CS 4120: Natural Language Processing

Instructor: Prof. Lu Wang
Northeastern University
Webpage: www.ccs.neu.edu/home/luwang
Logistics

• Office hours
  • Prof. Lu Wang, **Thursdays** 1:30pm - 2:30pm, or by appointment, Rm 2211, 177 Huntington Ave
    • To attend OH at 177 Huntington Ave., you'll need to put down your name on Piazza beforehand (by 1pm each Monday), and then bring a photo ID (e.g. husky card) with you and check in at the front desk.
  • TA Akshay Vasant Dangare (email: dangare.a@husky.neu.edu), **Mondays and Wednesdays**, 4pm-5pm, 162 WVH (exception: on April 8th, the OH will be in the 1st floor lab at WVH)
Logistics

• Sign up on piazza!
  • piazza.com/northeastern/spring2020/cs4120

• Course website
  • http://www.ccs.neu.edu/home/luwang/courses/cs4120_sp2020/cs4120_sp2020.html
Project proposal (due Jan 28)

- In general, we want to see that you have a clear goal in the project. The technical details can be described in a rough manner, but in principle, you need to show what problem you want to study, and what is novel of your project.
  - **Introduction**: the problem has to be well-defined. What are the input and output. Why this is an important problem to study.
  - **Related work**: put your work in context. Describe what has been done in previous work on the same or related subject. And why what you propose to do here is novel and different.
  - **Datasets**: what data do you want to use? What is the size of it? What information is contained? Why is it suitable for your task?
  - **Methodology**: what models do you want to use? You may change the model as the project goes, but you may want to indicate some type of models that might be suitable for your problem. Is it a supervised learning problem or unsupervised? What classifiers can you start with? Are you making improvements? You don't have to be crystal clear on this section, but it can be used to indicate the direction that your project goes to.
  - **Evaluation**: what metrics do you want to use for evaluating your models?

- Length: 1 page (or more if necessary). Single space if MS word is used. Or you can choose latex templates, e.g. https://www.acm.org/publications/proceedings-template.

- Grading: based on each section described above, 20 points per section. But as you can tell, they're related to each other.

- Each group just needs to submit one copy on blackboard with all group member names indicated.
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

[Modified from Dan Jurafsky’s slides]
Probabilistic Language Models

• Assign a probability to a sentence
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 minuets from my house
      • $P(\text{about fifteen minuets from}) > P(\text{about fifteen minuets from})$
  • 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:
  \[ P(W) = P(w_1, w_2, w_3, w_4, w_5 \ldots w_n) \]

• Related task: probability of an upcoming word:
  \[ P(w_5 \mid w_1, w_2, w_3, w_4) \]

• A model that computes either of these:
  \[ P(W) \quad \text{or} \quad P(w_n \mid w_1, w_2 \ldots w_{n-1}) \]
  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:

  $P(\text{its, water, is, so, transparent, that})$
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
  \[ P(x_1,x_2,x_3,...,x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)\ldots P(x_n|x_1,...,x_{n-1}) \]
The Chain Rule applied to compute joint probability of words in sentence

\[ P(w_1w_2\ldots w_n) = \prod_{i} P(w_i \mid w_1w_2\ldots w_{i-1}) \]
The Chain Rule applied to compute joint probability of words in sentence

$$P(w_1w_2\ldots w_n) = \prod_{i} P(w_i \mid w_1w_2\ldots w_{i-1})$$

P(“its water is so transparent”) =

\[ P(\text{its}) \times P(\text{water} \mid \text{its}) \times P(\text{is} \mid \text{its water}) \times P(\text{so} \mid \text{its water is}) \times P(\text{transparent} \mid \text{its water is so}) \]
How to estimate these probabilities

• Could we just count and divide?

\[ P(\text{the its water is so transparent that}) = \frac{\text{Count(its water is so transparent that the)}}{\text{Count(its water is so transparent that)}} \]
How to estimate these probabilities

• Could we just count and divide?

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

• No! Too many possible sentences!
• We’ll never see enough data for estimating these
Markov Assumption

• Simplifying assumption:

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

• Or maybe

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

\[ P(w_1w_2\ldots w_n) \approx \prod_i P(w_i | w_{i-k}\ldots w_{i-1}) \]

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

\[ P(w_i | w_1w_2\ldots w_{i-1}) \approx P(w_i | w_{i-k}\ldots w_{i-1}) \]
Simplest case: Unigram model

\[ P(w_1w_2\ldots w_n) \approx \prod_i P(w_i) \]

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:

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

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
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 for bigram probability

