CS 124/LINGUIST 180: From Language to Information

Dan Jurafsky

Lecture 3: Intro to Probability, Language Modeling

IP notice: some slides for today from: Jim Martin, Sandiway Fong, Dan Klein
Outline

• Probability
  ▶ Basic probability
  ▶ Conditional probability

• Language Modeling (N-grams)
  ▶ N-gram Intro
  ▶ The Chain Rule
  ▶ The Shannon Visualization Method
  ▶ Evaluation:
    ▪ Perplexity
  ▶ Smoothing:
    ▪ Laplace (Add-1)
    ▪ Add-prior
Language Modeling

• We want to compute
  - $P(w_1, w_2, w_3, w_4, w_5 \ldots w_n) = P(W)$
  - = the probability of a sequence

• Alternatively we want to compute
  - $P(w_5 | w_1, w_2, w_3, w_4)$
  - = the probability of a word given some previous words

• The model that computes
  - $P(W)$ or
  - $P(w_n | w_1, w_2 \ldots w_{n-1})$
  - is called the language model.

• A better term for this would be “The Grammar”
• But “Language model” or LM is standard
Computing $P(W)$

- How to compute this joint probability:
  - $P(\text{"the"}, \text{"other"}, \text{"day"}, \text{"I"}, \text{"was"}, \text{"walking"}, \text{"along"}, \text{"and"}, \text{"saw"}, \text{"a"}, \text{"lizard"})$

- Intuition: let’s rely on the Chain Rule of Probability
The Chain Rule

- Recall the definition of conditional probabilities
  \[ P(A \mid B) = \frac{P(A \land B)}{P(B)} \]
- Rewriting:
  \[ P(A \land B) = P(A \mid B)P(B) \]
- More generally
  \[ P(A,B,C,D) = P(A)P(B \mid A)P(C \mid A, B)P(D \mid A, B, C) \]
- In general
  \[ P(x_1, x_2, x_3, \ldots, x_n) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_1, x_2) \ldots P(x_n \mid x_1 \ldots x_{n-1}) \]
The Chain Rule applied to joint probability of words in sentence

\[
P(w_1^n) = P(w_1)P(w_2|w_1)P(w_3|w_1^2) \cdots P(w_n|w_1^{n-1})
= \prod_{k=1}^{n} P(w_k|w_1^{k-1})
\]

- \(P(\text{“the big red dog was”}) = P(\text{the}) \times P(\text{big}|\text{the}) \times P(\text{red}|\text{the big}) \times P(\text{dog}|\text{the big red}) \times P(\text{was}|\text{the big red dog})\)
Very easy estimate:

- How to estimate?
  - \( P(\text{the } | \text{ its water is so transparent that}) \)

\[
P(\text{the } | \text{ its water is so transparent that}) \\
= \\
\frac{C(\text{its water is so transparent that the})}{C(\text{its water is so transparent that})}
\]
Unfortunately

- There are a lot of possible sentences
- We’ll never be able to get enough data to compute the statistics for those long prefixes

- \( P(\text{lizard}|\text{the,other,day,I,was,walking,along,and,saw,a}) \)
- Or
- \( P(\text{the}|\text{its water is so transparent that}) \)
Markov Assumption

- Make the simplifying assumption
  - \( P(\text{lizard}| \text{the, other, day, I, was, walking, along, and, saw, a}) = P(\text{lizard}|a) \)

- Or maybe
  - \( P(\text{lizard}| \text{the, other, day, I, was, walking, along, and, saw, a}) = P(\text{lizard}|\text{saw, a}) \)
Markov Assumption

- So for each component in the product replace with the approximation (assuming a prefix of N)

\[ P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-N+1}^{n-1}) \]

- Bigram version

\[ P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-1}) \]
Estimating bigram probabilities

- The Maximum Likelihood Estimate

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

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

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

\[
P(I | <s>) = \frac{2}{3} = .67 \quad P(Sam | <s>) = \frac{1}{3} = .33 \quad P(\text{am} | I) = \frac{2}{3} = .67
\]
\[
P(</s> | Sam) = \frac{1}{2} = 0.5 \quad P(Sam | \text{am}) = \frac{1}{2} = .5 \quad P(\text{do} | I) = \frac{1}{3} = .33
\]

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

- This is the Maximum Likelihood Estimate, because it is the one which maximizes \(P(\text{Training set}|\text{Model})\)
Maximum Likelihood Estimates

• The maximum likelihood estimate of some parameter of a model M from a training set T
  ✷ Is the estimate
  ✷ that maximizes the likelihood of the training set T given the model M

