Text Corpora and Lexical Resources

Marina Sedinkina
- Folien von Desislava Zhekova -

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

December 19, 2017

Outline

1. Corpora
2. Accessing Text Corpora
3. Annotated Text Corpora
4. Lexical Resources
5. References

Corpora

- **Corpora** are large collections of linguistic data
- In fact, corpora are not always just random collections of data
- Many corpora are designed to contain a careful balance of material in one or more genres.

NLP and Corpora

Corpora are designed to achieve specific goal in NLP: data should provide best representation for the task. Such tasks are for example:

- word sense disambiguation
- coreference resolution
- machine translation
- part of speech tagging

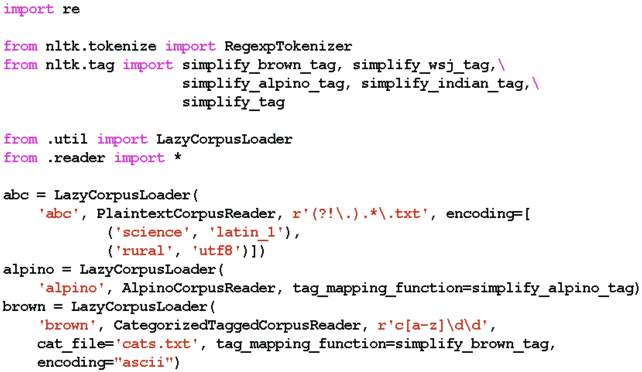
Corpora

- When the `nltk.corpus` module is imported, it automatically creates a set of corpus reader instances that can be used to access the corpora in the NLTK data distribution
- The corpus reader classes may be of several subtypes: `CategorizedTaggedCorpusReader`, `BracketParseCorpusReader`, `WordListCorpusReader`, `PlaintextCorpusReader` ...

```python
from nltk.corpus import brown

print(brown)

# prints
# <CategorizedTaggedCorpusReader in ... /corpora/brown (not loaded yet)>
```

Corpora

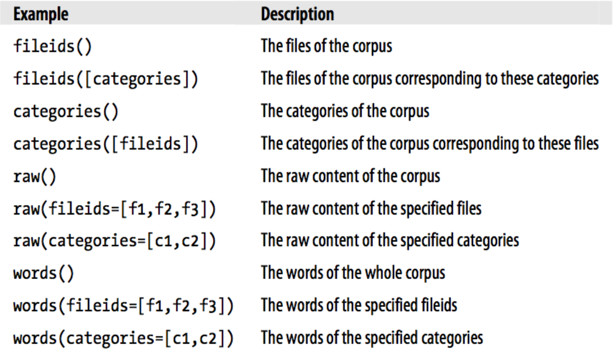
A look in the `nltk.corpus` module imports from its `__init__.py`



```
import re

from nltk.tokenize import RegexpTokenizer
from nltk.tag import simplify_brown_tag, simplify_wsj_tag,\
                    simplify_alpino_tag, simplify_indian_tag,\
                    simplify_tag

from .util import LazyCorpusLoader
from .reader import *

abc = LazyCorpusLoader(
    'abc', PlaintextCorpusReader, r'(?!\.).*\.txt', encoding=[
            ('science', 'latin_1'),
            ('rural', 'utf8')])
alpino = LazyCorpusLoader(
    'alpino', AlpinoCorpusReader, tag_mapping_function=simplify_alpino_tag)
brown = LazyCorpusLoader(
    'brown', CategorizedTaggedCorpusReader, r'c[a-z]\d\d',
    cat_file='cats.txt', tag_mapping_function=simplify_brown_tag,
    encoding="ascii")
```

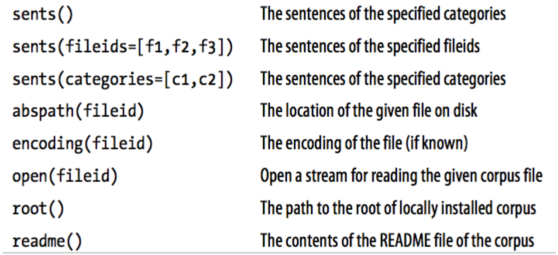
Corpus functions

Objects of type `CorpusReader` support the following functions:



Example	Description
fileids()	The files of the corpus
fileids([categories])	The files of the corpus corresponding to these categories
categories()	The categories of the corpus
categories([fileids])	The categories of the corpus corresponding to these files
raw()	The raw content of the corpus
raw(fileids=[f1,f2,f3])	The raw content of the specified files
raw(categories=[c1,c2])	The raw content of the specified categories
words()	The words of the whole corpus
words(fileids=[f1,f2,f3])	The words of the specified fileids
words(categories=[c1,c2])	The words of the specified categories

