NLTK and Lexical Information

Marina Sedinkina
- Folien von Desislava Zhekova -

CIS, LMU
marina.sedinkina@campus.lmu.de

December 12, 2017

Outline

NLTK Web

- created in 2001 in the University of Pennsylvania
- as part of a computational linguistics course in the Department of Computer and Information Science

NLTK 3.0 documentation

NEXT | MODULES | INDEX

Natural Language Toolkit

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

Thanks to a hands-on guide introducing programming fundamentals alongside topics in computational linguistics, plus comprehensive API documentation, NLTK is suitable for linguists, engineers, students, educators, researchers, and industry users alike. NLTK is available for Windows, Mac OS X, and Linux. Best of all, NLTK is a free, open source, community-driven project.

NLTK has been called "a wonderful tool for teaching, and working in, computational linguistics using Python," and "an amazing library to play with natural language."

Natural Language Processing with Python provides a practical introduction to programming for language processing. Written by the creators of NLTK, it guides the reader through the fundamentals of writing Python programs, working with corpora, categorizing text, analyzing linguistic structure, and more. The book is being updated for Python 3 and NLTK 3. (The original Python 2 version is still available at http://nltk.org/book_1ed.)

TABLE OF CONTENTS

- NLTK News
- Installing NLTK
- Installing NLTK Data
- Contribute to NLTK
- FAQ
- Wiki
- API
- HOWTO

SEARCH

Go

Enter search terms or a module, class or function name.

NLP Tasks

Language processing task	NLTK modules	Functionality
Accessing corpora	nltk.corpus	Standardized interfaces to corpora and lexicons
String processing	nltk.tokenize, nltk.stem	Tokenizers, sentence tokenizers, stemmers
Collocation discovery	nltk.collocations	t-test, chi-squared, point-wise mutual information
Part-of-speech tagging	nltk.tag	n-gram, backoff, Brill, HMM, TnT
Classification	nltk.classify, nltk.cluster	Decision tree, maximum entropy, naive Bayes, EM, k-means
Chunking	nltk.chunk	Regular expression, n-gram, named entity
Parsing	nltk.parse	Chart, feature-based, unification, probabilistic, dependency
Semantic interpretation	nltk.sem, nltk.inference	Lambda calculus, first-order logic, model checking
Evaluation metrics	nltk.metrics	Precision, recall, agreement coefficients
Probability and estimation	nltk.probability	Frequency distributions, smoothed probability distributions
Applications	nltk.app, nltk.chat	Graphical concordancer, parsers, WordNet browser, chatbots

NLTK book examples

1. open the Python interactive shell `python3`
2. execute the following commands:

   >>> import nltk

   >>> nltk.download()
3. choose"Everything used in the NLTK Book"

```
>>> from nltk.book import *
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
>>>
```

NLTK book.py

Source code: https://github.com/nltk/nltk/blob/develop/nltk/book.py

```
1  from __future__ import print_function
2  from nltk.corpus import (gutenberg, genesis, inaugural, nps_chat,
3                           webtext, treebank, wordnet)
4  from nltk.text import Text
5  from nltk.probability import FreqDist
6  from nltk.util import bigrams
7
8  print("*** Introductory Examples for the NLTK Book ***")
9  print("Loading text1, ..., text9 and sent1, ..., sent9")
10 print("Type the name of the text or sentence to view it.")
11 print("Type: 'texts()' or 'sents()' to list the materials.")
12
13 text1 = Text(gutenberg.words('melville-moby_dick.txt'))
14 print("text1:", text1.name)
15
16 text2 = Text(gutenberg.words('austen-sense.txt'))
17 print("text2:", text2.name)
18 ...
```

NLTK book examples

```
>>> text1
<Text: Moby Dick by Herman Melville 1851>
>>> text2
<Text: Sense and Sensibility by Jane Austen 1811>
>>>
```

Great, a couple of texts, but what to do with them? Well, let's explore them a bit!

nltk.text.Text

```
1  from nltk.corpus import gutenberg
2  from nltk.text import Text
3
4  moby = Text(gutenberg.words("melville-moby_dick.txt"))
5  print(moby.concordance("Moby"))
```

nltk.text.Text:

```
1  class nltk.text.Text(tokens, name=None)
2      collocations(num=20, window_size=2)
3      common_contexts(words, num=20)
4      concordance(word, width=79, lines=25)
5      count(word)
6      dispersion_plot(words)
7      findall(regexp)
8      index(word)
9      similar(word, num=20)
10     vocab()
```

Concordances

A **concordance** is the list of all occurrences of a given word together with its context.

Concordances

```
>>> text1.concordance("monstrous")
Building index...
Displaying 11 of 11 matches:
ong the former , one was of a most monstrous size . ... This came towards us ,
ON OF THE PSALMS . " Touching that monstrous bulk of the whale or ork we have r
ll over with a heathenish array of monstrous clubs and spears . Some were thick
d as you gazed , and wondered what monstrous cannibal and savage could ever hav
that has survived the flood ; most monstrous and most mountainous ! That Himmal
they might scout at Moby Dick as a monstrous fable , or still worse and more de
th of Radney .'" CHAPTER 55 Of the monstrous Pictures of Whales . I shall ere l
ing Scenes . In connexion with the monstrous pictures of whales , I am strongly
ere to enter upon those still more monstrous stories of them which are to be fo
```

Concordances

Contexts in which monstrous occurs:

```
the ___ pictures
the ___ size
```

Concordances

Contexts in which monstrous occurs:

```
the   ___   pictures
most  ___   size
```

???

So, what other words may have the same context?

Concordances

```
>>> text1.similar("monstrous")
Building word-context index...
subtly impalpable pitiable curious imperial perilous trustworthy
abundant untoward singular lamentable few maddens horrible loving lazy
mystifying christian exasperate puzzled
```

considerably different usage

```
>>> text2.similar("monstrous")
Building word-context index...
very exceedingly so heartily a great good amazingly as sweet
remarkably extremely vast
```

```
>>> from nltk.book import *
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
>>>
```

Concordances

```
>>> text2.common_contexts(["monstrous", "very"])
be_glad am_glad a_pretty is_pretty a_lucky
>>>
```

Concordances

```
>>> text2.common_contexts(["monstrous", "very"])
be_glad am_glad a_pretty is_pretty a_lucky
>>>
```

But wait! **be monstrous glad** ?!

Concordances

Apparently Jane Austen does use it this way:

> "Nay," cried Mrs. Jennings, "I am sure I shall be monstrous glad of Miss Marianne's company, whether Miss Dashwood will go or not, only the more the merrier say I, and I thought it would be more comfortable for them to be together; because, if they got tired of me, they might talk to one another, and laugh at my old ways behind my back. But one or the other, if not both of them, I must have. Lord bless me! how do you think I can live poking by myself, I who have been always used till this winter to have Charlotte with me. Come, Miss Marianne, let us strike hands upon the bargain, and if Miss Dashwood will change her mind by and bye, why so much the better."