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

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

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

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>
An example

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

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

\[ P(\text{I} | <s>) = \frac{2}{3} = .67 \quad P(\text{Sam} | <s>) = \frac{1}{3} = .33 \quad P(\text{am} | \text{I}) = \frac{2}{3} = .67 \]

\[ P(<s> | \text{I}) = \frac{1}{2} = 0.5 \quad P(\text{Sam} | \text{am}) = \frac{1}{2} = .5 \quad P(\text{do} | \text{I}) = \frac{1}{3} = .33 \]
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 counts

- Out of 9222 sentences

<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>
Raw bigram probabilities

- Normalize by unigrams:

<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>2533</td>
<td>927</td>
<td>2417</td>
<td>746</td>
<td>158</td>
<td>1093</td>
<td>341</td>
<td>278</td>
<td></td>
</tr>
</tbody>
</table>

- Result:

<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</td>
<td>0.0036</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>0.0017</td>
<td>0.0011</td>
<td>0.0065</td>
<td>0</td>
<td>0.0054</td>
</tr>
<tr>
<td>to</td>
<td>0.00083</td>
<td>0</td>
<td>0.0017</td>
<td>0.28</td>
<td>0.00083</td>
<td>0.021</td>
<td>0.0065</td>
<td>0.0025</td>
</tr>
<tr>
<td>eat</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.027</td>
<td>0.056</td>
</tr>
<tr>
<td>chinese</td>
<td>0.0063</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.52</td>
<td>0</td>
<td>0.0063</td>
</tr>
<tr>
<td>food</td>
<td>0.014</td>
<td>0</td>
<td>0.014</td>
<td>0</td>
<td>0.00092</td>
<td>0.0037</td>
<td>0.0029</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</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 estimates of sentence probabilities

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

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

\[ = .000031 \]
Knowledge

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

• Neural language models (will be discussed later)
  • Word2vec
  • Glove
  • Elmo and BERT
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?
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?
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!
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 evaluation of N-gram models

• Extrinsic evaluation
  • Time-consuming; can take days or weeks

• So
  • Sometimes use **intrinsic** evaluation: **perplexity**
Difficulty of extrinsic 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.
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
Intuition of Perplexity

• The Shannon Game:
  • How well can we predict the next word?
    I always order pizza with cheese and ____
    The 33\textsuperscript{rd} 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
    - mushrooms 0.1
    - pepperoni 0.1
    - anchovies 0.01
    - ....
    - fried rice 0.0001
    - ....
    - and 1e-100
Perplexity

The best language model is one that best predicts an unseen test set

• Gives the highest $P(\text{sentence})$

Perplexity is the inverse probability of the test set, normalized by the number of words:

$$PP(W) = P(w_1w_2...w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1w_2...w_N)}}$$
Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest P(sentence)

Perplexity is the inverse probability of the test set, normalized by the number of words:

\[ PP(W) = P(w_1w_2...w_N)^{-\frac{1}{N}} \]

\[ = \frac{1}{\sqrt[1]{P(w_1w_2...w_N)}} \]

Chain rule:

\[ PP(W) = \sqrt[N]{\frac{1}{\prod_{i=1}^{N} P(w_i|w_1...w_{i-1})}} \]

For bigrams:

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

The best language model is one that best predicts an unseen test set

• Gives the highest $P$ (sentence)

Perplexity is the inverse probability of the test set, normalized by the number of words:

$$PP(W) = \frac{1}{P(w_1w_2...w_N)^{\frac{1}{N}}}$$

$$= \frac{1}{\sqrt[N]{P(w_1w_2...w_N)}}$$

Chain rule:

For bigrams:

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1...w_{i-1})}}$$

Minimizing perplexity is the same as maximizing probability
Perplexity as branching factor

• Let’s suppose a sentence consisting of random digits
• What is the perplexity of this sentence according to a model that assign P=1/10 to each digit?
Perplexity as branching factor

• Let’s suppose a sentence consisting of random digits
• What is the perplexity of this sentence according to a model that assign $P=1/10$ to each digit?

\[
\text{PP}(W) = P(w_1w_2\ldots w_N)^{-\frac{1}{N}}
\]

\[
= \left( \frac{1}{10} \right)^{-\frac{1}{N}}
\]

\[
= \frac{1}{10^{-1}}
\]

\[
= \frac{1}{10}
\]

\[
= 10
\]
Lower perplexity = better 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>
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 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!
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

In training set, we see
  ... denied the allegations
  ... denied the reports
  ... denied the claims
  ... denied the request

But in test set,
  ... denied the offer
  ... denied the loan

\[ P(\text{“offer”} \mid \text{denied the}) = 0 \]
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:

\[ P(w \mid \text{denied the}) \]
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
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! (Instead of taking away counts)

- MLE estimate:
  \[ P_{MLE}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})} \]

- Add-1 estimate:
  \[ P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V} \]