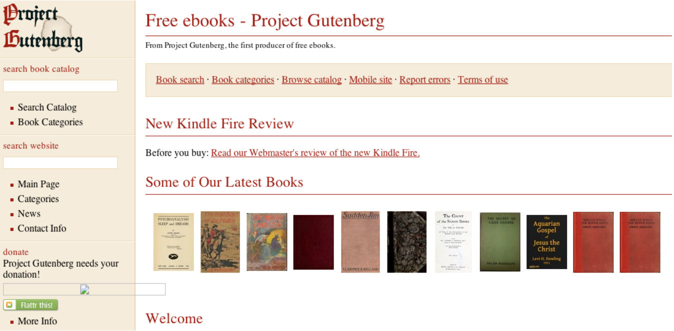
Corpus functions



`sents()`	The sentences of the specified categories
`sents(fileids=[f1,f2,f3])`	The sentences of the specified fileids
`sents(categories=[c1,c2])`	The sentences of the specified categories
`abspath(fileid)`	The location of the given file on disk
`encoding(fileid)`	The encoding of the file (if known)
`open(fileid)`	Open a stream for reading the given corpus file
`root()`	The path to the root of locally installed corpus
`readme()`	The contents of the README file of the corpus

Gutenberg Corpus

NLTK includes a small selection of texts from the Project Gutenberg electronic text archive, which contains more than 50 000 free electronic books, hosted at `http://www.gutenberg.org`.



Project Gutenberg

search book catalog

- Search Catalog
- Book Categories

search website

- Main Page
- Categories
- News
- Contact Info

donate
Project Gutenberg needs your donation!

Flattr this!

- More Info

Free ebooks - Project Gutenberg

From Project Gutenberg, the first producer of free ebooks.

Book search · Book categories · Browse catalog · Mobile site · Report errors · Terms of use

New Kindle Fire Review

Before you buy: Read our Webmaster's review of the new Kindle Fire.

Some of Our Latest Books

Welcome

Gutenberg Corpus

```
>>> import nltk
>>> nltk.corpus.gutenberg.fileids()
["austen-emma.txt", "austen-persuasion.txt", "austen-sense.
    txt", "bible-kjv.txt", "blake-poems.txt", "bryant-
    stories.txt", "burgess-busterbrown.txt", "carroll-
    alice.txt", "chesterton-ball.txt", "chesterton-brown.
    txt", "chesterton-thursday.txt", "edgeworth-parents.
    txt", "melville-moby_dick.txt", "milton-paradise.txt",
     "shakespeare-caesar.txt", "shakespeare-hamlet.txt", "
    shakespeare-macbeth.txt", "whitman-leaves.txt"]
```

Gutenberg Corpus

Naturally, each of the files into a corpus you can turn to a `nltk.Text` object and apply the functions this class provides:

```python
import nltk
from nltk.corpus import gutenberg

emma = nltk.Text(gutenberg.words("austen-emma.txt"))
print(emma.concordance("surprize", 40, 10))

# prints
# Building index ...
# Displaying 10 of 37 matches:
# etimes taken by surprize at his being st
# y good ." " You surprize me ! Emma must
# looked red with surprize and displeasure
# nd to his great surprize , that Mr . Elt
# rs . Weston   s surprize , and felt that
# ken up with the surprize of so sudden a
```

Gutenberg Corpus

It is often handy to know what all these `nltk` functions give us back, namely their return types:

```
words(): list of str
sents(): list of (list of str)
paras(): list of (list of (list of str))
tagged_words(): list of (str,str) tuple
tagged_sents(): list of (list of (str,str))
tagged_paras(): list of (list of (list of (str,str)))
chunked_sents(): list of (Tree with (str,str) leaves)
parsed_sents(): list of (Tree with str leaves)
parsed_paras(): list of (list of (Tree with str leaves))
xml(): A single xml ElementTree
raw(): unprocessed corpus contents
```

More documentation can be found using `help(nltk.corpus.reader)` and by reading the online Corpus HOWTO at `http://nltk.org/howto`.

