Language Models

29 January 2024
A note about the readings
3 Language models

Jan. 29 Language models
Jan. 31 Model evaluation and smoothing
Assignment 2

Core reading:
- SLT § 5: N-gram language models (through 5.6)

Further exploration:
- Michel et al., 2011: Quantitative analysis of culture using millions of digitized books
- Jurgens et al., 2017: Incorporating dialectical variability for socially equitable language identification

4 Machine learning

Feb. 5 Text classification
Feb. 7 Regression
Assignment 3

Recommended background videos:
- 3brows/Blue: Bayes theorem, the geometry of changing beliefs
- 3brows/Blue: The quick proof of Bayes' theorem

Core reading:
- SLT § 4: Naive Bayes and sentiment classification
- SLT § 5: Logistic regression

Further exploration:
- Mitchell, 2020: Generative and discriminative classifiers: Naive Bayes and logistic regression
- Pang et al., 2002: Thumbs up! Sentiment classification using machine learning techniques
- Koolen & van Cranenburgh, 2017: These are not the stereotypes you are looking for: bias and fairness in authorial gender attribution

5 Lexical semantics
Programming in Python
Functional programming
Python is a multi-paradigm language:

You can use it for *imperative* programming (like C)
You can use it for *functional* programming (like Lisp)
You can use it for *object-oriented* programming (like Java)

And often you’ll do all three in a single program!
Out of those three paradigms, the one that most often gets forgotten is functional programming – but it’s very helpful!
Functions like this shouldn’t change their input!
Testing
When you’re working on the assignments, you’re not required to write test cases, but it’s a good idea to convince yourself each function does what it should.
$ pip3 install pytest
def add_three(x):
    return x + 3

def test_add_three():
    assert add_three(0) == 3
    assert add_three(-3) == 0
    assert add_three(3) == 6
; pytest example.py

============= test session starts ======================
platform darwin -- Python 3.12.1, pytest-8.0.0, pluggy-1.4.0
rootdir: /Users/jgordon/Downloads
collected 1 item

eexample.py .                              [100%]

============ 1 passed in 0.01s ===============
Sometimes the assignment gives you example output. If so, use these as test cases!

If not, read it carefully to be sure you understand the input and output types and what the function should do, and then write your own test cases.
Style and naming
PEP 8 – Style Guide for Python Code

Author: Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Alyssa Coghlan <ncoghlan at gmail.com>

Status: Active
Type: Process
Created: 05-Jul-2001
Post-History: 05-Jul-2001, 01-Aug-2013

Introduction

This document gives coding conventions for the Python code comprising the standard library in the main Python distribution. Please see the companion informational PEP describing style guidelines for the C code in the C implementation of Python.

This document and PEP 257 (Docstring Conventions) were adapted from Guido’s original Python Style Guide essay, with some additions from Barry’s style guide [2].

This style guide evolves over time as additional conventions are identified and past conventions are rendered obsolete by changes in the language itself.

Many projects have their own coding style guidelines. In the event of any conflicts, such project-specific guides take precedence for that project.

A Foolish Consistency is the Hobgoblin of Little Minds

One of Guido’s key insights is that code is read much more often than it is written. The guidelines provided here are intended to improve the readability of code and make it consistent across the wide spectrum of Python code. As PEP 20 says, “Readability counts!”
How to Write Beautiful Python Code With PEP 8

by Jasmine Finer  43 Comments

Table of Contents

- Why We Need PEP 8
- Naming Conventions
  - Naming Styles
  - How to Choose Names
variable_name = ...

def function_name(): ...

class ClassName: ...
Use spaces around operators:

\[(1 + 2) * 3\]
Use spaces after every comma, just like in English:

```python
function(arg1, arg2)
```
Everyone – except Google – uses 4-space indentation in Python.

Be sure every level of indentation is the same!
Comments
# These are the imports
import sys
import numpy as np
# Read the file
with open("foo.txt", "r") as f:
    text = f.read()
i += 1  # Increment i

Who hurt you?
Imagine an equally experienced programmer is reading the code over your shoulder.

What you should write in comments is what you’d need to explain to them so they could understand your code.
“…basically, avoid comments. If your code needs a comment to be understood, it would be better to rewrite it so it’s easier to understand.”

Docstrings
def frobulate(s):
    """Apply Frob's Rule to turn the string into an incomprehensible integer. If the string is empty, raise an error."
    """
    ...
    ...
Comments are for someone trying to understand your source code.

Docstrings are for people who want to use your functions, telling them what the function is for and what to give it as input.
>>> import math
>>> help(math.log)
Help on built-in function log in module math:

log(...)
    log(x, [base=math.e])
    Return the logarithm of x to the given base.