Sense and Sensibility – Chapter 25

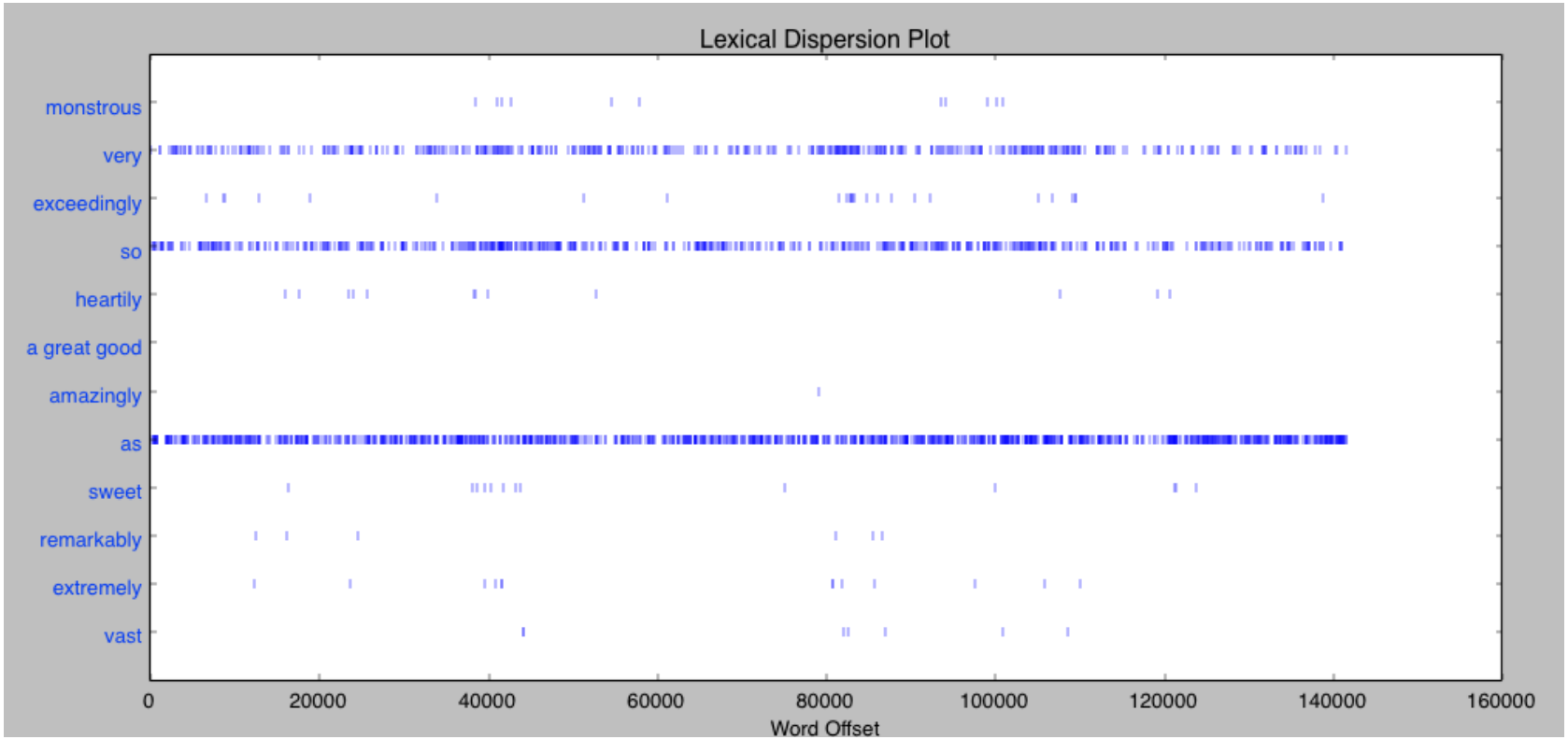
Lexical Dispersion Plots

Location of a word in the text can be displayed using a **dispersion plot**

Lexical Dispersion Plots

```
1 from nltk.book import *
2
3 list = ["monstrous", "very", "exceedingly", "so", "heartily","a
      great good", "amazingly", "as", "sweet", "remarkably", "
      extremely", "vast"]
4
5 text4.dispersion_plot(list)
```