• Suppose the word Chinese occurs 400 times in a corpus of a million words (Brown corpus)

• What is the probability that a random word from some other text will be “Chinese”

• MLE estimate is 400/1000000 = .004
  ✷ This may be a bad estimate for some other corpus

• But it is the estimate that makes it most likely that “Chinese” will occur 400 times in a million word corpus.
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
  - Eg. “I want” occurred 827 times

<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></td>
<td>1093</td>
<td>341</td>
<td>278</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>0.0036</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>0.00083</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>0.0027</td>
<td>0.056</td>
<td>0</td>
</tr>
<tr>
<td>chinese</td>
<td>0.0063</td>
<td>0</td>
<td>0.014</td>
<td>0.0014</td>
<td>0</td>
<td>0.52</td>
<td>0.0037</td>
<td>0</td>
</tr>
<tr>
<td>food</td>
<td>0.014</td>
<td>0</td>
<td>0.014</td>
<td>0</td>
<td>0.00092</td>
<td>0.021</td>
<td>0.0027</td>
<td>0</td>
</tr>
<tr>
<td>lunch</td>
<td>0.0059</td>
<td>0</td>
<td>0.0036</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0063</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

- \( P(<s> \text{ I want english food } </s>) = \)
  \[ P(i|<s>) \times P(\text{want}|I) \times P(\text{english}|\text{want}) \times P(\text{food}|\text{english}) \times P(</s>|\text{food}) \]
  \[ = .000031 \]
What kinds of knowledge?

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

- Generate random sentences:
- Choose a random bigram $<s>, w$ according to its probability
- Now choose a random bigram $(w, x)$ according to its probability
- And so on until we choose $</s>$
- Then string the words together
- $<s>$ I
  I want
  want to
  to eat
  eat Chinese
  Chinese food
  food $</s>$
Approximating Shakespeare

Unigram
To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have
Every enter now severally so, let
Hill he late speaks; or! a more to leg less first you enter
Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like

Bigram
What means, sir. I confess she? then all sorts, he is trim, captain.
Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.
What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman?

Trigram
Sweet prince, Falstaff shall die. Harry of Monmouth’s grave.
This shall forbid it should be branded, if renown made it empty.
Indeed the duke; and had a very good friend.
Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, ’tis done.

Quadrigram
King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv’d in;
Will you not tell me who I am?
It cannot be but so.
Indeed the short and the long. Marry, ’tis a noble Lepidus.
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 \textit{is} Shakespeare
The wall street journal is not shakespeare (no offense)

Unigram

Months the my and issue of year foreign new exchange’s september were recession exchange new endorsed a acquire to six executives

Bigram

Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her

Trigram

They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions
Lesson 1: 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, adapt to test set, etc
Lesson 2: zeros or not?

- Some of those zeros are really zeros...
  - Things that really aren’t ever going to happen
- On the other hand, some of them are just rare events.
  - If the training corpus had been a little bigger they would have had a count
    - What would that count be in all likelihood?
- Zipf’s Law (long tail phenomenon):
  - A small number of events occur with high frequency
  - A large number of events occur with low frequency
  - You can quickly collect statistics on the high frequency events
  - You might have to wait an arbitrarily long time to get valid statistics on low frequency events
- Result:
  - Our estimates are sparse! We have no counts at all for the vast bulk of things we want to estimate!
- Answer:
  - Estimate the likelihood of unseen (zero count) N-grams!
Smoothing is like Robin Hood: Steal from the rich and give to the poor (in probability mass)

- We often want to make predictions from sparse statistics:

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

- Smoothing flattens spiky distributions so they generalize better

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

- Very important all over NLP, but easy to do badly!

Slide from Dan Klein
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: \[ c^*_i = (c_i + 1) \frac{N}{N + V} \]
Laplace smoothed bigram counts

<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>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>
<td>212</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>
<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>0.0029</td>
<td>0.00078</td>
<td>0.0029</td>
<td>0.0018</td>
<td>0.00084</td>
</tr>
<tr>
<td>to</td>
<td>0.00078</td>
<td>0.00026</td>
<td>0.0013</td>
<td>0.00084</td>
<td>0.00026</td>
<td>0.0014</td>
<td>0.0014</td>
<td>0.055</td>
</tr>
<tr>
<td>eat</td>
<td>0.00046</td>
<td>0.00046</td>
<td>0.0014</td>
<td>0.0078</td>
<td>0.00026</td>
<td>0.052</td>
<td>0.002</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>0.0011</td>
<td>0.00056</td>
<td>0.00062</td>
</tr>
<tr>
<td>food</td>
<td>0.0063</td>
<td>0.00039</td>
<td>0.0063</td>
<td>0.00079</td>
<td>0.00039</td>
<td>0.00039</td>
<td>0.00039</td>
<td></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>0.00058</td>
<td>0.00058</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>
Reconstituted counts

