text{english | want}) \times \ P(\text{food | english}) \times \ P(\text{</s> | food}) = .000031
\]

Knowledge

• \( P(\text{english | want}) = .0011 \)
• \( P(\text{chinese | want}) = .0065 \)
• \( P(\text{to | want}) = .66 \)
• \( P(\text{eat | to}) = .28 \)
• \( P(\text{food | to}) = 0 \)
• \( P(\text{want | spend}) = 0 \)
• \( P(\text{ i | <s>}) = .25 \)

Practical Issues

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

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

Language Modeling Toolkits

• SRILM
  • \( \text{http://www.speech.sri.com/projects/srilm/} \)

Google N-Gram Release, August 2006

All Our N-gram are Belong to You

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...

That's why we decided to share this enormous dataset with everyone. We processed 1,024,000,287,220 words of running text and are publishing the counts for all 1,176,470,653 five-word sequences that appear at least 40 times. There are 15,989,391 unique words, after discarding words that appear less than 200 times.
Google N-Gram Release

• serve as the incoming 92
• serve as the incubator 99
• serve as the independent 794
• serve as the index 223
• serve as the indication 72
• serve as the indicator 120
• serve as the indicators 45
• serve as the indispensable 111
• serve as the indispensable 40
• serve as the individual 234

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

Today’s Outline

• Probabilistic language model and n-grams
• Estimating n-gram probabilities
  • Language model evaluation and perplexity
• Generalization and zeros
  • Smoothing: add-one
  • Interpolation, backoff, and web-scale LMs
  • Smoothing: Kneser-Ney Smoothing

Evaluation: How good is our model?

• Does our language model prefer good sentences to bad ones?
  • Assign higher probability to “real” or “frequently observed” sentences
  • Than “ungrammatical” or “rarely observed” sentences?
• We train parameters of our model on a training set.
• We test the model’s performance on data we haven’t seen.
  • A test set is an unseen dataset that is different from our training set, totally unused.
  • An evaluation metric tells us how well our model does on the test set.

Training on the test set

• We can’t allow test sentences into the training set
• We will assign it an artificially high probability when we set it in the test set
  • “Training on the test set”
• Bad science!
  • And violates the honor code

Extrinsic evaluation of N-gram models

• Best evaluation for comparing models A and B
  • Put each model in a task
    • spelling corrector, speech recognizer, MT system
    • Run the task, get an accuracy for A and for B
    • How many misspelled words corrected properly
    • How many words translated correctly
    • Compare accuracy for A and B

Difficulty of extrinsic (in-vivo) evaluation of N-gram models

• Extrinsic evaluation
  • Time-consuming; can take days or weeks
  • So
    • Sometimes use intrinsic evaluation: perplexity
    • Bad approximation
      • unless the test data looks just like the training data
    • So generally only useful in pilot experiments
    • But is helpful to think about.
The Shannon Game:
- How well can we predict the next word?
- 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

Perplexity

The best language model is one that best predicts an unseen test set
- Gives the highest $P(\text{sentence})$
- Minimizing perplexity is the same as maximizing probability

Perplexity

Perplexity as branching factor

Lower perplexity = better model

Today's Outline

The Shannon Visualization Method

<table>
<thead>
<tr>
<th>N-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>
Approximating Shakespeare

Shakespeare as corpus

- N=884,647 tokens, V=29,066
- 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)
- Quadrigrams worse: What’s coming out looks like Shakespeare because it is Shakespeare

The wall street journal is not shakespeare (no offense)

The perils of overfitting

- N-grams only work well for word prediction if the test corpus looks like the training corpus
  - In real life, it often doesn’t
- We need to train robust models that generalize!
  - One kind of generalization: Zeros!
  - Things that don’t ever occur in the training set
    - But occur in the test set

Zeros

- Training set:
  - ... denied the allegations
  - ... denied the reports
  - ... denied the claims
  - ... denied the request
- Test set:
  - ... denied the offer
  - ... denied the loan
- P("offer" | denied the) = 0