Lexical Dispersion Plots



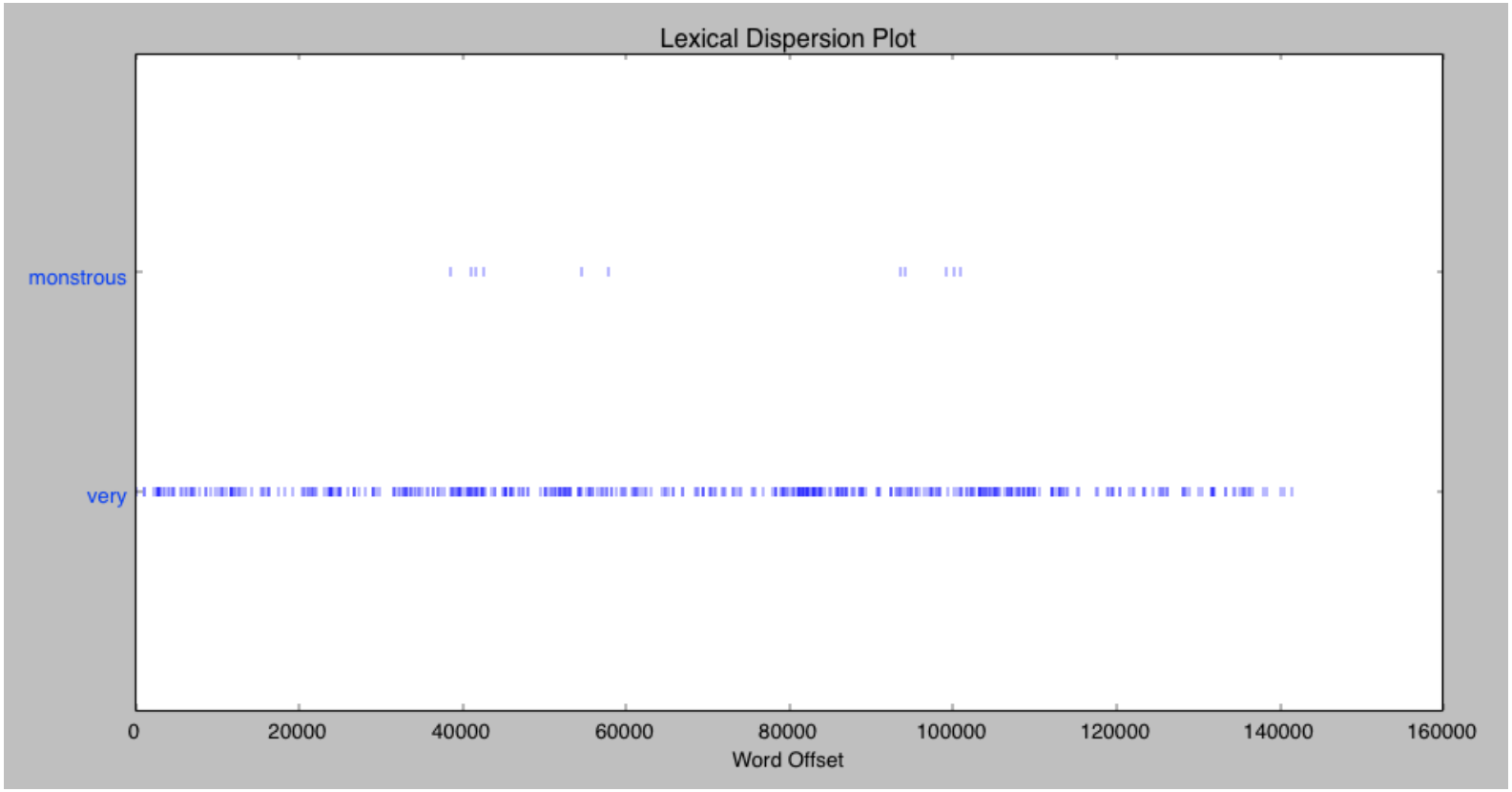
Lexical Dispersion Plots

For most of the visualization and plotting from the NLTK book you would need to install additional modules:

- NumPy – a scientific computing library with support for multidimensional arrays and linear algebra, required for certain probability, tagging, clustering, and classification tasks
  `sudo pip3 install -U numpy`
- Matplotlib – a 2D plotting library for data visualization, and is used in some of the book's code samples that produce line graphs and bar charts
  `sudo pip3 install -U matplotlib`

Lexical Dispersion Plots

```
>>> text2.common_contexts(["monstrous", "very"])
be_glad am_glad a_pretty is_pretty a_lucky
>>>
```



Lexical Dispersion Plots

???

Can you think of a good usage for lexical dispersion plots?

Diachronic vs Synchronic Language Studies

- Language data may contain information about the time in which it has been elicited

Diachronic vs Synchronic Language Studies

- Language data may contain information about the time in which it has been elicited
- This information provides capability to perform **diachronic language studies**.

Diachronic vs Synchronic Language Studies

- Language data may contain information about the time in which it has been elicited
- This information provides capability to perform **diachronic language studies**.
- **Diachronic language study** is the exploration of natural language when time is considered as a factor

Diachronic vs Synchronic Language Studies

- Language data may contain information about the time in which it has been elicited
- This information provides capability to perform **diachronic language studies**.
- **Diachronic language study** is the exploration of natural language when time is considered as a factor
- The opposite approach is called **synchronic language study**.

Diachronic vs Synchronic Language Studies

For example:

- **synchronic** – extracting the occurrence of words in the full corpus
- **diachronic** – extracting the occurrence of words comparing the results over time

Inaugural Address

The Inaugural Address is the first speech that each newly elected president in the US holds.

Inaugural Address

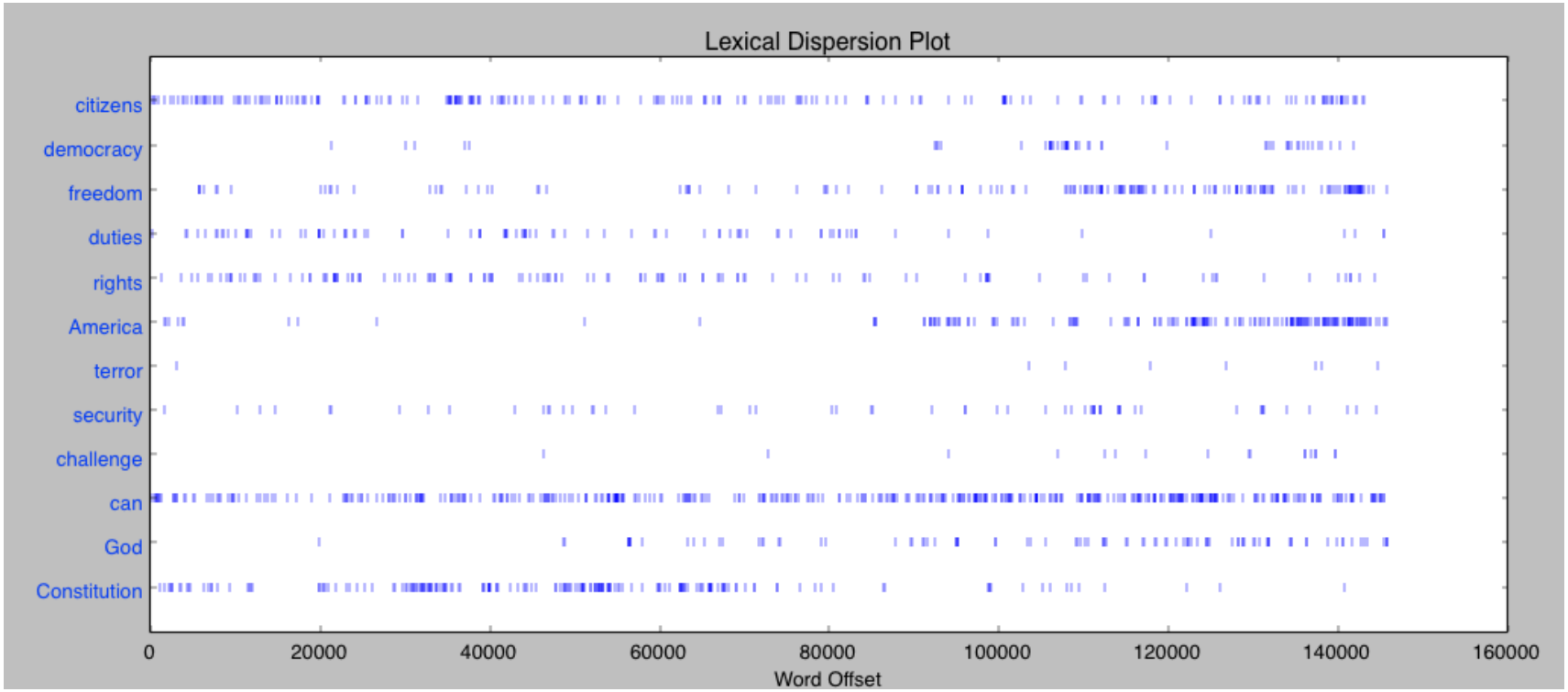
The Inaugural Address is the first speech that each newly elected president in the US holds.

```
>>> from nltk.book import *
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
>>>
```