Gutenberg Corpus

Extract statistics about the corpus:

```
from nltk.corpus import gutenberg

for fileid in gutenberg.fileids():
    num_chars = len(gutenberg.raw(fileid))
    num_words = len(gutenberg.words(fileid))
    num_sents = len(gutenberg.sents(fileid))
    num_vocab = len(set([w.lower() for w in gutenberg.words
        (fileid)]))
    print(int(num_chars/num_words), int(num_words/num_sents
        ), int(num_words/num_vocab), fileid)
```

Gutenberg Corpus

```python
from nltk.corpus import gutenberg

for fileid in gutenberg.fileids():
    num_chars = len(gutenberg.raw(fileid))
    num_words = len(gutenberg.words(fileid))
    num_sents = len(gutenberg.sents(fileid))
    num_vocab = len(set([w.lower() for w in gutenberg.words(fileid)]))
    print(int(num_chars/num_words), int(num_words/num_sents), int(num_words/num_vocab), fileid)
```

Statistics:

- `num_chars/num_words` – average word length
- `num_words/num_sents` – average sentence length
- `num_words/num_vocab` – number of times each vocabulary item appears in the text on average (our lexical diversity score)

Gutenberg Corpus

```
4 21 26 austen-emma.txt
4 23 16 austen-persuasion.txt
4 24 22 austen-sense.txt
4 33 79 bible-kjv.txt
4 18 5  blake-poems.txt
4 17 14 bryant-stories.txt
4 17 12 burgess-busterbrown.txt
4 16 12 carroll-alice.txt
4 17 11 chesterton-ball.txt
4 19 11 chesterton-brown.txt
4 16 10 chesterton-thursday.txt
```

- The value of 4 shows that the average word length appears to be a general property of English.
- Average sentence length and lexical diversity appear to be characteristics of particular authors.

Other Corpora

- Gutenberg contains established literature texts
- Other, less formal types of texts are also available e.g. **nltk.corpus.webtext**:
  - Discussions from a Firefox forum
  - Conversations overheard in New York
  - Movie script, advertisement, reviews

Web and Chat Text

```python
from nltk.corpus import webtext

for fileid in webtext.fileids():
    print(fileid, webtext.raw(fileid)[:30])

# prints
# firefox.txt Cookie Manager: "Don t allow s
# grail.txt SCENE 1: [wind] [clop clop clo
# overheard.txt White guy: So, do you have any
# pirates.txt PIRATES OF THE CARRIBEAN: DEAD
# singles.txt 25 SEXY MALE, seeks attrac old
# wine.txt Lovely delicate, fragrant Rhon
```

Web and Chat Text

Different corpora contain different linguistic information:

- What are the special characteristics of informal texts?
  - Different terminology (e.g. slang terms)
  - Different grammar (less strict)
- The choice of corpus thus always depends on what we want to find out!

Web and Chat Text

The chat corpus for example has the following characteristics:

1. collected for research on detection of Internet predators
2. contains over 10,000 posts
3. organized into 15 files
4. each file contains several hundred posts collected on a given date
5. each file also represents an age-specific chatroom (teens, 20s, 30s, 40s, plus a generic adults chatroom)
6. the filename contains the date, chatroom, and number of posts

???

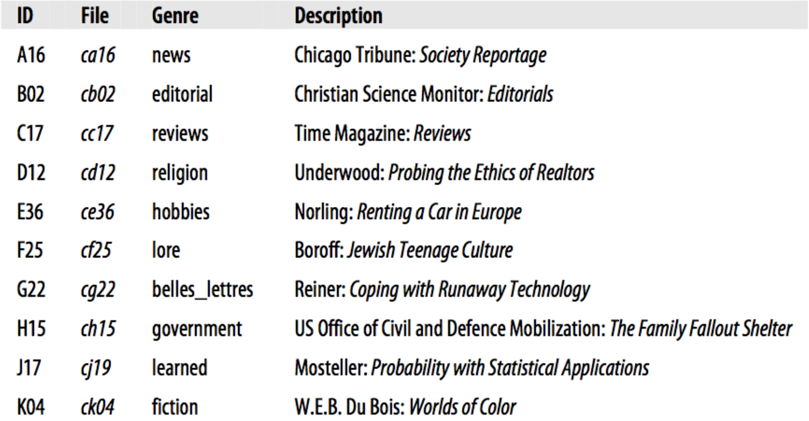
What other research questions could Web and Chat corpora answer?

```
from nltk.corpus import nps_chat

print(nps_chat)
# <NPSChatCorpusReader in ... /corpora/nps_chat (not loaded yet)>

chatroom = nps_chat.posts("10–19–20s_706posts.xml")
# same as using
# chatroom = nps_chat.posts(nps_chat.fileids()[0])

print(chatroom[123])

# prints
# ["i", "do", "n"t", "want", "hot", "pics", "of", "a", "female",
      ",", "I", "can", "look", "in", "a", "mirror", "."]
```

Brown Corpus

- 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**.