Zero probability bigrams

- Bigrams with zero probability
  - Mean that we will assign 0 probability to the test set
  - And hence we cannot compute perplexity (can’t divide by 0)
Today's Outline
• Probabilistic language model and n-grams
• Estimating n-gram probabilities
• Language model evaluation and perplexity
• Generalization and zeros
• Smoothing: add-one
• Interpolation, backoff, and web-scale LMs
• Smoothing: Kneser-Ney Smoothing

The intuition of smoothing (from Dan Klein)
• When we have sparse statistics:
  • Steal probability mass to generalize better

Add-one estimation
• Also called Laplace smoothing
• Pretend we saw each word one more time than we did
• Just add one to all the counts!
• MLE estimate:
  \[
P_{\text{MLE}}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}
\]
• Add-1 estimate:
  \[
P_{\text{Add-1}}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}
\]

Maximum Likelihood Estimates
• The maximum likelihood estimate
  • of some parameter of a model M from a training set T
  • maximizes the likelihood of the training set T given the model M
• Suppose the word “bagel” occurs 400 times in a corpus of a million words
• What is the probability that a random word from some other text will be “bagel”?
• MLE estimate is 400/1,000,000 = 0.0004
• This may be a bad estimate for some other corpus
  • But it is the estimate that makes it most likely that “bagel” will occur 400 times in a million word corpus.

Berkeley Restaurant Corpus: Laplace-smoothed bigram counts

<table>
<thead>
<tr>
<th></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>6</td>
<td>828</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>want</td>
<td>3</td>
<td>1</td>
<td>609</td>
<td>2</td>
<td>7</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>to</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>687</td>
<td>3</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>eat</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>17</td>
<td>3</td>
<td>43</td>
</tr>
<tr>
<td>chinese</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>food</td>
<td>16</td>
<td>1</td>
<td>16</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>lunch</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>spend</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Laplace-smoothed bigrams

\[
P^*(w_i | w_{i-1}) = \frac{C(w_{i-1}w_i) + 1}{C(w_{i-1}) + V}
\]
Reconstituted counts

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

<table>
<thead>
<tr>
<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>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>1.86</td>
<td>0.78</td>
</tr>
<tr>
<td>to</td>
<td>1.9</td>
<td>0.63</td>
<td>3.1</td>
<td>4.53</td>
<td>1.9</td>
<td>0.63</td>
<td>1.33</td>
</tr>
</tbody>
</table>

Compare with raw bigram counts

<table>
<thead>
<tr>
<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>want</td>
<td>1.5</td>
<td>0.39</td>
<td>238</td>
<td>0.78</td>
<td>2.7</td>
<td>1.86</td>
<td>0.78</td>
</tr>
<tr>
<td>to</td>
<td>1.9</td>
<td>0.63</td>
<td>3.1</td>
<td>4.53</td>
<td>1.9</td>
<td>0.63</td>
<td>1.33</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>1.15</td>
<td>0.34</td>
</tr>
<tr>
<td>chinese</td>
<td>0.2</td>
<td>0.098</td>
<td>0.098</td>
<td>0.098</td>
<td>8.2</td>
<td>0.2</td>
<td>0.098</td>
</tr>
<tr>
<td>food</td>
<td>6.9</td>
<td>0.43</td>
<td>6.9</td>
<td>0.43</td>
<td>0.86</td>
<td>2.2</td>
<td>0.43</td>
</tr>
<tr>
<td>lunch</td>
<td>0.57</td>
<td>0.19</td>
<td>0.19</td>
<td>0.19</td>
<td>0.38</td>
<td>0.19</td>
<td>0.19</td>
</tr>
<tr>
<td>spend</td>
<td>0.32</td>
<td>0.16</td>
<td>0.32</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
</tr>
</tbody>
</table>

Add-1 estimation is a blunt instrument

- So add-1 isn’t used for N-grams:
  - We’ll see better methods
- But add-1 is used to smooth other NLP models
  - For text classification
  - In domains where the number of zeros isn’t so huge.