The Inaugural Address Corpus

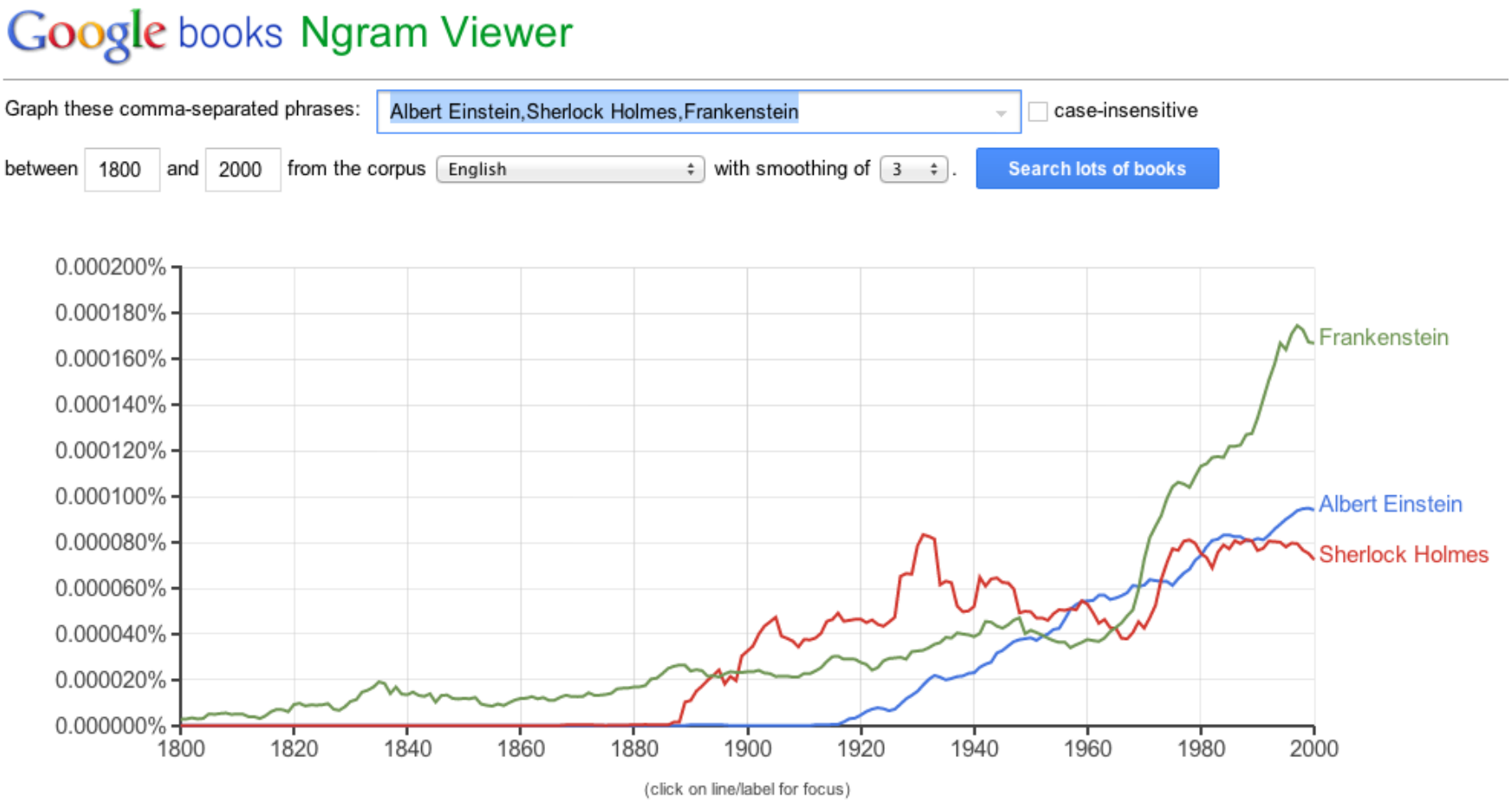
1789-Washington.txt
1793-Washington.txt
1797-Adams.txt
1801-Jefferson.txt
1805-Jefferson.txt
1809-Madison.txt
1813-Madison.txt
1817-Monroe.txt
1821-Monroe.txt
1825-Adams.txt
1829-Jackson.txt
1833-Jackson.txt
1837-VanBuren.txt
1841-Harrison.txt
1845-Polk.txt
1849-Taylor.txt
1853-Pierce.txt
1857-Buchanan.txt
1861-Lincoln.txt
1865-Lincoln.txt
1869-Grant.txt
1873-Grant.txt
1877-Hayes.txt
1881-Garfield.txt
1885-Cleveland.txt
1889-Harrison.txt
1893-Cleveland.txt
1897-McKinley.txt
1901-McKinley.txt
1905-Roosevelt.txt
1909-Taft.txt
1913-Wilson.txt
1917-Wilson.txt
1921-Harding.txt
1925-Coolidge.txt
1929-Hoover.txt
1933-Roosevelt.txt
1937-Roosevelt.txt
1941-Roosevelt.txt
1945-Roosevelt.txt
1949-Truman.txt
1953-Eisenhower.txt
1957-Eisenhower.txt
1961-Kennedy.txt
1965-Johnson.txt
1969-Nixon.txt
1973-Nixon.txt
1977-Carter.txt
1981-Reagan.txt
1985-Reagan.txt
1989-Bush.txt
1993-Clinton.txt
1997-Clinton.txt
2001-Bush.txt
2005-Bush.txt
2009-Obama.txt

