Introduction to N-grams
Probabilistic Language Models

Today’s goal: assign a probability to a sentence

- Machine Translation:
  - \( P(\text{high winds tonite}) > P(\text{large winds tonite}) \)

- Spell Correction
  - The office is about fifteen minuets from my house
  - \( P(\text{about fifteen minutes from}) > P(\text{about fifteen minuets from}) \)

- Speech Recognition
  - \( P(\text{I saw a van}) >> P(\text{eyes awe of an}) \)

- + Summarization, question-answering, etc., etc., etc.!!
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:
\[ P(w_5 | w_1, w_2, w_3, w_4) \]

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

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:

- $P(\text{its, water, is, so, transparent, that})$

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

Recall the definition of conditional probabilities
\[ p(B|A) = \frac{P(A,B)}{P(A)} \quad \text{Rewriting:} \quad P(A,B) = P(A)P(B|A) \]

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

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

P(“its water is so transparent”) =

P(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?

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

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

Simplifying assumption:

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

Or maybe

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

\[ P(w_{1}w_{2}\ldots w_{n}) \quad P(w_{i} \mid w_{i-k}\ldots w_{i-1}) \]

In other words, we approximate each component in the product

\[ P(w_{i} \mid w_{1}w_{2}\ldots w_{i-1}) \quad P(w_{i} \mid w_{i-k}\ldots w_{i-1}) \]
Simplest case: Unigram model

\[ P(w_1w_2 \cdots w_n) \quad P(w_i) \]

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:

\[ P(w_i \mid w_1w_2 \cdots w_{i-1}) \quad P(w_i \mid w_{i-1}) \]

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 which I had just put into the machine room on the fifth floor crashed.”

But we can often get away with N-gram models
Estimating N-gram Probabilities
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

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

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

\[
\begin{align*}
P(\text{I} | <s>) &= \frac{2}{3} = 0.67 \\
P(\text{Sam} | <s>) &= \frac{1}{3} = 0.33 \\
P(\text{am} | \text{I}) &= \frac{2}{3} = 0.67 \\
P(<s> | \text{Sam}) &= \frac{1}{2} = 0.5 \\
P(\text{Sam} | \text{am}) &= \frac{1}{2} = 0.5 \\
P(\text{do} | \text{I}) &= \frac{1}{3} = 0.33
\end{align*}
\]
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>15</td>
<td>0</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>0.0036</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</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>0.021</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</td>
<td>0</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>0.00092</td>
<td>0.52</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</td>
<td>0.0027</td>
<td>0</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>0.0037</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>0.0029</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Bigram estimates of sentence probabilities

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

= 0.000031
What kinds of knowledge?

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

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

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

\( 0 < P_i \ll 1 \)
Language Modeling Toolkits

SRILM

KenLM
- https://kheafield.com/code/kenlm/
That's why we decided 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.
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
Google Book N-grams

http://ngrams.googlelabs.com/
Language Modeling

Evaluation and Perplexity
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.
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.
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

\[
\begin{align*}
\text{mushrooms} & \text{ 0.1} \\
\text{pepperoni} & \text{ 0.1} \\
\text{anchovies} & \text{ 0.01} \\
\text{...} & \\
\text{fried rice} & \text{ 0.0001} \\
\text{...} & \\
\text{and 1e-100} & 
\end{align*}
\]
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:

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

Chain rule:

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

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?

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

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

$$= \frac{1}{10}^{-1}$$

$$= \frac{1}{10}$$

$$= 10$$
The Shannon Game intuition for perplexity

From Josh Goodman

Perplexity is weighted equivalent branching factor

How hard is the task of recognizing digits ‘0,1,2,3,4,5,6,7,8,9’
- Perplexity 10

How hard is recognizing (30,000) names at Microsoft.
- Perplexity = 30,000

Let's imagine a call-routing phone system gets 120K calls and has to recognize
- "Operator" (let's say this occurs 1 in 4 calls)
- "Sales" (1 in 4)
- "Technical Support" (1 in 4)
- 30,000 different names (each name occurring 1 time in the 120K calls)
- What is the perplexity? Next slide
The Shannon Game intuition for perplexity