Brown Corpus



ID	File	Genre	Description
A16	*ca16*	news	Chicago Tribune: *Society Reportage*
B02	*cb02*	editorial	Christian Science Monitor: *Editorials*
C17	*cc17*	reviews	Time Magazine: *Reviews*
D12	*cd12*	religion	Underwood: *Probing the Ethics of Realtors*
E36	*ce36*	hobbies	Norling: *Renting a Car in Europe*
F25	*cf25*	lore	Boroff: *Jewish Teenage Culture*
G22	*cg22*	belles_lettres	Reiner: *Coping with Runaway Technology*
H15	*ch15*	government	US Office of Civil and Defence Mobilization: *The Family Fallout Shelter*
J17	*cj19*	learned	Mosteller: *Probability with Statistical Applications*
K04	*ck04*	fiction	W.E.B. Du Bois: *Worlds of Color*

Brown Corpus

```python
from nltk.corpus import brown

print(brown.categories())
# ["adventure", "belles_lettres", "editorial", "fiction", "government", "hobbies", "humor", "learned", "lore", "mystery", "news", "religion", "reviews", "romance", "science_fiction"]
```

Brown Corpus

```python
from nltk.corpus import brown

print(brown.categories())
print(brown.words(categories="news"))
# ["The", "Fulton", "County", "Grand", "Jury", "said", ... ]
```

Access the list of words, but restrict them to a specific category.

Brown Corpus

```python
from nltk.corpus import brown

print(brown.categories())
print(brown.words(categories="news"))

print(brown.words(fileids=["cg22"]))
# ["Does", "our", "society", "have", "a", "runaway", ",", ... ]
```

Access the list of words, but restrict them to a specific file.

Brown Corpus

```python
from nltk.corpus import brown

print(brown.categories())
print(brown.words(categories="news"))
print(brown.words(fileids=["cg22"]))

print(brown.sents(categories=["news", "editorial", "reviews"]))
# [["The", "Fulton", "County" ... ], ["The", "jury", "further" ... ], ... ]
```

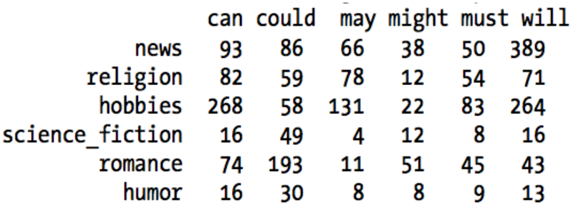
Access the list of sentences, but restrict them to a given list of categories.

Brown Corpus

We can compare genres in their usage of modal verbs:

```python
import nltk
from nltk.corpus import brown

news_text = brown.words(categories="news")
fdist = nltk.FreqDist([w.lower() for w in news_text])
modals = ["can", "could", "may", "might", "must", "will"]

for m in modals:
    print(m + ":", fdist[m])

# can: 94
# could: 87
# may: 93
# might: 38
# must: 53
# will: 389
```

Brown Corpus



```
                 can could  may might must will
           news   93    86   66    38   50  389
       religion   82    59   78    12   54   71
        hobbies  268    58  131    22   83  264
science_fiction   16    49    4    12    8   16
        romance   74   193   11    51   45   43
          humor   16    30    8     8    9   13
```

Observe that the most frequent modal in the news genre is **will**, while the most frequent modal in the romance genre is **could**.

Reuters Corpus

- contains 10,788 news documents
- totaling 1.3 million word
- documents have been classified into 90 topics, grouped into two sets, called “training" and “test"
- the text with file ID `test/14826` is a document drawn from the test set
- designed to detect the topic of a document

Reuters Corpus

```
>>> from nltk.corpus import reuters
>>> reuters.fileids()
["test/14826", "test/14828", "test/14829", "test/14832", ... ]
>>> reuters.categories()
["acq", "alum", "barley", "bop", "carcass", "castor-oil", "cocoa",
    "coconut", "coconut-oil", "coffee", "copper", "copra-cake",
   "corn", "cotton", "cotton-oil", "cpi", "cpu", "crude", "dfl",
    "dlr", ... ]
```

Reuters Corpus

- categories in the Reuters Corpus overlap with each other: news story often covers multiple topic
- topics can be covered by one or more document
- documents can be included in one or more categories

```
>>> reuters.categories("training/9865")
["barley", "corn", "grain", "wheat"]
>>> reuters.categories(["training/9865", "training/9880"])
["barley", "corn", "grain", "money-fx", "wheat"]
>>> reuters.fileids("barley")
["test/15618", "test/15649", "test/15676", "test/15728", "test/15871", ... ]
>>> reuters.fileids(["barley", "corn"])
["test/14832", "test/14858", "test/15033", "test/15043", "test/15106", "test/15287", "test/15341", "test/15618", "test/15618", "test/15648", ... ]
```

Inaugural Address Corpus

Time dimension property:

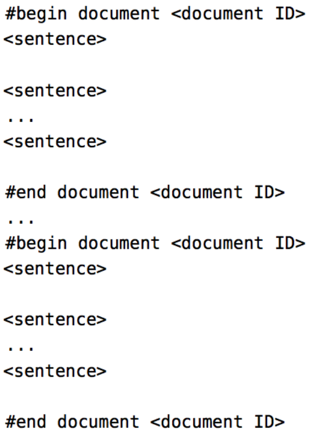
```
>>> from nltk.corpus import inaugural
>>> inaugural.fileids()
["1789–Washington.txt", "1793–Washington.txt", "1797–Adams.txt",
     ... ]
>>> [fileid[:4] for fileid in inaugural.fileids()]
["1789", "1793", "1797", "1801", "1805", "1809", "1813", "1817", "
     1821", ... ]
```

Annotated Text Corpora

Many text corpora contain linguistic annotations:

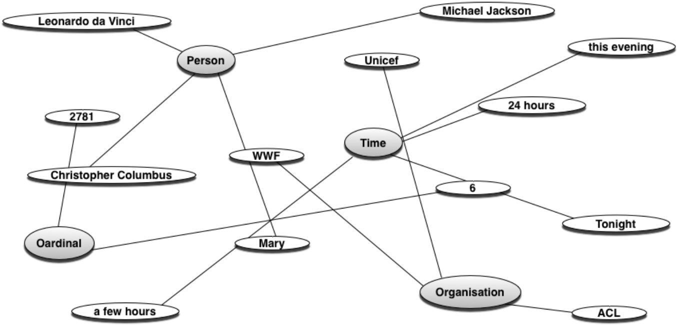
- part-of-speech tags
- named entities
- syntactic structures
- semantic roles

Annotated Text Corpora

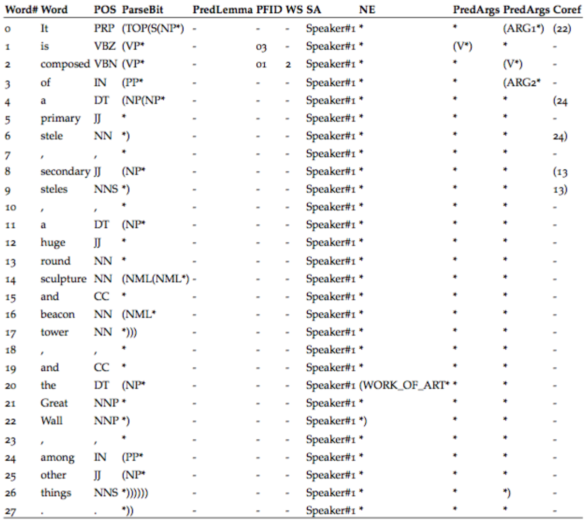


```
#begin document <document ID>
<sentence>

<sentence>
...
<sentence>

#end document <document ID>
...
#begin document <document ID>
<sentence>

<sentence>
...
<sentence>

#end document <document ID>
```