Diachronic Studies via Lexical Dispersion Plots



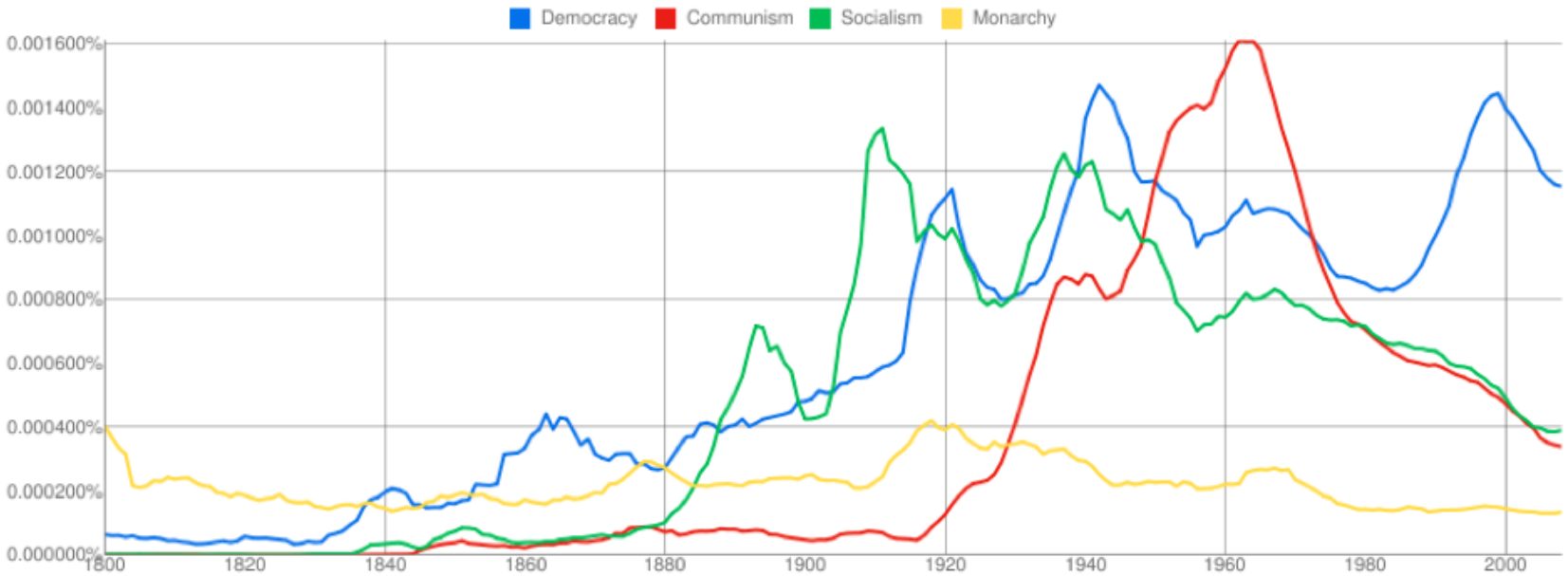
Diachronic Studies and Google

https://books.google.com/ngrams



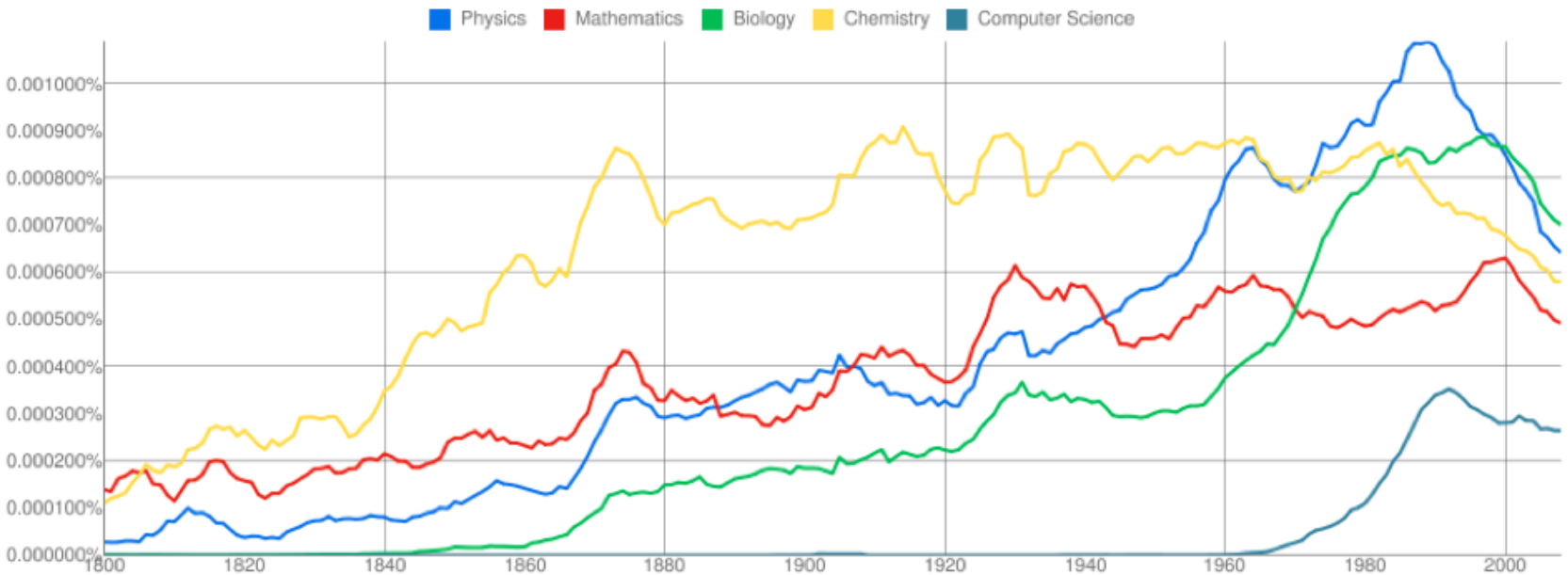
Diachronic Studies and Google

Mirroring social and economic systems and forms of government:



Diachronic Studies and Google

Scientific fields of study:



Basic Text Statistics

- `len(text1)` – extract the number of **tokens** in text1
- `len(set(text1))` – extract the number of unique tokens (**types**) in text1 (**vocabulary of text1**). You can also use `nltk.text.Text.vocab()`.
- `sorted(set(text1))` – extract the number of item types in text1 in sorted order

Basic Text Statistics

???

What does the following code calculate and print out?

```
1 from nltk.book import *
2
3 print(len(text3) / len(set(text3)))
```

Basic Text Statistics

???

What does the following code calculate and print out?

```
1 from nltk.book import *
2
3 print(len(text3) / len(set(text3)))
4
5 # prints 16.050197203298673
```

Answer

It measures the lexical richness of text3 from the nltk.book collection.

Basic Text Statistics

- measuring how often a word occurs in a text
- computing what percentage of the text is taken up by a specific word

```
>>> text3.count("smote")
5
>>> 100 * text4.count('a') / len(text4)
1.464301643938312
>>>
```

Brown Corpus Stats

Lexical diversity of various genres in the Brown Corpus:

Genre	Tokens	Types	Lexical diversity
skill and hobbies	82345	11935	6.9
humor	21695	5017	4.3
fiction: science	14470	3233	4.5
press: reportage	100554	14394	7.0
fiction: romance	70022	8452	8.3
religion	39399	6373	6.2

Basic Text Statistics

???

What output do you expect from the following code?

```
1  saying = ["After", "all", "is", "said", "and", "done", "
       more", "is", "said", "than", "done"]
2
3  tokens = set(saying)
4  print(tokens)
5
6  tokens = sorted(tokens)
7  print(tokens)
8
9  print(tokens[-2:])
```

Basic Text Statistics

???

