

NLTK and Lexical Information

Florian Fink

- Folien von Desislava Zhekova -

CIS, LMU

January 12, 2021

Outline

- ▶ 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

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

NLP Tasks

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

see documentation by typing:

```
1 >>>help(Text)
2
3 class nltk.text.Text(tokens, name=None)
4     collocations(num=20, window_size=2)
5     common_contexts(words, num=20)
6     concordance(word, width=79, lines=25)
7     count(word)
8     dispersion_plot(words)
9     findall(regexp)
10    index(word)
11    similar(word, num=20)
12    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
a ___ size

Concordances

Contexts in which **monstrous** occurs:

the	___	pictures
a	___	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.”

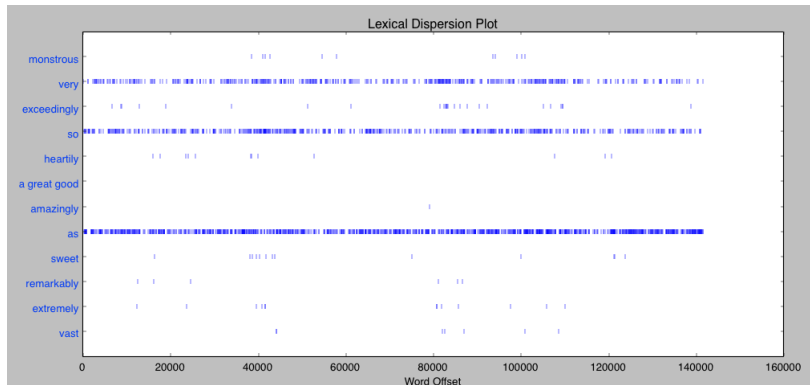
Sense and Sensibility – Chapter 25

Lexical Dispersion Plots

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

```
1 from nltk.book import *
2
3 list = ["monstrous", "very", "exceedingly", "so", "heartily", "a
         great good", "amazingly", "as", "sweet", "remarkably", "
         extremely", "vast"]
4
5 text2.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:

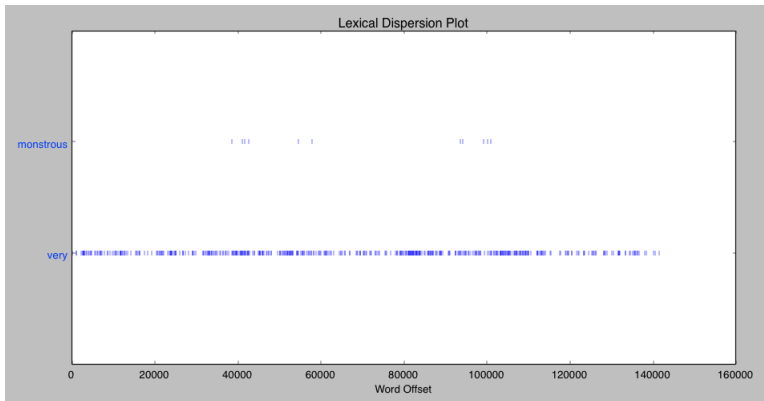
- ▶ NumPy – a scientific computing library with support for multidimensional arrays and linear algebra, required for certain probability, tagging, clustering, and classification tasks
- ▶ Matplotlib – a 2D plotting library for data visualization, and is used in some of the book's code samples that produce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

```
sudo pip3 install -U numpy
```

```
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

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.

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

Inaugural Address

```
1 from nltk.corpus import inaugural
2 >>> inaugural.fileids()
3 [ '1789-Washington.txt ', '1793-Washington.txt ', ... ]
4 >>> inaugural.words('1789-Washington.txt ')
5 >>> [ 'Fellow ', '- ', 'Citizens ', 'of ', 'the ', 'Senate ', ... ]
```