Backoff and Interpolation

- Sometimes it helps to use less context
  - Condition on less context for contexts you haven’t learned much about
- Backoff:
  - Use trigram if you have good evidence,
  - Otherwise bigram, otherwise unigram
- Interpolation:
  - Mix unigram, bigram, trigram
- Interpolation works better

Today’s Outline

- Probabilistic language model and n-grams
- Estimating n-gram probabilities
- Language model evaluation and perplexity
- Generalization and zeros
- Smoothing: add-one
  - Interpolation, backoff, and web-scale LMs
- Smoothing: Kneser-Ney Smoothing

Linear Interpolation

- Simple interpolation
  \[
  \hat{P}(w_tw_{t+1}) = \lambda_1 P(w_tw_{t+1}) + \lambda_2 P(w_{t-1}w_{t+1}) + \lambda_3 P(w_{t-2}w_{t+1}) + \sum_{i=1}^{3} \lambda_i = 1
  \]
- Lambda conditional on context:
  \[
  \hat{P}(w_tw_{t+1}) = \lambda_1 P(w_tw_{t+1}) + \lambda_2 P(w_{t-1}w_{t+1}) + \lambda_3 P(w_{t-2}w_{t+1}) + \lambda_4 P(w_t) + \sum_{i=1}^{4} \lambda_i = 1
  \]
**Efficiency**

How to deal with, e.g., Google N-gram corpus

• Use a held-out corpus

• Choose \( \lambda \) to maximize the probability of held-out data:
  
  \[
  \log P(w_1…w_n | M(\lambda_1…\lambda_k)) = \sum \log P_{\lambda_1…\lambda_k}(w_j | w_{j-1})
  \]

**Pruning**

• Fix the N-gram probabilities (on the training data)
• Then search for \( \lambda \) that give largest probability to held-out set:

**How to set the lambdas?**

- Choose \( \lambda \) to maximize the probability of held-out data.
- Fix the N-gram probabilities (on the training data).
- Then search for \( \lambda \) that give largest probability to held-out set:

\[
\log P(w_1…w_n | M(\lambda_1…\lambda_k)) = \sum \log P_{\lambda_1…\lambda_k}(w_j | w_{j-1})
\]

**Smoothing for Web-scale N-grams**

• “Stupid backoff” (Brants et al., 2007)
• No discounting, just use relative frequencies

\[
S(w_i | w_{i-1}^{out}) = \begin{cases} 
\frac{\text{count}(w_i | w_{i-1}^{out})}{\text{count}(w_{i-1}^{out})} & \text{if count}(w_{i-1}^{out}) > 0 \\
0.4S(w_i | w_{i-1}^{out}) & \text{otherwise}
\end{cases}
\]

\[
S(w_i) = \frac{\text{count}(w_i)}{N}
\]

**Unknown words: Open versus closed vocabulary tasks**

• If we know all the words in advanced
  • Vocabulary \( V \) is fixed
  • Closed vocabulary task

• Often we don’t know this
  • Out Of Vocabulary = OOV words
  • Open vocabulary task

• 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>)
    • At decoding time
      • If test input: Use \(<UNK>) probabilities for any word not in training

**Huge web-scale n-grams**

• How to deal with, e.g., Google N-gram corpus
• Pruning
  • Only store N-grams with count > threshold.
  • Remove singletons of higher-order n-grams
  • Entropy-based pruning
• Efficiency
  • Efficient data structures like tries
  • Bloom filters: approximate language models
  • Store words as indexes, not strings
  • Use Huffman coding to fit large numbers of words into two bytes
  • Quantize probabilities (4.8 bits instead of 8-byte float)

**Today’s Outline**

• Probabilistic language model and n-grams
• Estimating n-gram probabilities
• Language model evaluation and perplexity
• Generalization and zeros
• Smoothing: add-one
• Interpolation, backoff, and web-scale LMs
• Smoothing: Kneser-Ney Smoothing

**Absolute discounting: just subtract a little from each count**

• Suppose we wanted to subtract a little from a count of 4 to save probability mass for the zeros
• How much to subtract?