What output do you expect from the following code?

```
1  saying = ["After", "all", "is", "said", "and", "done", "
       more", "is", "said", "than", "done"]
2
3  tokens = set(saying)
4  print(tokens)
5  # {"more", "than", "is", "After", "and", "done", "all", "
       said"}
6
7  tokens = sorted(tokens)
8  print(tokens)
9  # ["After", "all", "and", "done", "is", "more", "said", "
       than"]
10
11 print(tokens[-2:])
```

Using List Comprehensions for Filtering

You can make good use of list comprehensions to extract specific information out of the texts.

```
>>> V = set(text1)
>>> long_words = [w for w in V if len(w) > 15]
>>> sorted(long_words)
['CIRCUMNAVIGATION', 'Physiognomically', 'apprehensiveness', 'cannibalistically',
'characteristically', 'circumnavigating', 'circumnavigation', 'circumnavigations',
'comprehensiveness', 'hermaphroditical', 'indiscriminately', 'indispensableness',
'irresistibleness', 'physiognomically', 'preternaturalness', 'responsibilities',
'simultaneousness', 'subterraneousness', 'supernaturalness', 'superstitiousness',
'uncomfortableness', 'uncompromisedness', 'undiscriminating', 'uninterpenetratingly']
>>>
```

Using List Comprehensions for Filtering

```
>>> fdist5 = FreqDist(text5)
>>> sorted([w for w in set(text5) if len(w) > 7 and fdist5[w] > 7])
['#14-19teens', '#talkcity_adults', '((((((((((', '........', 'Question',
'actually', 'anything', 'computer', 'cute.-ass', 'everyone', 'football',
'innocent', 'listening', 'remember', 'seriously', 'something', 'together',
'tomorrow', 'watching']
>>>
```

Frequency Distributions

>>> from nltk import FreqDist

```
>>> fdist1 = FreqDist(text1) ❶
>>> fdist1 ❷
<FreqDist with 260819 outcomes>
>>> vocabulary1 = fdist1.keys() ❸
>>> vocabulary1[:50] ❹
[',', 'the', '.', 'of', 'and', 'a', 'to', ';', 'in', 'that', "'", '-',
'his', 'it', 'I', 's', 'is', 'he', 'with', 'was', 'as', '"', 'all', 'for',
'this', '!', 'at', 'by', 'but', 'not', '--', 'him', 'from', 'be', 'on',
'so', 'whale', 'one', 'you', 'had', 'have', 'there', 'But', 'or', 'were',
'now', 'which', '?', 'me', 'like']
>>> fdist1['whale']
906
>>>
```

Frequency Distributions

- **hapaxes**: words that only occur once in the text
- use NLTK to extract these: `fdist1.hapaxes()`
- hapaxes in the Inaugural Address: `...'Brutus', 'Budapest', 'Bureau', 'Burger', 'Burma'...`

Frequency Distributions

Frequency distributions:

- **differ based on the text** they have been calculated on

Frequency Distributions

Frequency distributions:

- **differ based on the text** they have been calculated on
- may also **differ based on other factors: e.g. categories** of a text (genre, topic, author, etc.)

Frequency Distributions

Frequency distributions:

- **differ based on the text** they have been calculated on
- may also **differ based on other factors: e.g. categories** of a text (genre, topic, author, etc.)
- we can maintain separate frequency distributions for each category.

Conditional Frequency Distributions

Conditional frequency distributions:

- are collections of frequency distributions
- each frequency distribution is measured for a different **condition** (e.g. category of the text)

Condition: News

the	𝍸 𝍸 𝍸 ||||
cute	
Monday	𝍸 ||||
could	|
will	𝍸 |||

Condition: Romance

the	𝍸 𝍸 |||
cute	|||
Monday	|
could	𝍸 𝍸 𝍸
will	||||

Conditional Frequency Distributions

Conditional frequency distributions allow us to:

- focus on specific categories
- study systematic differences between the categories

Conditional Frequency Distributions

- frequency distribution counts observable events
- conditional frequency distribution needs to pair each event with a condition `(condition, event)`

```
1 >>> text = ["The", "Fulton", "County", "Grand", ... ]
2 >>> pairs = [("news", "The"),("news", "Fulton"), ... ]
```

Conditional Frequency Distributions

```
>>> genre_word = [(genre, word)
...     for genre in ["news", "romance"]
...     for word in brown.words(categories=genre)]
>>> len(genre_word)
170576

>>> genre_word[:2]
[("news", "The"), ("news", "Fulton")]
>>> genre_word[-2:]
[("romance", "afraid"), ("romance", "not")]
```

Conditional Frequency Distributions

Then you can pass the list to `ConditionalFreqDist()`:

```
>>> from nltk import ConditionalFreqDist
>>> cfd = nltk.ConditionalFreqDist(genre_word)
>>> cfd
<ConditionalFreqDist with 2 conditions>
>>> cfd.conditions()
["news", "romance"]
```

Conditional Frequency Distributions

```
>>> cfd["news"]
<FreqDist with 100554 outcomes>
>>> cfd["romance"]
<FreqDist with 70022 outcomes>
>>> list(cfd["romance"])
[",", ".", "the", "and", "to", "a", "of", "was",
    "I", "in", "he", "had", "?", "her", "that",
   "it", "his", "she", "with", "you", "for", "at
   ", "He", "on", "him", "said", "!", "—", "be"
   , "as", ";", "have", "but", "not", "would", "
   She", "The", ... ]
>>> cfd["romance"]["could"]
193
```

Conditional Frequency Distributions

```
1 import nltk
2 from nltk.corpus import inaugural
3
4 cfd = nltk.ConditionalFreqDist((w, fileid[:4])
5     for fileid in inaugural.fileids()
6     for w in inaugural.words(fileid)
7     for target in ["american", "citizen"]
8     if w.lower().startswith(target))
```

???

How many conditions will be generated here?