Annotated Text Corpora



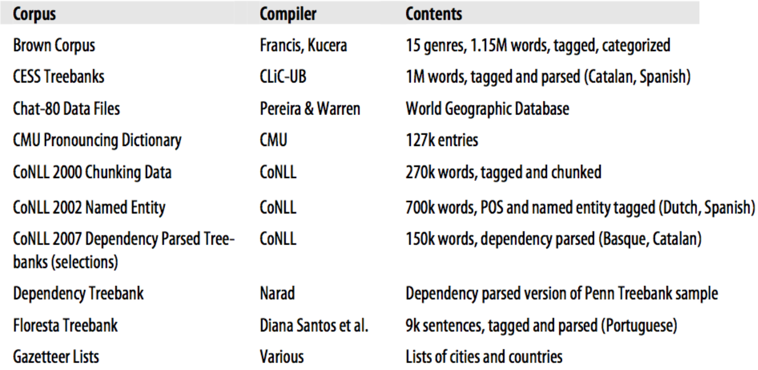
Annotated Text Corpora



Word#	Word	POS	ParseBit	PredLemma	PFID	WS	SA	NE	PredArgs	PredArgs	Coref
0	It	PRP	(TOP(S(NP*)	-	-	-	Speaker#1	*	*	(ARG1*)	(22)
1	is	VBZ	(VP*	-	03	-	Speaker#1	*	(V*)	*	-
2	composed	VBN	(VP*	-	01	2	Speaker#1	*	*	(V*)	-
3	of	IN	(PP*	-	-	-	Speaker#1	*	*	(ARG2*	-
4	a	DT	(NP(NP*	-	-	-	Speaker#1	*	*	*	(24
5	primary	JJ	*	-	-	-	Speaker#1	*	*	*	-
6	stele	NN	*)	-	-	-	Speaker#1	*	*	*	24)
7	,	,	*	-	-	-	Speaker#1	*	*	*	-
8	secondary	JJ	(NP*	-	-	-	Speaker#1	*	*	*	(13
9	steles	NNS	*)	-	-	-	Speaker#1	*	*	*	13)
10	,	,	*	-	-	-	Speaker#1	*	*	*	-
11	a	DT	(NP*	-	-	-	Speaker#1	*	*	*	-
12	huge	JJ	*	-	-	-	Speaker#1	*	*	*	-
13	round	NN	*	-	-	-	Speaker#1	*	*	*	-
14	sculpture	NN	(NML(NML*)	-	-	-	Speaker#1	*	*	*	-
15	and	CC	*	-	-	-	Speaker#1	*	*	*	-
16	beacon	NN	(NML*	-	-	-	Speaker#1	*	*	*	-
17	tower	NN	*)))	-	-	-	Speaker#1	*	*	*	-
18	,	,	*	-	-	-	Speaker#1	*	*	*	-
19	and	CC	*	-	-	-	Speaker#1	*	*	*	-
20	the	DT	(NP*	-	-	-	Speaker#1	(WORK_OF_ART*	*	*	-
21	Great	NNP	*	-	-	-	Speaker#1	*	*	*	-
22	Wall	NNP	*)	-	-	-	Speaker#1	*)	*	*	-
23	,	,	*	-	-	-	Speaker#1	*	*	*	-
24	among	IN	(PP*	-	-	-	Speaker#1	*	*	*	-
25	other	JJ	(NP*	-	-	-	Speaker#1	*	*	*	-
26	things	NNS	*))))))	-	-	-	Speaker#1	*	*	*)	-
27	.	.	*))	-	-	-	Speaker#1	*	*	*	-

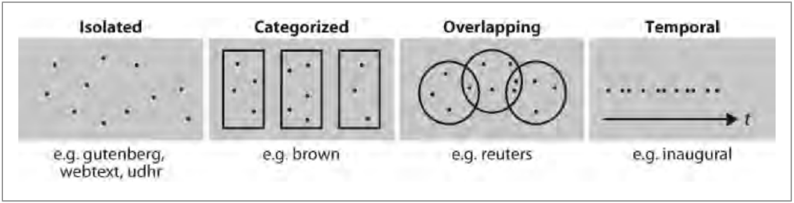
Annotated Text Corpora

download required corpus via nltk.download()



Corpus	Compiler	Contents
Brown Corpus	Francis, Kucera	15 genres, 1.15M words, tagged, categorized
CESS Treebanks	CLiC-UB	1M words, tagged and parsed (Catalan, Spanish)
Chat-80 Data Files	Pereira & Warren	World Geographic Database
CMU Pronouncing Dictionary	CMU	127k entries
CoNLL 2000 Chunking Data	CoNLL	270k words, tagged and chunked
CoNLL 2002 Named Entity	CoNLL	700k words, POS and named entity tagged (Dutch, Spanish)
CoNLL 2007 Dependency Parsed Treebanks (selections)	CoNLL	150k words, dependency parsed (Basque, Catalan)
Dependency Treebank	Narad	Dependency parsed version of Penn Treebank sample
Floresta Treebank	Diana Santos et al.	9k sentences, tagged and parsed (Portuguese)
Gazetteer Lists	Various	Lists of cities and countries

Corpora Structure



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

So far, we have worked with the following:

- `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

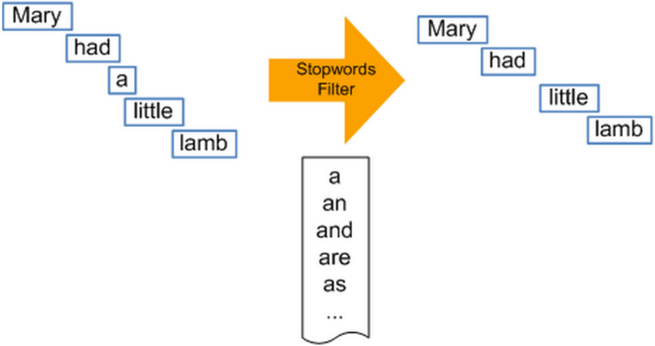
Lexical Resources: Wordlists

Word lists are another type of lexical resources. NLTK includes some examples:

- `nltk.corpus.stopwords`
- `nltk.corpus.names`
- `nltk.corpus.swadesh`
- `nltk.corpus.words`

Stopwords

Stopwords are high-frequency words with little lexical content such as **the**, **to**,**and**.