- Church and Gale (1991)’s clever idea
- Divide up 22 million words of AP Newswire
  • Training and held-out set
  • for each bigram in the training set
  • see the actual count in the held-out set!

- It sure looks like \( c^* = (c - .75) \)

<table>
<thead>
<tr>
<th>Bigram count in training</th>
<th>Bigram count in heldout set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000270</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>
### Absolute Discounting Interpolation

- **Save ourselves some time and just subtract 0.75 (or some d)!**

  \[
P_{\text{AbsoluteDiscounting}}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1}) P(w)\
  \]

- *(Maybe keeping a couple extra values of d for counts 1 and 2)*
- *But should we really just use the regular unigram P(w)?*

### Kneser-Ney Smoothing I

- **Better estimate for probabilities of lower-order unigrams!**
  - Shannon game: *I can’t see without my reading glasses Francisco*
  - *Francisco* is more common than *glasses* in this context.
  - *Francisco* always follows *San*
- **The unigram is useful exactly when we haven’t seen this bigram!**
- **Instead of P(w): “How likely is w”**
- **P(continued): “How likely is w to appear as a novel continuation?**
  - For each word, count the number of bigram types it completes
  - Every bigram type was a novel continuation the first time it was seen

\[
P_{\text{CONTINUATION}}(w) = \frac{[\{w_{i-1}; c(w_{i-1}, w) > 0\}]}{\sum [\{w_{i-1}; c(w_{i-1}, w) > 0\}]}
\]

### Kneser-Ney Smoothing II

- **How many times does w appear as a novel continuation:**

  \[
P_{\text{CONTINUATION}}(w) = \frac{[\{w_{i-1}; c(w_{i-1}, w) > 0\}]}{\sum [\{w_{i-1}; c(w_{i-1}, w) > 0\}]}
  \]

- Normalized by the total number of word bigram types

### Kneser-Ney Smoothing III

- **Alternative metaphor:** The number of # of word types seen to precede w
  \[
  \frac{\{\{w_{i-1}; c(w_{i-1}, w) > 0\}\}}{\# words preceding all words}\
  \]
  - **normalized by the # of words preceding all words:**

\[
P_{\text{CONTINUATION}}(w) = \frac{[\{w_{i-1}; c(w_{i-1}, w) > 0\}]}{\sum [\{w_{i-1}; c(w_{i-1}, w) > 0\}]}
\]

- A frequent word (Francisco) occurring in only one context (San) will have a low continuation probability

### Kneser-Ney Smoothing IV

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

- \(\lambda\) is a normalizing constant; the probability mass we’ve discounted

\[
\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} [\{w_i; c(w_{i-1}, w) > 0\}]
\]

### Kneser-Ney Smoothing: Recursive formulation

\[
P_{\text{KN}}(w_j | w_{i-1}^{i+1}) = \frac{\max(c_{\text{KN}}(w_{i-1}^{j+1}) - d, 0)}{c_{\text{KN}}(w_{i-1}^{j+1})} + \lambda(w_{i-1}^{j+1}) P_{\text{KN}}(w_j | w_{i-1}^{j+1})
\]

\[
c_{\text{KN}}(\bullet) = \begin{cases} \text{count(\bullet) for the highest order} \\ \text{continuationcount(\bullet) for lower order} \end{cases}
\]

Continuation count = Number of unique single word contexts for \(\bullet\)
What We Learned Today

• Probabilistic language model and n-grams
• Estimating n-gram probabilities
• Language model evaluation and perplexity
• Generalization and zeros
• Smoothing: add-one
• Interpolation, backoff, and web-scale LMs
• Smoothing: Kneser-Ney Smoothing

Homework

• Reading J&M ch1 and ch4.1-4.9
• Sign up at Piazza
  • http://piazza.com/northeastern/fall2017/cs6120
  • Start thinking about course project and find a team
  • Project proposal due Oct 2