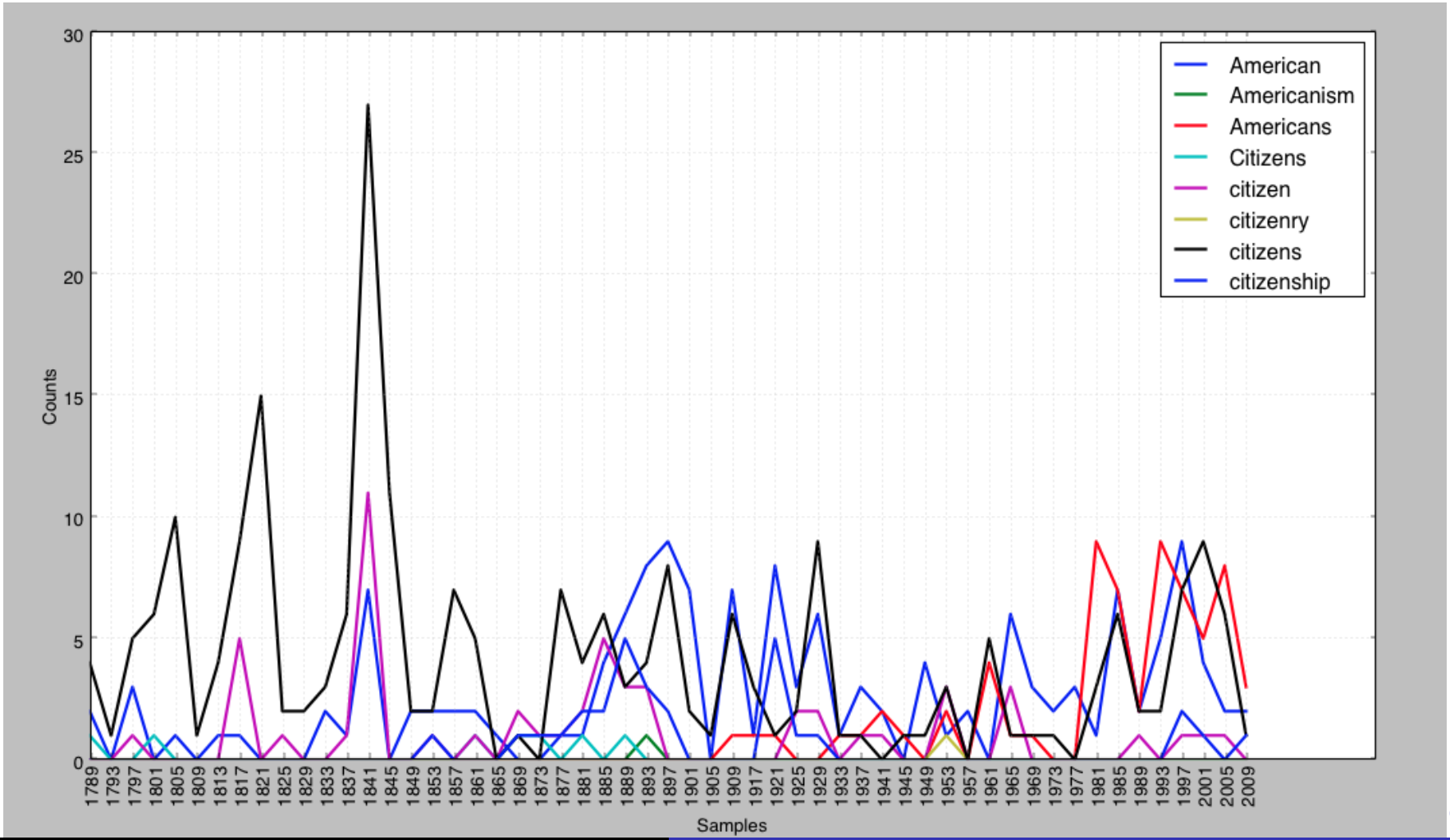
Conditional Frequency Distributions

```
1  import nltk
2  from nltk.corpus import inaugural
3
4  cfd = nltk.ConditionalFreqDist((w, fileid[:4])
5      for fileid in inaugural.fileids()
6      for w in inaugural.words(fileid)
7      for target in ["american", "citizen"]
8      if w.lower().startswith(target))
9  print(cfd.conditions())
10 # ['American', 'Americanism', 'Americans', 'Citizens', '
       citizen', 'citizenry', 'citizens', 'citizenship']
```

Conditional Frequency Distributions

Visualize cfd with: cfd.plot()

Conditional Frequency Distributions



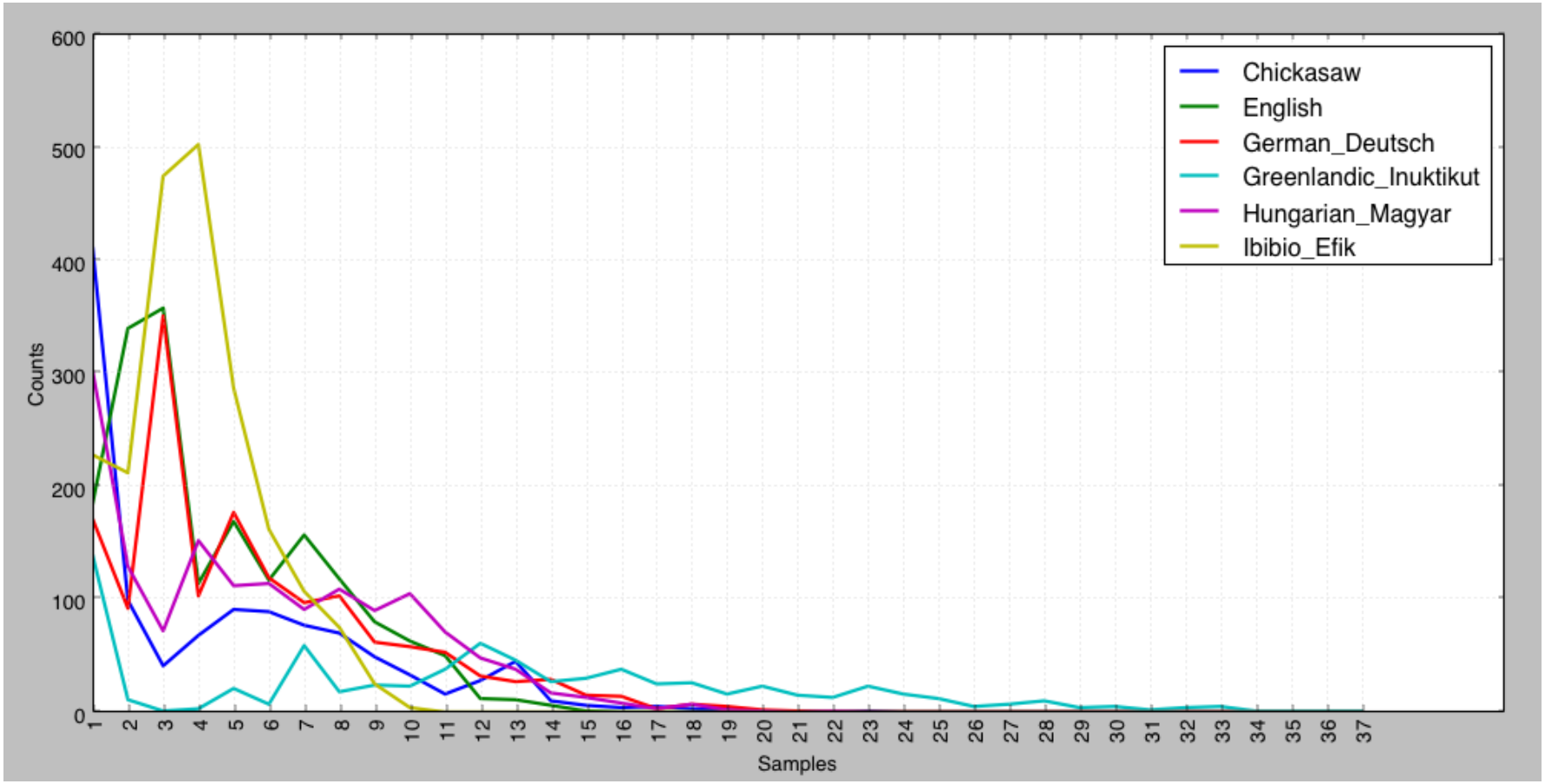
Conditional Frequency Distributions

`udhr` – Universal Declaration of Human Rights Corpus: the declaration of human rights in more than 300 languages.

```
1 from nltk.corpus import udhr
2
3 languages = ["Chickasaw", "English", "
    German_Deutsch", "Greenlandic_Inuktikut", "
    Hungarian_Magyar", "Ibibio_Efik"]
4 cfd = nltk.ConditionalFreqDist((lang, len(word))
5 for lang in languages
6 for word in udhr.words(lang + "-Latin1"))
```