```
>>> 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
>>>
```


Diachronic Studies and Google

`https://books.google.com/ngrams`

Google books Ngram Viewer

Graph these comma-separated phrases: ☐ case-insensitive

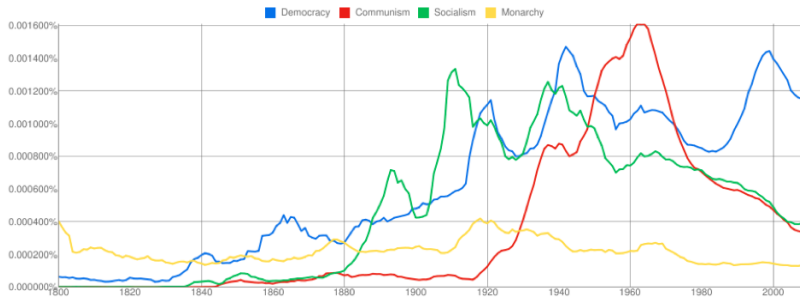
between and from the corpus with smoothing of .

[Search lots of books](#)



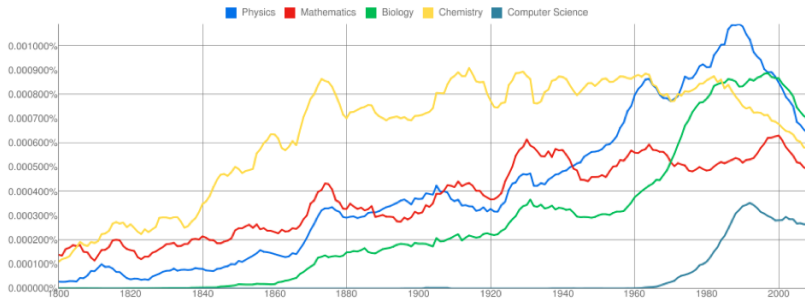
Diachronic Studies and Google

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()`.
- ▶ `len(text3) / len(set(text3))` – **lexical diversity**

Basic Text Statistics

It measures the lexical diversity of text3 from the nltk.book collection:

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

Brown Corpus Stats

- ▶ The Brown Corpus was the first million-word electronic corpus of English
- ▶ created in 1961 at Brown University
- ▶ contains text from 500 sources
- ▶ the sources have been categorized by genre
- ▶ a convenient resource for studying systematic differences between genres, a kind of linguistic inquiry known as **stylistics**.

```
1 from nltk.corpus import brown
2
3 >>> brown.categories()
4 >>> ["adventure", "belles_lettres", "editorial", "fiction",
5      "government", "hobbies", "humor", "learned", "lore", "mystery",
6      "news", "religion", "reviews", "romance", "science_fiction"]
7
8 >>> brown.words(categories="news")
9 ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ... ]
```

Lexical Resources

- ▶ A lexicon, or lexical resource, is a collection of words and/or phrases along with associated information (part-of-speech, sense definitions)
- ▶ Lexical resources are secondary to texts, usually created and enriched with the help of texts.

Lexical Resources Example

- ▶ `vocab = sorted(set(my_text))` – builds the vocabulary of `my_text`
- ▶ `word_freq = FreqDist(my_text)` – counts the frequency of each word in the text
- ▶ `con_freq = ConditionalFreqDist(list_of_tuples)` – calculates conditional frequencies

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	
cute	
Monday	
could	
will	

Condition: Romance

the	
cute	
Monday	
could	
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

```
1 >>> genre_word = [(genre, word)
2 ...   for genre in ["news", "romance"]
3 ...   for word in brown.words(categories=genre)]
4 >>> len(genre_word)
5 170576
6
7 >>> genre_word[:2]
8 [ ("news", "The"), ("news", "Fulton") ]
9 >>> genre_word[-2:]
10 [ ("romance", "afraid"), ("romance", "not") ]
```

Conditional Frequency Distributions

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

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


Conditional Frequency Distributions

```
1 >>> cfd["news"]
2 <FreqDist with 100554 outcomes>
3 >>> cfd["romance"]
4 <FreqDist with 70022 outcomes>
5 >>> list(cfd["romance"])
6 [",", ".", "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", ... ]
7 >>> cfd["romance"]["could"]
8 193
9 >>> cfd["news"]["could"]
10 93
```

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

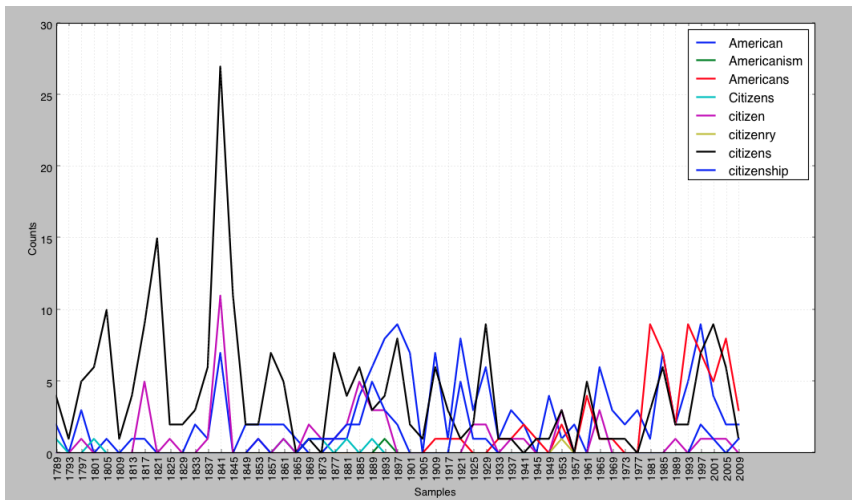
```
1 import nltk
2 from nltk.corpus import inaugural
3
4 cfd = nltk.ConditionalFreqDist((fileid[:4],w)
5     for fileid in inaugural.fileids()
6     for w in inaugural.words(fileid)
7     for target in ["american", "citizen"]
8     if w.lower().startswith(target))
9 print(cfd.conditions())
10 # ['American',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