    If the base not specified, returns the natural logarithm (base e) of x.

Python’s help shows you the docstring
Online documentation for Python and for libraries is almost always the docstrings from the source code!
Type hints
Unlike C or Java, Python doesn’t require you to specify the types of data a function takes as input or returns – but doing so is good for documentation and usability.
def *frobulate*(s: str) -> int:
    """Apply Frob's Rule to turn the string into an incomprehensible integer. If the string is empty, raise an error.
    """
    ...
I've posted example solutions to Assignment 0.
Also pay attention to how the starter code for Assignment 1 looks
Any questions?
On Wednesday, we’ll do an in-class exercise using Python – bring a computer!
Language models
The idea of a statistical *language model* (LM) is to compute the probability of a sequence of words. Why should we care about these probabilities?
Speech recognition

\[ P(I \text{ saw a van}) > P(\text{eyes awe of an}) \]

Spelling correction

The office is about fifteen minuets from my house.

\[ P(\text{about fifteen minutes from}) > P(\text{about fifteen minuets from}) \]

Machine translation

Translating The doctor recommended a cat scan,

\[ P(\text{La doctora recomendó una tomografía}) > P(\text{La doctora recomendó una exploración del gato}) \]

And more!
These are examples of computing

\[ P(W) = P(w_1, w_2, \ldots, w_n), \]

the probability of a sequence of words,

but language models also let us compute

\[ P(w_n \mid w_1, w_2, \ldots, w_{n-1}), \]

the probability of a word given some previous words.

Why would that be useful?
Hi! I'm heading to the store.
\[ P(\text{The other day I was walking along and saw a lizard}) \]
\[ = P(\text{The, other, day, I, was, walking, along, and, saw, a, lizard}) \]
\[ = \ldots \]
\[ P(\text{The other day I was walking along and saw a lizard})? \]
\[ = P(\text{The, other, day, I, was, walking, along, and, saw, a, lizard}) \]
\[ = \ldots \quad \text{Chain rule of probability!} \]
The chain rule of probability

Recall the definition of conditional probabilities,

\[ P(B \mid A) = \frac{P(A, B)}{P(A)} \]

which we can rewrite to get

\[ P(A, B) = P(A) \ P(B \mid A). \]
The chain rule of probability

If we have more variables, we get more terms, e.g.,

\[ P(A, B, C, D) = P(A) \ P(B \mid A) \ P(C \mid A, B) \ P(D \mid A, B, C) \]

In general,

\[ P(x_1, x_2, x_3, \ldots, x_n) = P(x_1) \ P(x_2 \mid x_1) \ P(x_3 \mid x_1, x_2) \cdots P(x_n \mid x_1, \ldots, x_{n-1}) \]
\[ P(\text{The other day I was walking along and saw a lizard})? \]
\[ = P(\text{The, other, day, I, was, walking, along, and, saw, a, lizard}) \]
\[ = P(\text{The}) \ P(\text{other} \mid \text{the}) \ P(\text{day} \mid \text{the other}) \ P(\text{I} \mid \text{The other day}) \cdots \]
\[ P(\text{The other day I was walking along and saw a lizard})? \]

\[ = P(\text{The}, \text{other}, \text{day}, \ I, \ \text{was, walking, along, and, saw, a, lizard}) \]

\[ = P(\text{The}) \cdot P(\text{other} \mid \text{the}) \cdot P(\text{day} \mid \text{the other}) \cdot P(I \mid \text{The other day}) \cdots \]

\[ = \text{🤔 How do we compute this?} \]
To estimate conditional probabilities, we use a text corpus that we’ve tokenized, and we do some counting!

\[ P(\text{lizard} \mid \text{The other day I was walking along and saw a}) = \frac{C(\text{The other day I was walking along and saw a lizard})}{C(\text{The other day I was walking along and saw a})} \]

\( C(x) \) is the count of how many times \( x \) occurs in the corpus.
In practice, we make a simplifying *Markov assumption* that we can predict the probability of a future event without looking too far into the past, e.g.,

\[
P(\text{lizard} \mid \text{the, other, day, I, was, walking, along, and, saw, a}) \approx P(\text{lizard} \mid \text{saw, a})
\]
We can estimate the true probabilities using $n$-grams – sequences of text that are always $n$ tokens long.
Colorless green ideas sleep furiously.
Colorless green ideas sleep furiously.

Unigrams:

<s>
Colorless
green
ideas
sleep
furiously
.
</s>
Colorless green ideas sleep furiously.