V is the size of vocabulary
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! (Instead of taking away counts)

• MLE estimate:

$$P_{MLE}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

• Add-1 estimate:

$$P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

Why add 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>6</td>
<td>828</td>
<td>1</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>1</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>
<td>212</td>
</tr>
<tr>
<td>eat</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>17</td>
<td>3</td>
<td>43</td>
<td>1</td>
</tr>
<tr>
<td>chinese</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>83</td>
<td>2</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>
<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>
<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>
<td>1</td>
</tr>
</tbody>
</table>
Laplace-smoothed bigrams

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

<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>0.0015</td>
<td>0.21</td>
<td>0.00025</td>
<td>0.0025</td>
<td>0.00025</td>
<td>0.00025</td>
<td>0.00025</td>
<td>0.00075</td>
</tr>
<tr>
<td>want</td>
<td>0.0013</td>
<td>0.00042</td>
<td>0.26</td>
<td>0.00084</td>
<td>0.0029</td>
<td>0.0029</td>
<td>0.0025</td>
<td>0.00084</td>
</tr>
<tr>
<td>to</td>
<td>0.00078</td>
<td>0.00026</td>
<td>0.0013</td>
<td>0.18</td>
<td>0.00078</td>
<td>0.00026</td>
<td>0.0018</td>
<td>0.055</td>
</tr>
<tr>
<td>eat</td>
<td>0.00046</td>
<td>0.00046</td>
<td>0.0014</td>
<td>0.00046</td>
<td>0.0078</td>
<td>0.0014</td>
<td>0.02</td>
<td>0.00046</td>
</tr>
<tr>
<td>chinese</td>
<td>0.0012</td>
<td>0.00062</td>
<td>0.00062</td>
<td>0.00062</td>
<td>0.00062</td>
<td>0.052</td>
<td>0.0012</td>
<td>0.00062</td>
</tr>
<tr>
<td>food</td>
<td>0.0063</td>
<td>0.00039</td>
<td>0.0063</td>
<td>0.00039</td>
<td>0.00079</td>
<td>0.002</td>
<td>0.00039</td>
<td>0.00039</td>
</tr>
<tr>
<td>lunch</td>
<td>0.0017</td>
<td>0.00056</td>
<td>0.00056</td>
<td>0.00056</td>
<td>0.00056</td>
<td>0.0011</td>
<td>0.00056</td>
<td>0.00056</td>
</tr>
<tr>
<td>spend</td>
<td>0.0012</td>
<td>0.00058</td>
<td>0.0012</td>
<td>0.00058</td>
<td>0.00058</td>
<td>0.00058</td>
<td>0.00058</td>
<td>0.00058</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
  • (nowadays, neural LM becomes popular, will discuss later)

• But add-1 is used to smooth other NLP models
  • For text classification (coming soon!)
  • In domains where the number of zeros isn’t so huge.

• Add-1 can be extended to add-k (k can be any positive real number, sometimes also called add-alpha)
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
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 (e.g. the trigram is observed in training)
  • otherwise bigram
  • otherwise unigram
• Interpolation:
  • mix unigram, bigram, trigram

• In general, interpolation works better
Linear Interpolation

• Simple interpolation

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

\[\sum_i \lambda_i = 1\]
How to set the lambdas?

• Use a **held-out** corpus

<table>
<thead>
<tr>
<th>Training Data</th>
<th>Held-Out Data</th>
<th>Test Data</th>
</tr>
</thead>
</table>

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

$$\log P(w_1...w_n \mid M(\lambda_1...\lambda_k)) = \sum_i \log P_{M(\lambda_1...\lambda_k)}(w_i \mid w_{i-1})$$

An assignment of $\lambda$s
A Common Method – Grid Search

• Take a list of possible values, e.g. [0.1, 0.2, ..., 0.9]
• Try all combinations
4.4.3 Backoff and Interpolation

In language modeling, generating counts with poor variances and often inappropriate tasks (including text classification), it turns out that it still doesn't work well for done, for example, by optimizing on a devset. Although add-k is useful for some tasks, it is not sufficient, otherwise we use the bigram, otherwise the unigram. In other words, we use the trigram if the evidence is particularly accurate counts for a particular bigram, we assume that the probability mass is moved to all the zeros.

There are two ways to use this N-gram "hierarchy". In

In a slightly more sophisticated version of linear interpolation, each

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

\[ \sum_i \lambda_i = 1 \]

The discounting we have been discussing so far can help solve the problem of zero discounts. But there is an additional source of knowledge we can draw from the seen to the unseen events. Instead of adding 1 to each count, we add a fraction of the probability mass to all the zeros.