Conditional Frequency Distributions

Visualize cfd with: `cfd.plot()`

```
1 from nltk import ConditionalFreqDist
2 help(ConditionalFreqDist)
3 >>>
4 ...
5 plot(self, *args, **kwargs)
6 |     Plot the given samples from the conditional frequency
7 |     distribution.
8 |
9 |     :param samples: The samples to plot
10 |     :type samples: list
11 |     :param title: The title for the graph
12 |     :type title: str
13 |     :param conditions: The conditions to plot (default is all)
14 |     :type conditions: list
```

Conditional Frequency Distributions



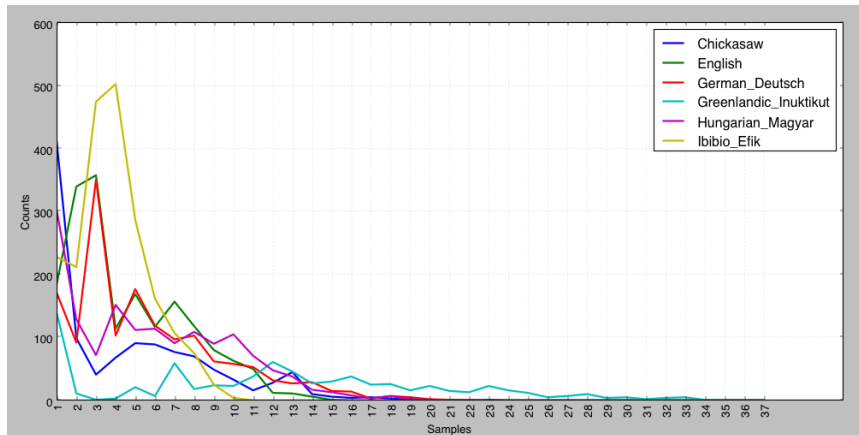
Conditional Frequency Distributions

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

```
1 from nltk.corpus import udhr
2 >>>udhr.fileids()
3 ['Abkhaz-Cyrillic+Abkh', 'Abkhaz-UTF8', 'Achehnese-Latin1', ... ]
4 >>>udhr.words("English-Latin1")
5 ['Universal', 'Declaration', 'of', 'Human', 'Rights', ... ]
6
7 languages = ["Chickasaw", "English", "German_Deutsch", "
              Greenlandic_Inuktitut", "Hungarian_Magyar", "Ibibio_Efik"]
8 cfd = nltk.ConditionalFreqDist((lang, len(word))
9 for lang in languages
10 for word in udhr.words(lang + "-Latin1"))
```

Conditional Frequency Distributions

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

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

```
1 >>> from nltk import bigrams
2 >>> list(bigrams(["more", "is", "said", "than", "done"]))
3
4 >>> [( 'more', 'is '), ( 'is ', 'said '), ( 'said ', 'than '), ( '
    than ', 'done' )]
```

```
5
6 >>> from nltk import trigrams
7 >>> list(trigrams(["more", "is", "said", "than", "done"]))
8
9 >>> [( 'more', 'is ', 'said '), ( 'is ', 'said ', 'than '), ( 'said
    ', 'than ', 'done' )]
```

Generating Random Text with Bigrams

```
1 >>> sent = ["In", "the", "beginning", "God", "created", "the", "heaven", "and", "the", "earth", "."]
2 >>> [y for y in nltk.bigrams(sent)]
3
4 [("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 result = cfd["living"].max()
```

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
```

Language Guesser Task

Implement a language guesser that takes a given text and outputs the language it thinks the text is written in

- ▶ `build_language_models()` should calculate a conditional frequency distribution where
 - ▶ the languages are the conditions
 - ▶ the values are frequency distribution of the lower case characters