Bigrams:

<s> Colorless
Colorless green
green ideas
ideas sleep
sleep furiously
furiously .
. </s>
Colorless green ideas sleep furiously.

Trigrams:

<s> Colorless green
Colorless green ideas
green ideas sleep
ideas sleep furiously
sleep furiously .
furiously . </s>
Colorless green ideas sleep furiously.

4-grams:

<s> Colorless green ideas
Colorless green ideas sleep
green ideas sleep furiously
ideas sleep furiously .
sleep furiously . </s>
What’s the best value of $n$?

That is, how many previous words do we need?
Given any choice of $n$, are $n$-grams a sufficient model of language?
Language has *long-distance dependencies*:

The *computer / computers* 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.
Corpora and n-grams
All Our N-gram are Belong to You

THURSDAY, AUGUST 03, 2006
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 corpora containing at most a few billion words, we have been harnessing the vast power of Google’s datacenters and distributed processing infrastructure to process larger and larger training corpora. We found that there’s no data like more data, and scaled up the size of our data by one order of magnitude, and then another, and then one more - resulting in a training corpus of one trillion words from public Web pages.

We believe that the entire research community can benefit from access to such massive amounts of data. It will advance the state of the art, it will focus research in the promising direction of large-scale, data-driven approaches, and it will allow all research groups, no matter how large or small their computing resources, to play together. 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.

Watch for an announcement at the Linguistics Data Consortium (LDC), who will be distributing it soon, and then order your set of 6 DVDs. And let us hear from you - we’re excited to hear what you will do with the data, and we’re always interested in feedback about this dataset, or other potential datasets that might be useful for the research community.

Update (22 Sept. 2006): The LDC now has the data available in their catalog. The counts are as follows:

File size: nearly 34 GB compressed (original) text files.
All Our N-gram are Belong to You

THURSDAY, AUGUST 03, 2006
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 corpora containing at most a few billion words, we have been harnessing the vast power of Google's datacenters and distributed processing infrastructure to process larger and larger training corpora. We found that there's no data like more data, and scaled up the size of our data by one order of magnitude, and then another, and then one more - resulting in a training corpus of one trillion words from public Web pages.

We believe that the entire research community can benefit from access to such massive amounts of data. It will advance the state of the art, it will focus research in the promising direction of large-scale, data-driven approaches, and it will allow all research groups, no matter how large or small, their computing resources, to play together. 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.

Watch for an announcement at the Linguistics Data Consortium (LDC), who will be distributing it soon, and then order your set of 6 DVDs. And let us hear from you - we're excited to hear what you will do with the data, and we're always interested in feedback about this dataset, or other potential datasets that might be useful for the research community.

Update (22 Sept. 2006): The LDC now has the data available in their catalog. The counts are as follows:

File sizes approx. 34 GB compressed (original) text files.
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
The Google Books Ngram Viewer is optimized for quick inquiries into the usage of small sets of phrases. If you're interested in performing a large scale analysis on the underlying data, you might prefer to download a portion of the corpus yourself. Or all of it, if you have the bandwidth and space. We're happy to oblige.

These datasets were generated in July 2012 (Version 2) and July 2009 (Version 1); we will update these datasets as our book scanning continues, and the updated versions will have distinct and persistent version identifiers (20120761 and 20090715 for the current sets).

Each of the numbered links below will directly download a fragment of the corpus. In Version 2 the ngrams are grouped alphabetically (languages with non-Latin scripts were transliterated); in Version 1 the ngrams are partitioned into files of equal size. In addition, for each corpus we provide a file named total_counts, which records the total number of 1-grams contained in the books that make up the corpus. This file is useful for computing the relative frequencies of ngrams.

A summary of how the corpora were constructed can be found here. We explain it in greater depth here (Version 2) and here (Version 1). In both, we point out that we've included only ngrams that appear over 40 times across the corpus. That's why the sum of the 1-gram occurrences in any given corpus is smaller than the number given in the total_counts file.

File format: Each of the files below is compressed tab-separated. In Version 2 each line has the following format:

```
 ngram TAB year TAB match_count TAB volume_count NNEWLINE
```

As an example, here are the 3,000,000th and 3,000,001st lines from the a file of the English 1-grams (googlebooks-eng-all-1gram-20120701-a.gz):

```
circumvalate 1978 335 91
circumvalate 1979 291 91
```

The first line tells us that in 1978, the word "circumvalate" (which means "surround with a rampart or other fortification", in case you were wondering) occurred 335 times overall, in 91 distinct books of our sample.