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

<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>
<tr>
<td>chinese</td>
<td>0.2</td>
<td>0.098</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>
<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>0.19</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>
<td>0.16</td>
</tr>
</tbody>
</table>
Note big change to counts

- C(count 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
- 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.
Add-k

- Add a small fraction instead of 1
Bayesian unigram prior smoothing for bigrams

- Maximum Likelihood Estimation

\[ P(w_2 \mid w_1) = \frac{C(w_1, w_2)}{C(w_1)} \]

- Laplace Smoothing

\[ P_{Laplace}(w_2 \mid w_1) = \frac{C(w_1, w_2) + 1}{C(w_1) + vocab} \]

- Bayesian prior Smoothing

\[ P_{Prior}(w_2 \mid w_1) = \frac{C(w_1, w_2) + P(w_2)}{C(w_1) + 1} \]
Practical Issues

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

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

- SRILM
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 to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 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 indispensible 40
- serve as the individual 234
Evaluating N-gram models

- Best evaluation for an N-gram
  - Put model A in a speech recognizer
  - Run recognition, get word error rate (WER) for A
  - Put model B in speech recognition, get word error rate for B
  - Compare WER for A and B
  - **In-vivo evaluation**
Difficulty of in-vivo evaluation of N-gram models

• In-vivo 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 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.
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$>$
    - Now we train its probabilities like a normal word
  - At decoding time
    - If text input: Use UNK probabilities for any word not in training
Evaluating N-gram models

- Best evaluation for an N-gram
  - Put model A in a task (language identification, speech recognizer, machine translation system)
  - Run the task, get an accuracy for A (how many lgs identified correctly, or Word Error Rate, or etc)
  - Put model B in task, get accuracy for B
  - Compare accuracy for A and B
  - **Extrinsic evaluation**
Difficult 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.
Perplexity

- Perplexity is the probability of the test set (assigned by the language model), normalized by the number of words:

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

- Chain rule:

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

- For bigrams:

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

- Minimizing perplexity is the same as maximizing probability
- The best language model is one that best predicts an unseen test set
• How hard is the task of recognizing digits ‘0,1,2,3,4,5,6,7,8,9,oh’: easy, perplexity 11 (or if we ignore ‘oh’, perplexity 10)

• How hard is recognizing (30,000) names at Microsoft. Hard: perplexity = 30,000

• If a system has to recognize
  • Operator (1 in 4)
  • Sales (1 in 4)
  • Technical Support (1 in 4)
  • 30,000 names (1 in 120,000 each)
  • Perplexity is 54

• Perplexity is weighted equivalent branching factor
Perplexity as branching factor

\[ PP(W) = P(w_1w_2 \ldots w_N)^{-\frac{1}{N}} \]
\[ = \left( \frac{1}{10} \right)^{-\frac{N}{N}} \]
\[ = \left( \frac{1}{10} \right)^{-1} \]
\[ = \frac{1}{10} \]
\[ = 10 \]
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>
Advanced LM stuff

- Current best smoothing algorithm
  - Kneser-Ney smoothing

- Other stuff
  - Interpolation
  - Backoff
  - Variable-length n-grams
  - Class-based n-grams
    - Clustering
    - Hand-built classes
  - Cache LMs
  - Topic-based LMs
  - Sentence mixture models
  - Skipping LMs
  - Parser-based LMs
Better discounting algorithms

- 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
Types, Tokens and Squirrels

• Much of what’s coming up was first studied by field biologists who are often faced with 2 related problems
  ◦ Determining how many species occupy a particular area (types)
  ◦ And determining how many individuals of a given species are living in a given area (tokens)
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 next fish caught will be an eel?

How likely is it that next fish caught will be a member of an unseen species (i.e. catfish or bass)

- 3/18

Now how likely is it that next fish caught will be an eel?