The sharp change in counts and probabilities occurs because too much probabil-

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

\[ \text{Short notation for } w_{n-1}w_{n-2} \]

The discounting we have been discussing so far can help solve the problem of zero discounts. But there is an additional source of knowledge we can draw from the seen to the unseen events. Instead of adding 1 to each count, we add a fraction of the probability mass to all the zeros.

The sharp change in counts and probabilities occurs because too much probabil-

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

\[ \text{Short notation for } w_{n-1}w_{n-2} \]
Linear Interpolation

• Simple interpolation

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

\[ \sum_i \lambda_i = 1 \]

• Lambdas conditional on context:

\[ \hat{P}(w_n | w_{n-2}w_{n-1}) = \lambda_1 (w_{n-1}^{n-1}) P(w_n | w_{n-2}w_{n-1}) + \lambda_2 (w_{n-1}^{n-1}) P(w_n | w_{n-1}) + \lambda_3 (w_{n-2}^{n-1}) P(w_n) \]
Unknown words: Open versus closed vocabulary tasks

• If we know all the words in advance
  • Vocabulary $V$ is fixed
  • Closed vocabulary task

• Often we don’t know this
  • **Out Of Vocabulary** = OOV words
  • Open vocabulary task
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 $<\text{UNK}>$
  • Training of $<\text{UNK}>$ probabilities
    • Create a fixed lexicon $L$ of size $V$ (e.g. selecting high frequency words)
    • At text normalization phase, any training word not in $L$ changed to $<\text{UNK}>$
    • Now we train its probabilities like a normal word
  • At test time
    • Use UNK probabilities for any word not in training
Smoothing for Web-scale N-grams

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

\[
S(w_i | w_{i-k+1}^{i-1}) = \begin{cases} 
\frac{\text{count}(w_{i-k+1}^i)}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \text{count}(w_{i-k+1}^i) > 0 \\
0.4S(w_i | w_{i-k+2}^{i-1}) & \text{otherwise}
\end{cases}
\]

\[
S(w_i) = \frac{\text{count}(w_i)}{N} \quad \text{Until unigram probability}
\]
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 $c^* = (c - .7)$

<table>
<thead>
<tr>
<th>Bigram count in training</th>
<th>Bigram count in heldout set</th>
</tr>
</thead>
<tbody>
<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 $d$)!

$$P_{\text{AbsoluteDiscounting}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1})P(w_i)$$

• 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_________*?  
  • “Francisco” is more common than “glasses”  
  • … but “Francisco” always follows “San”
Kneser-Ney Smoothing I

• Better estimate for probabilities of lower-order unigrams!
  • Shannon game: *I can’t see without my reading*_________?  *glasses*
  • “Francisco” is more common than “glasses”
  • … but “Francisco” always follows “San”

• The unigram is useful exactly when we haven’t seen this bigram!

• Instead of  $P(w)$: “How likely is $w$”

• $P_{\text{continuation}}(w)$: “How likely is $w$ to appear as a novel continuation?”
  • For each word, count the number of unique bigrams it completes
  • Every unique bigram was a novel continuation the first time it was seen

\[
P_{\text{CONTINUATION}}(w) \propto |\{w_{i-1} : c(w_{i-1}, w) > 0\}|
\]
Kneser-Ney Smoothing II

• How many times does $w$ appear as a novel continuation (unique bigrams):

$$P_{\text{CONTINUATION}}(w) \propto \left| \{ w_{i-1} : c(w_{i-1}, w) > 0 \} \right|$$

• Normalized by the total number of word bigram types

$$P_{\text{CONTINUATION}}(w) = \frac{\left| \{ w_{i-1} : c(w_{i-1}, w) > 0 \} \right|}{\left| \{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0 \} \right|}$$
Kneser-Ney Smoothing III

• Alternative metaphor: The number of unique words seen to precede w

\[ | \{ w_{i-1} : c(w_{i-1}, w) > 0 \} | \]

• normalized by the number of (unique) words preceding all words:

\[ P_{\text{CONTINUATION}}(w) = \frac{ | \{ w_{i-1} : c(w_{i-1}, w) > 0 \} | }{ \sum_{w'} \left| \{ w'_{i-1} : c(w'_{i-1}, w') > 0 \} \right| } \]

• A frequent word (Francisco) occurring in only one context (San) will have a low continuation probability
Kneser-Ney Smoothing IV (not required)

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

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

\[ \lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} \left| \{ w : c(w_{i-1}, w) > 0 \} \right| \]

The normalized discount

the normalized discount

The number of word types that can follow \( w_{i-1} \)

= # of word types we discounted

= # of times we applied normalized discount
Language Modeling

- 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
• Start thinking about course project and find a team
• Project proposal due Jan 28th.
• The format of the proposal will be posted on Piazza