```
1 languages = ['English', 'German_Deutsch', 'French_Francais']
2
3
4 # udhr corpus contains the Universal Declaration of Human Rights
5 in over 300 languages
6 language_base = dict((language, udhr.words(language + '-Latin1'))
7                       for language in languages)
8
9 # build the language models
10 langModeler = LangModeler(languages, language_base)
11 language_model_cfd = langModeler.build_language_models()
```

Language Guesser Task

Implement a language guesser that takes a given text and outputs the language it thinks the text is written in

```
1 languages = [ 'English', 'German_Deutsch', 'French_Francais' ]
2
3 # udhr corpus contains the Universal Declaration of Human Rights
4 in over 300 languages
5
6 language_base = dict((language, udhr.words(language + '-Latin1'))
7                       for language in languages)
8
9
10 # build the language models
11 langModeler = LangModeler(languages, language_base)
12 language_model_cfd = langModeler.build_language_models()
13
14 # print the models for visual inspection (you always should have a
15 look at the data)
16
17 for language in languages:
18     for letter in list(language_model_cfd[language].keys())[:10]:
19         print(language, letter, language_model_cfd[language].freq(letter))
```

Language Guesser Task

- ▶ `guess_language(language_model_cfd, text)`
returns the most likely language for a given text according to the algorithm that uses language models

```
1 text1 = "Peter had been to the office before they arrived."  
2 text2 = "Si tu finis tes devoirs, je te donnerai des bonbons."  
3 text3 = "Das ist ein schon recht langes deutsches Beispiel."  
4  
5 # guess the language by comparing the frequency distributions  
6 print("guess for english text is", guess_language(  
    language_model_cfd, text1))  
7 print("guess for french text is", guess_language(  
    language_model_cfd, text2))  
8 print("guess for german text is", guess_language(  
    language_model_cfd, text3))
```

Language Guesser Task

Implementation of

`guess_language(language_model_cfd, text):`

1. calculate the overall score of a given text based on the frequency of characters accessible by
`language_model_cfd[language].freq(character).`

```
1 for language in language_model_cfd.conditions():  
2     score = 0  
3     for character in text:  
4         score += language_model_cfd[language].freq(character)
```

2. return the most likely language with the maximum score

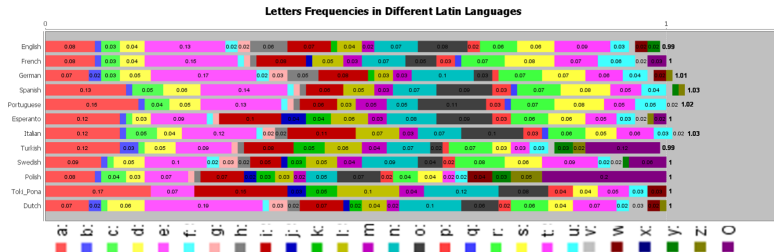
Language Guesser Task

Language models:

- ▶ the languages are the conditions
- ▶ the values: FreqDist of the lower case **characters** → **character level unigram** model
- ▶ the values: FreqDist of **bigrams of characters** → **character level bigram** model
- ▶ the values: FreqDist of **words** → **word level unigram** model
- ▶ the values: FreqDist of **bigrams of words** → **word level bigram** model

Language Guesser Task

- ▶ The distribution of characters in a languages of the same language family is usually not very different.
- ▶ Thus, it is difficult to differentiate between those languages using a unigram character model.



Exercise

What is calculated here?

```
1 import nltk
2 languages = [ 'eng', 'de', 'fr' ]
3 words = { 'eng': [ 'Universal', 'Declaration', 'of', 'Human' ],
4           'de': [ 'Die', 'Allgemeine', 'Erklärung', 'der', ... ],
5           'fr': [ 'Déclaration', 'universelle', 'des', ... ] }
6
7 cfd = nltk.ConditionalFreqDist(
8     (language, word)
9     for language in languages
10    for word in words[language])
11 >>> print(cfd.conditions())
12 >>> 1. ???
13 >>> print(cfd[ 'eng' ])
14 >>> 2. ???
15 >>> print(cfd[ 'eng' ].keys())
16 >>> 3. ???
17 >>> print(cfd[ 'eng' ][ 'Human' ])
18 >>> 4. ???
```

References

- ▶ <http://www.nltk.org/book/>
- ▶ <https://github.com/nltk/nltk>
- ▶ Christopher D. Manning, Hinrich Schütze 2000. Foundations of Statistical Natural Language Processing. *The MIT Press Cambridge, Massachusetts London, England.*
http://ics.upjs.sk/~pero/web/documents/pillar/Manning_Schuetze_StatisticalNLP.pdf