- Must be less than 1/18
Good-Turing Intuition

• Notation: $N_x$ is the frequency-of-frequency-$x$
  - So $N_{10} = 1$, $N_1 = 3$, etc

• 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

$$c^* = (c + 1) \frac{N_c + 1}{N_c}$$

$C^*(eel) = c^*(1) = (1+1) \frac{1}{3} = 2/3$

Slide from Josh Goodman
<table>
<thead>
<tr>
<th></th>
<th>unseen (bass or catfish)</th>
<th>trout</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>MLE (p)</td>
<td>(p = \frac{0}{18} = 0)</td>
<td>(\frac{1}{18})</td>
</tr>
<tr>
<td>(c^*)</td>
<td></td>
<td>(c^*(\text{trout}) = 2 \times \frac{N_2}{N_1} = 2 \times \frac{1}{3} = .67)</td>
</tr>
<tr>
<td>GT (p_{GT}^*)</td>
<td>(p_{GT}^*(\text{unseen}) = \frac{N_1}{N} = \frac{3}{18} = .17)</td>
<td>(p_{GT}^*(\text{trout}) = \frac{.67}{18} = \frac{1}{27} = .037)</td>
</tr>
</tbody>
</table>
Bigram frequencies of frequencies and GT re-estimates

<table>
<thead>
<tr>
<th>AP Newswire</th>
<th>Berkeley Restaurant —</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>c (MLE)</strong></td>
<td><strong>N_c</strong></td>
</tr>
<tr>
<td>0</td>
<td>74,671,100,000</td>
</tr>
<tr>
<td>1</td>
<td>2,018,046</td>
</tr>
<tr>
<td>2</td>
<td>449,721</td>
</tr>
<tr>
<td>3</td>
<td>188,933</td>
</tr>
<tr>
<td>4</td>
<td>105,668</td>
</tr>
<tr>
<td>5</td>
<td>68,379</td>
</tr>
<tr>
<td>6</td>
<td>48,190</td>
</tr>
</tbody>
</table>
## 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>
Complications

• In practice, assume large counts \((c > k)\) for some \(k\) are reliable:

\[
c^* = c \quad \text{for } c > k
\]

• Also: we assume singleton counts \(c=1\) are unreliable, so treat N-grams with count of 1 as if they were count=0

• Also, need the \(N_k\) to be non-zero, so we need to smooth (interpolate) the \(N_k\) counts before computing \(c^*\) from them
Backoff and Interpolation

• Another really useful source of knowledge
• If we are estimating:
  ◦ trigram $p(z|xy)$
  ◦ but $c(xyz)$ is zero
• Use info from:
  ◦ Bigram $p(z|y)$
• Or even:
  ◦ Unigram $p(z)$
• How to combine the trigram/bigram/unigram info?
Backoff versus interpolation

• **Backoff**: use trigram if you have it, otherwise bigram, otherwise unigram

• **Interpolation**: mix all three
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** corpus
- Choose lambdas which maximize the probability of some held-out data
  - 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

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

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

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

- MLE probabilities sum to 1
  \[ \sum_i P(w_i|w_jw_k) = 1 \]

- 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

\[ P^*(w_n|w_{n-N+1}^{n-1}) = \frac{c^*(w_{n-N+1}^n)}{c(w_{n-N+1}^{n-1})} \]
<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>
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
ARPA format

unigram: \( \log p^* (w_i) \quad w_i \quad \log \alpha(w_i) \)
bigram: \( \log p^* (w_i | w_{i-1}) \quad w_{i-1}w_i \quad \log \alpha(w_{i-1}w_i) \)
trigram: \( \log p^* (w_i | w_{i-2}, w_{i-1}) \quad w_{i-2}w_{i-1}w_i \)
\data\nngram 1=1447
ngram 2=9420
ngram 3=5201

1-grams:
-0.8679678  </s>
-99  <s>   -1.068532
-4.743076  chow-fun  -0.1943932
-4.266155  fries  -0.5432462
-3.175167  thursday  -0.7510199
-1.776296  want  -1.04292
...

2-grams:
-0.6077676  <s>  i  -0.6257131
-0.4861297  i  want  0.0425899
-2.832415  to  drink  -0.06423882
-0.5469525  to  eat  -0.008193135
-0.09403705  today  </s>
...

3-grams:
-2.579416  <s>  i  prefer
-1.148009  <s>  about  fifteen
-0.4120701  to  go  to
-0.3735807  me  a  list
-0.260361  at  jupiter  </s>
-0.260361  a  malaysian restaurant
...
\end\
Summary

- Language Modeling (N-grams)
  - N-gram Intro
  - The Shannon Visualization Method
  - Evaluation:
    - Perplexity
  - Smoothing:
    - Laplace (Add-1)
    - Add-k
    - Good-Turing
    - Backoff
    - Interpolation