The files vary widely in size because some patterns of letters are more common than others: the "na" file will be larger than the "ng" file since so many more words begin with "nta" than "ntg". Files with a letter followed by an underscore (e.g., _ADJ_) contain ngrams that begin with the first letter, but have an unusual second character.

We've included separate files for ngrams that start with punctuation or with other non-alphabetic characters. Finally, we have separate files for ngrams in which the first word is a part of speech tag (e.g., _ADJ_, _ADV_, _INF_, _VERB_).

In Version 1, the format is similar, but we also include the number of pages each ngram occurred on:

```
 ngram TAB year TAB match_count TAB page_count TAB volume_count NNEWLINE
```

Here's the 9,000,000th file from 0 of the English 5-grams (googlebooks-eng-all-5gram-20090615-0-csv.zip):

```
analysis is often described as 1991 1 1 1
```

In 1991, the phrase "analysis is often described as" occurred one time (that's the first 1), and on one page (the second 1), and in one book (the third 1). We do not provide page counts in Version 2 since we extract ngrams that span page boundaries.

The ngrams inside each file in Version 1 are sorted alphabetically and then chronologically. Note that the files themselves aren't ordered with respect to one another. A French two word phrase starting with 'en' will be in the middle of one of the French 2-gram files, but there's no way to know which without checking them all.

The format of the total_counts files are similar, except that the ngram field is absent and there is one triplet of values (match_count, page_count, volume_count) per year.

Usage: This compilation is licensed under a Creative Commons Attribution 3.0 Unported License.

English
Version 20120701

```
total_counts
1-grams 0 123456789 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
2-grams 0 123456789 ADJ ADP ADV CONJ DET NOUN NUM PRON PRT VFRB A AB AD AE AF AG AH AI AJ AK AL AM AN ANG AR AS AT AU AV AW AX AU B BB BC BD BE BF
```
## Corpora

Below are descriptions of the corpora that can be searched with the Google Books Ngram Viewer. All corpora were generated in July 2009, July 2012, and February 2020; we will update these corpora as our book scanning continues, and the updated versions will have distinct persistent identifiers. Books with low OCR quality and serials were excluded.

<table>
<thead>
<tr>
<th>Informal corpus name</th>
<th>Shorthand</th>
<th>Persistent identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>American English 2019</td>
<td>eng_us_2019</td>
<td>googlebooks-eng-us-20200217</td>
<td>Books predominantly in the English language that were published in the United States.</td>
</tr>
<tr>
<td>American English 2012</td>
<td>eng_us_2012</td>
<td>googlebooks-eng-us-all-20120701</td>
<td></td>
</tr>
<tr>
<td>American English 2009</td>
<td>eng_us_2009</td>
<td>googlebooks-eng-us-all-20090715</td>
<td></td>
</tr>
<tr>
<td>British English 2019</td>
<td>eng_gb_2019</td>
<td>googlebooks-eng-gb-20200217</td>
<td>Books predominantly in the English language that were published in Great Britain.</td>
</tr>
<tr>
<td>British English 2012</td>
<td>eng_gb_2012</td>
<td>googlebooks-eng-gb-all-20120701</td>
<td></td>
</tr>
<tr>
<td>British English 2009</td>
<td>eng_gb_2009</td>
<td>googlebooks-eng-gb-all-20090715</td>
<td></td>
</tr>
<tr>
<td>English</td>
<td>eng_2019</td>
<td>googlebooks-eng-</td>
<td></td>
</tr>
</tbody>
</table>
Estimating $n$-gram probabilities
We estimate the probabilities of $n$-grams using the \textit{maximum likelihood estimate} (MLE) – the estimate that maximizes the likelihood of the training data given the model.
For unigram probabilities,
that's the fraction of times the word occurs in the corpus:

\[ P(w_i) = \frac{C(w_i)}{|V|} \]

For bigram probabilities,
that's the number of times that word follows the other word divided
by the number of times the other word occurs in the corpus:

\[ P(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})} \]
For example, given the corpus

\[
<s> I am Sam </s>
\]

\[
<s> Sam I am </s>
\]

\[
<s> I do not like green eggs and ham </s>
\]

we can compute

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

and get these probabilities:

\[
P(I \mid <s>) = \frac{2}{3} = 0.67
\]

\[
P(\langle s\rangle \mid Sam) = \frac{1}{2} = 0.5
\]

\[
P(Sam \mid <s>) = \frac{1}{3} = 0.33
\]

\[
P(Sam \mid am) = \frac{1}{2} = 0.5
\]

\[
P(am \mid I) = \frac{2}{3} = 0.67
\]

\[
P(do \mid I) = \frac{1}{3} = 0.33
\]
Probability is assigned \textit{exactly} based on the $n$-gram count in the training corpus.