Conditional Frequency Distributions

```
cfd.plot()
```



Conditional Frequency Distributions

```
1  cfd.tabulate(conditions=["English", "
       German_Deutsch"], samples=range(5))
2
3  #                    0    1    2    3    4
4  #            English    0  185  340  358  114
5  #  German_Deutsch       0  171   92  351  103
```

Conditional Frequency Distributions

Example	Description
`cfdist = ConditionalFreqDist(pairs)`	Create a conditional frequency distribution from a list of pairs
`cfdist.conditions()`	Alphabetically sorted list of conditions
`cfdist[condition]`	The frequency distribution for this condition
`cfdist[condition][sample]`	Frequency for the given sample for this condition
`cfdist.tabulate()`	Tabulate the conditional frequency distribution
`cfdist.tabulate(samples, conditions)`	Tabulation limited to the specified samples and conditions
`cfdist.plot()`	Graphical plot of the conditional frequency distribution
`cfdist.plot(samples, conditions)`	Graphical plot limited to the specified samples and conditions
`cfdist1 < cfdist2`	Test if samples in `cfdist1` occur less frequently than in `cfdist2`

Collocations and Bigrams

- A **collocation** is a sequence of words that occur together unusually often.

Collocations and Bigrams

- A **collocation** is a sequence of words that occur together unusually often.
- Thus *red wine* is a collocation, whereas *the wine* is not.

Collocations and Bigrams

- A **collocation** is a sequence of words that occur together unusually often.
- Thus *red wine* is a collocation, whereas *the wine* is not.
- Collocations are resistant to substitution with words that have similar senses; for example, *maroon wine* sounds very odd.

Collocations and Bigrams

```
>>> text4.collocations()
Building collocations list
United States; fellow citizens; years ago; Federal Government; General
Government; American people; Vice President; Almighty God; Fellow
citizens; Chief Magistrate; Chief Justice; God bless; Indian tribes;
public debt; foreign nations; political parties; State governments;

>>> text8.collocations()
Building collocations list
medium build; social drinker; quiet nights; long term; age open;
financially secure; fun times; similar interests; Age open; poss
rship; single mum; permanent relationship; slim build; seeks lady;
Late 30s; Photo pls; Vibrant personality; European background; ASIAN
LADY; country drives
```

Collocations and Bigrams

- Bigrams are a list of word pairs extracted from a text
- Collocations are essentially just frequent bigrams

```
>>> from nltk import bigrams
>>> list(bigrams(["more", "is","said", "than", "done"]))

>>> [('more', 'is'), ('is', 'said'), ('said', 'than'), ('
    than', 'done')]

>>> from nltk import trigrams
>>> list(trigrams(["more", "is","said", "than", "done"]))

>>> [('more', 'is', 'said'), ('is', 'said', 'than'), ('said
    ', 'than', 'done')]
```

Generating Random Text with Bigrams

```
>>> sent = ["In", "the", "beginning", "God", "created", "
    the", "heaven", "and", "the", "earth", "."]
>>> [y for y in nltk.bigrams(sent)]

[("In", "the"), ("the", "beginning"), ("beginning", "God"),
     ("God", "created"), ("created", "the"), ("the", "
    heaven"), ("heaven", "and"), ("and", "the"), ("the", "
    earth"), ("earth", ".")]
```

Generating Random Text with Bigrams

```
1  import nltk
2
3  text = nltk.corpus.genesis.words("english-kjv.txt")
4  bigrams = nltk.bigrams(text)
5  cfd = nltk.ConditionalFreqDist(bigrams)
6
7  print(cfd.conditions())
8  >>> ['In', 'the', 'beginning', 'God', 'created', ... ]
```

We treat each word as a condition, and for each one we create a frequency distribution over the following words

Generating Random Text with Bigrams

```
1  import nltk
2
3  text = nltk.corpus.genesis.words("english-kjv.txt")
4  bigrams = nltk.bigrams(text)
5  cfd = nltk.ConditionalFreqDist(bigrams)
6
7  print(list(cfd["living"]))
8  >>>['creature', 'thing', 'soul', '.', 'substance', ',']
```

6 words that have condition "living": living creature, living thing, living soul, ...

Generating Random Text with Bigrams

```
1  import nltk
2
3  text = nltk.corpus.genesis.words("english-kjv.txt")
4  bigrams = nltk.bigrams(text)
5  cfd = nltk.ConditionalFreqDist(bigrams)
6
7  print(list(cfd["living"]))
8  >>>['creature', 'thing', 'soul', '.', 'substance', ',']
9
10 print(list(cfd["living"].values()))
11 >>> [7, 4, 1, 1, 2, 1]
```

living creature = 7 times, living thing = 4 times, ...

Generating Random Text with Bigrams

```
1  import nltk
2
3  text = nltk.corpus.genesis.words("english-kjv.txt")
4  bigrams = nltk.bigrams(text)
5  cfd = nltk.ConditionalFreqDist(bigrams)
6
7  print(list(cfd["living"]))
8  >>>['creature', 'thing', 'soul', '.', 'substance', ',']
9
10 print(list(cfd["living"].values()))
11 >>> [7, 4, 1, 1, 2, 1]
12
13 print(cfd["living"].max())
14 >>> creature
```

Most likely token in that context is "creature"

Generating Random Text with Bigrams

```
1  import nltk
2
3  def generate_model(cfdist, word, num=15):
4      for i in range(num):
5          print(word, end=' ')
6          word = cfdist[word].max()
7
8
9  text = nltk.corpus.genesis.words("english-kjv.txt")
10 bigrams = nltk.bigrams(text)
11 cfd = nltk.ConditionalFreqDist(bigrams)
12
13 generate_model(cfd, 'living')
14 >>> living creature that he said , and the land of the land
       of the land
```

Lexical Dispersion Plots

???

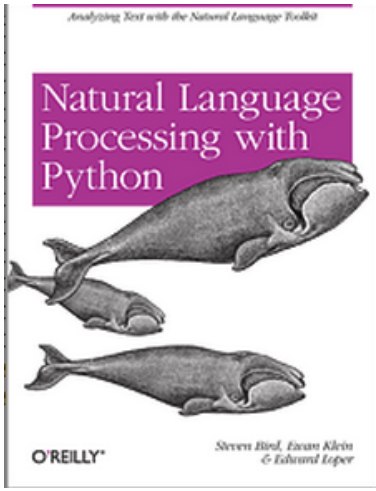
Can you think of a good usage for natural language generation???

Lexical Dispersion Plots

Natural language generation applications:

- document summarization (e.g. of databases, business data, medical records)
- text simplification
- sentence compression
- question answering
- textual weather forecasts from weather data

References

- 

  http://www.nltk.org/book/
- https://github.com/nltk/nltk