Wordlists: Stopwords

```
>>> from nltk.corpus import stopwords
>>> stopwords.words("english")
["a", "a s", "able", "about", "above", "according", "
    accordingly", "across", "actually", "after", "
    afterwards", "again", "against", "ain t", "all", "
    allow", "allows", "almost", "alone", "along", "already
    ", "also", "although", "always", ... ]
```

Also available for: Danish, Dutch, English, Finnish, French, German, Hungarian, Italian, Norwegian, Portuguese, Russian, Spanish, Swedish and Turkish

Wordlist Corpora

```
def fraction(text):
... stopwords = nltk.corpus.stopwords.words("english")
... content = [w for w in text if w.lower() not in stopwords]
... return len(content) / len(text)
>>> fraction(nltk.corpus.reuters.words())
```

???

What is calculated here?

Wordlist Corpora

```
def fraction(text):
... stopwords = nltk.corpus.stopwords.words("english")
... content = [w for w in text if w.lower() not in stopwords]
... return len(content) / len(text)
>>> fraction(nltk.corpus.reuters.words())
# prints 0.65997695393285261
```

Wordlists: Names

- Names Corpus is a wordlist corpus, containing 8,000 first names categorized by gender.
- The male and female names are stored in separate files.

Wordlists

```python
import nltk

names = nltk.corpus.names
print(names.fileids())
# ["female.txt", "male.txt"]

female_names = names.words(names.fileids()[0])
male_names = names.words(names.fileids()[1])

print([w for w in male_names if w in female_names])
#["Abbey", "Abbie", "Abby", "Addie", "Adrian", "Adrien", "Ajay", "Alex", "Alexis", "Alfie", "Ali", "Alix", "Allie", "Allyn", "Andie", "Andrea", "Andy", "Angel", "Angie", "Ariel", "Ashley", "Aubrey", "Augustine", "Austin", "Averil", ... ]
```

Wordlists

NLP application for which gender information would be helpful

Anaphora Resolution:
Adrian drank from the cup. **He** liked the tea.

Note

Both **he** as well as **she** will be possible solutions when Adrian is the antecedent, since this name occurs in both lists: female and male names.

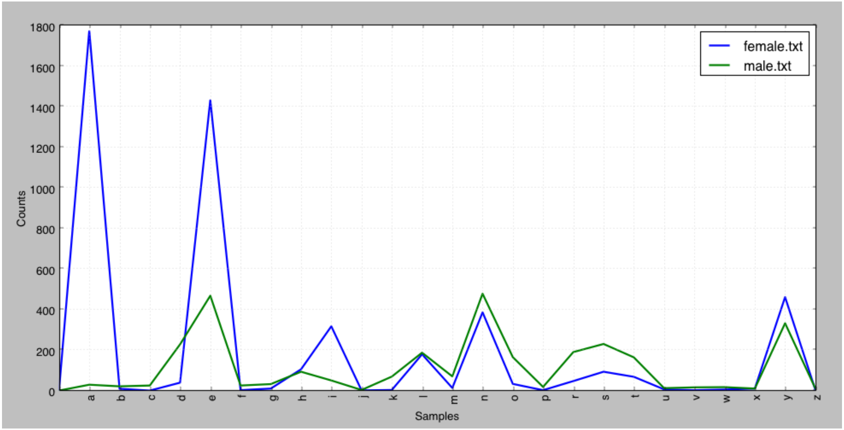
Wordlists

```python
import nltk
names = nltk.corpus.names

cfd = nltk.ConditionalFreqDist(
          (fileid, name[-1])
          for fileid in names.fileids()
          for name in names.words(fileid))
```

???

What will be calculated for the conditional frequency distribution stored in `cfd`?

Wordlists


1800
1600
1400
1200
1000
800
600
400
200
0
Counts
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
r
s
t
u
v
w
x
y
z
Samples
female.txt
male.txt

Wordlists: Swadesh

- comparative wordlist
- lists about 200 common words in several languages.

Comparative Wordlists

```
>>> from nltk.corpus import swadesh
>>> swadesh.fileids()
["be", "bg", "bs", "ca", "cs", "cu", "de", "en", "es", "fr"
    , "hr", "it", "la", "mk", "nl", "pl", "pt", "ro", "ru"
    , "sk", "sl", "sr", "sw", "uk"]
>>> swadesh.words("en")
["I", "you (singular), thou", "he", "we", "you (plural)", "
    they", "this", "that", "here", "there", "who", "what",
     "where", "when", "how", "not", "all", "many", "some",
     "few", "other", "one", "two", "three", "four", "five"
    , "big", "long", "wide", ... ]
```