Anything not found in the training corpus gets probability 0.
Downside of MLE

Suppose you toss a coin 10 times and get 8 heads.

The MLE is that this coin comes down heads 8 times out of 10.

Would you agree?
Downside of MLE

Suppose you toss a coin 10 times and get 8 heads.

The MLE is that this coin comes down heads 8 times out of 10.

Would you agree?

This is the *prior belief* that influences beliefs even in the face of contradicting evidence.

Bayesian statistics measure degrees of belief:

Start with prior beliefs and update them in the face of evidence using *Bayes Theorem* – more on this next week!
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
Berkeley Restaurant Project: Bigram counts

From 9,222 sentences

<table>
<thead>
<tr>
<th></th>
<th>$w_1$</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></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></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></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></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></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>82</td>
<td>1</td>
</tr>
<tr>
<td>food</td>
<td></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></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></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>
<tr>
<td></td>
<td>want</td>
<td>to</td>
<td>eat</td>
<td>chinese</td>
<td>food</td>
<td>lunch</td>
<td>spend</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>------</td>
<td>----</td>
<td>------</td>
<td>---------</td>
<td>------</td>
<td>-------</td>
<td>-------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<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>

### Normalize by unigram 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>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>608</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>0</td>
<td>2</td>
<td>0</td>
<td>16</td>
<td>2</td>
<td>42</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>
</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>
### Berkeley Restaurant Project: Bigram probabilities

<table>
<thead>
<tr>
<th></th>
<th>(w_1)</th>
<th>(w_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td>0.002</td>
<td>0.0036</td>
</tr>
<tr>
<td>want</td>
<td>0.0022</td>
<td>0.0011</td>
</tr>
<tr>
<td>to</td>
<td>0.00083</td>
<td>0.0017</td>
</tr>
<tr>
<td>eat</td>
<td>0</td>
<td>0.0027</td>
</tr>
<tr>
<td>chinese</td>
<td>0.0063</td>
<td>0</td>
</tr>
<tr>
<td>food</td>
<td>0.014</td>
<td>0.014</td>
</tr>
<tr>
<td>lunch</td>
<td>0.0059</td>
<td>0</td>
</tr>
<tr>
<td>spend</td>
<td>0.0036</td>
<td>0</td>
</tr>
</tbody>
</table>

\(w_1\) and \(w_2\) are the probabilities of the bigrams formed by the words in the context. The table shows the conditional probabilities of the next word given the previous word for the Berkeley Restaurant Project dataset.
We use the bigram model to compute sentence probabilities:

\[
P(<s> \text{I want english food } </s>) \\
= P(I \mid <s>) \ P(\text{want } \mid I) \ P(\text{english } \mid \text{want}) \ P(\text{food } \mid \text{english}) \\
P(</s> \mid \text{food}) \\
= 0.00031
\]
As simple as they are, \( n \)-gram probabilities capture a range of interesting facts about language:

\[
\begin{align*}
P(\text{english} \mid \text{want}) &= 0.0011 \\
P(\text{chinese} \mid \text{want}) &= 0.0065 \\
P(\text{to} \mid \text{want}) &= 0.66 \\
P(\text{eat} \mid \text{to}) &= 0.28 \\
P(\text{food} \mid \text{to}) &= 0 \\
P(\text{want} \mid \text{spend}) &= 0 \\
P(i \mid <s>) &= 0.25
\end{align*}
\]

- **World knowledge; culture**
- **Syntactic preferences**
- **Discourse**
A practical concern

When programming, we handle probabilities in log space:

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

It’s nice that adding is faster than multiplying, but the main reason is that it avoids underflow.

This will be true for the rest of the class!
Numeric underflow:

\[
\begin{align*}
    a &= 1 \times 10^{-10} \\
    b &= 1 \times 10^{-90} \\
    c &= 1 \times 10^{-30} \\
    d &= 5 \times 10^{-130} \\
    e &= 1 \times 10^{-40} \\
    f &= 1 \times 10^{-100}
\end{align*}
\]

\[a \times b \times c \times d \times e \times f \rightarrow 0.0\]

But, using log-space math:

```python
from math import log

log(a) + log(b) + log(c) + log(d) + log(e) + log(f) → -919.4245992851843
```
Next time

Smoothing and generalization

How do we know if a language model is good?

Text generation using language models

Bring a computer!
Acknowledgments

This class incorporates material from:

Carolyn Anderson, Wellesley College
Nancy Ide, Vassar College
Katie Keith, Williams College
Jurafsky & Martin, *Speech and Language Processing*, 3rd ed. draft