Josh Goodman: imagine a call-routing phone system gets 120K calls and has to recognize
   ◦ "Operator" (let's say this occurs 1 in 4 calls)
   ◦ "Sales" (1 in 4)
   ◦ "Technical Support" (1 in 4)
   ◦ 30,000 different names (each name occurring 1 time in the 120K calls)

We get the perplexity of this sequence of length 120K by first multiplying 120K probabilities (90K of which are 1/4 and 30K of which are 1/120K), and then taking the inverse 120,000th root:

   \[ \text{Perp} = (\frac{1}{4} \times \frac{1}{4} \times \frac{1}{4} \times \frac{1}{4} \times \ldots \times \frac{1}{120K} \times \frac{1}{120K} \times \ldots)^{-\frac{1}{120K}} \]

But this can be arithmetically simplified to just \( N = 4 \): the operator (1/4), the sales (1/4), the tech support (1/4), and the 30,000 names (1/120,000):

   \[ \text{Perplexity} = (\frac{1}{4} \times \frac{1}{4} \times \frac{1}{4} \times \frac{1}{120K})^{-\frac{1}{4}} = 52.6 \]
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>
Language Modeling

Generalization and zeros
The Shannon Visualization Method

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
want
I want to
want to eat
eat Chinese
Chinese food
food
</s>

I want to eat Chinese food
Approximating Shakespeare

<table>
<thead>
<tr>
<th>gram</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have Hill he late speaks; or! a more to leg less first you enter</td>
</tr>
<tr>
<td>2</td>
<td>Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. What means, sir. I confess she? then all sorts, he is trim, captain.</td>
</tr>
<tr>
<td>3</td>
<td>Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, ’tis done. This shall forbid it should be branded, if renown made it empty.</td>
</tr>
<tr>
<td>4</td>
<td>King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv’d in; It cannot be but so.</td>
</tr>
</tbody>
</table>
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)

1 gram

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

2 gram

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.

3 gram

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.
Can you guess the training set author of the LM that generated these random 3-gram sentences?

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 gram Brazil on market conditions.

This shall forbid it should be branded, if renown made it empty.

“You are uniformly charming!” cried he, with a smile of associating and now and then I bowed and they perceived a chaise and four to wish for.
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

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

• Test set
  ... denied the offer
  ... denied the loan
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)!
Smoothing: Add-one (Laplace) smoothing
The intuition of smoothing (from Dan Klein)

When we have sparse statistics:

\[ P(w \mid \text{denied the}) \]
- 3 allegations
- 2 reports
- 1 claim
- 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!

MLE estimate:

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

Add-1 estimate:

\[
P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}
\]
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 $400/1,000,000 = .0004$

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 smooth 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>0.00084</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>
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>
### Compare with raw 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>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>

<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>
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.
Language Modeling

Interpolation, Backoff, and Web-Scale LMs
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
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:

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

Use a held-out corpus

Choose $\lambda$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:

$$\log P(w_1 \ldots w_n \mid M(1 \ldots k)) = \log P_{M(1 \ldots k)}(w_i \mid w_{i-1})$$
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
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)
Smoothing for Web-scale N-grams

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

\[
S(w_i \mid w_{i-k+1}^i) = \frac{\text{count}(w_{i-k+1}^i)}{\text{count}(w_{i-k+1}^i)} \quad \text{if} \quad \text{count}(w_{i-k+1}^i) > 0
\]

\[
0.4S(w_i \mid w_{i-k+2}^i) \quad \text{otherwise}
\]

\[
S(w_i) = \frac{\text{count}(w_i)}{N}
\]
N-gram Smoothing Summary

Add-1 smoothing:
- OK for text categorization, not for language modeling

The most commonly used method:
- Extended Interpolated Kneser-Ney

For very large N-grams like the Web:
- Stupid backoff
Advanced Language Modeling

Discriminative models:
- choose n-gram weights to improve a task, not to fit the training set

Parsing-based models

Caching Models
- Recently used words are more likely to appear

\[ P_{\text{CACHE}}(w \mid \text{history}) = P(w_i \mid w_{i-2}w_{i-1}) + (1 - \frac{c(w \mid \text{history})}{|\text{history}|}) \]
- These turned out to perform very poorly for speech recognition (why?)