Comparative Wordlists

```
>>> fr2en = swadesh.entries(["fr", "en"])
>>> fr2en
[("je", "I"), ("tu, vous", "you (singular), thou"), ("il",
    "he"), ... ]
>>> translate = dict(fr2en)
>>> translate["chien"]
"dog"
>>> translate["jeter"]
"throw"
```

Comparative Wordlists

```
>>> de2en = swadesh.entries(["de", "en"]) # German-English
>>> es2en = swadesh.entries(["es", "en"]) # Spanish-English
>>> translate.update(dict(de2en))
>>> translate.update(dict(es2en))
>>> translate["Hund"] "dog"
>>> translate["perro"] "dog"
```

Comparative Wordlists

```
>>> languages = ["en", "de", "nl", "es", "fr", "pt", "la"]
>>> for i in [139, 140, 141, 142]:
...     print swadesh.entries(languages)[i]
...
 ("say", "sagen", "zeggen", "decir", "dire", "dizer", "dicere")
 ("sing", "singen", "zingen", "cantar", "chanter", "cantar", "canere")
 ("play", "spielen", "spelen", "jugar", "jouer", "jogar, brincar", "ludere")
"float", "schweben", "zweven", "flotar", "flotter", "flutuar, boiar", "fluctuare")
```

Words Corpus

- NLTK includes some corpora that are nothing more than wordlists.
- We can use it to find unusual or misspelt words in a text.
- The Words Corpus `/usr/share/dict/words` from Unix is used by some spell checkers.

```

def unusual_words(text):
  text_vocab=set(w.lower() for w in text if w.isalpha())
  english_vocab=set(w.lower() for w in nltk.corpus.words.words())
  unusual=text_vocab - english_vocab
  return sorted(unusual)

>>> unusual_words(nltk.corpus.gutenberg.words( austen-sense.txt ))
[ abbeyland , abhorred , abilities , abounded , ... ]
```

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 frequencies of the lower case characters

```

languages = [ English , German_Deutsch , French_Francais ]

# udhr corpus contains the Universal Declaration of Human Rights
    in over 300 languages
language_base = dict((language, udhr.words(language + -Latin1 ))
    for language in languages)

# build the language models
langModeler = LangModeler(languages, language_base)
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

```

languages = [ English , German_Deutsch , French_Francais ]

# udhr corpus contains the Universal Declaration of Human Rights in over 300 languages
language_base = dict((language, udhr.words(language + -Latin1 )) for language in languages)

# build the language models
langModeler = LangModeler(languages, language_base)
language_model_cfd = langModeler.build_language_models()


# print the models for visual inspection (you always should have a look at the data)
for language in languages:
 for letter in list(language_model_cfd[language].keys())[:10]:
  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

```
text1 = "Peter had been to the office before they arrived."
text2 = "Si tu finis tes devoirs, je te donnerai des bonbons."
text3 = "Das ist ein schon recht langes deutsches Beispiel."

# guess the language by comparing the frequency distributions
print("guess for english text is", guess_language(
     language_model_cfd,text1))
print("guess for french text is", guess_language(
     language_model_cfd,text2))
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)`.

```python
for language in language_model_cfd.conditions():
    score = 0
    for character in text:
        score += language_model_cfd[language].freq(character)
```

2. return the most likely language with the maximum score

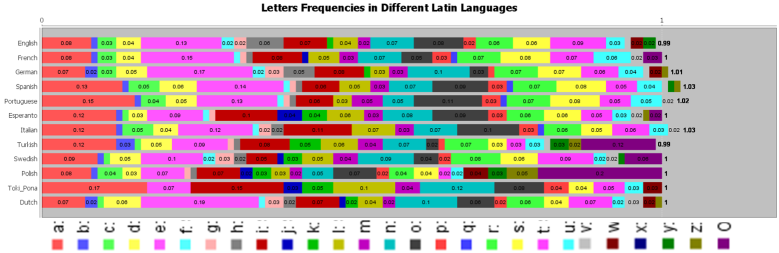
Language Guesser Task

Language models:

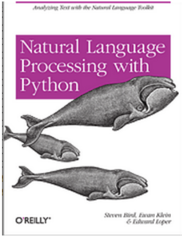
- the languages are the conditions
- the values: FreqDist of the lower case **characters** → **character level unigram** model
- the values: FreqDist of **bigrams of characters** → **character level bigram** model
- the values: FreqDist of **words** → **word level unigram** model
- the values: FreqDist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



References

- 
Analyzing Text with the Natural Language Toolkit
Natural Language Processing with Python
O'REILLY
Steven Bird, Ewan Klein & Edward Loper
  `http://www.nltk.org/book/`
- `https://github.com/nltk/nltk`