The Natural Language Toolkit (NLTK)

L435/L555

Dept. of Linguistics, Indiana University

Fall 2016

NLTK

The Natural Language Toolkit (NLTK) is:

> *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.*

http://www.nltk.org/ (retrieved 8/9/16)

Today, we'll look at some basic functionality for working with text files (taken from ch. 1 & 3 of the NLTK book)

- http://www.nltk.org/book/

Getting Started

Download the materials from the NLTK book (if you have not done so already):

```
>>> import nltk
>>> nltk.download()
```

Assuming that book material have been downloaded, for today do:

```
>>> import nltk
>>> from nltk.book import *
```

This last command loads various texts to work with

Searching Text

We now have texts available:

```
>>> text1
<Text: Moby Dick by Herman Melville 1851>
```

Methods:

- concordance
  `text1.concordance("monstrous")`
- similar
  `text1.similar("monstrous")`
  `text2.similar("monstrous")`
- common_contexts
  `text2.common_contexts(["monstrous", "very"])`

Texts as Lists of Words

NLTK treats texts as lists of words

Here are the first 20 words of *Moby Dick*:

```
>>> text1[:20]
['[', 'Moby', 'Dick', 'by', 'Herman', 'Melville',
'1851', ']', 'ETYMOLOGY', '.', '(', 'Supplied',
'by', 'a', 'Late', 'Consumptive', 'Usher', 'to',
'a', 'Grammar']
```

Counting Vocabulary

Because it's Python-based, it's easy to create functions to analyze the texts

```
>>> def lexical_diversity(text):
...     return len(text) / len(set(text))
...
>>> lexical_diversity(text1)
13.502044830977896
>>> lexical_diversity(text2)
20.719449729255086
```

Note: `set()` converts a list to a set

- More on sets and functions later this semester ...

Simple Statistics

Frequency Distributions

NLTK has pre-built packages for creating distributions

```
>>> fdist1 = FreqDist(text1)
>>> fdist1
<FreqDist with 19317 samples and 260819 outcomes>
>>> fdist1['whale']
906
```

We will build our own dictionaries, but some capabilities are quickly calculated with `FreqDist()`:

```
>>> fdist1.most_common(50)
[(',', 18713), ('the', 13721), ('.', 6862), ('of', 6536),
 ('and', 6024), ('a', 4569), ('to', 4542), (';', 4072), ...
```

Simple Statistics

Frequency Distributions

More FreqDist fun ...

```
>>> vocabulary1 = fdist1.keys()
>>> list(vocabulary1)[:5]
['wrestlings', 'symbolically', 'sternly',
 'disparagement', 'drama']

>>> [w for w in set(text1) if len(w) > 10 and fdist1[w] > 20]
['circumstances', 'Nevertheless', 'Nantucketer',
 'considerable', 'considering', 'circumstance',
 'harpooneers', 'nevertheless']
```

Organizing by word length

```
>>> fdist = FreqDist([len(w) for w in text1])
>>> fdist
<FreqDist with 19 samples and 260819 outcomes>
>>> list(fdist.keys())
[3, 1, 4, 2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, ... ]
>>> list(fdist.items())
[(3, 50223), (1, 47933), (4, 42345), (2, 38513), ... ]
>>> fdist.max()
3
>>> fdist[3]
50223
>>> fdist.freq(3)
0.19255882431878046
```

An NLTK FreqDist from scratch

```
import nltk
example = ['a','rose','is','a','rose','is','a','rose']

# note that no arguments are given to FreqDist
# -> this creates an empty frequency distribution
fdist = nltk.FreqDist()

# iterate over words in example sentence
for word in example:
    # add 1 to the count of each word
    # note: if never defined before, it starts at 0
    fdist[word] += 1

# see the contents of fdist
print(fdist.items())
```

Notes on previous slide

- To create an empty frequency distribution, use `FreqDist()` with no arguments
- To add to item frequencies, use the += notation
  - Access each word's frequency using square brackets
  - We'll see this type of notation with dictionaries
- If you have only done `import nltk` (and not `from nltk.book import *`), you need to use `nltk.FreqDist()` (and not just `FreqDist()`)

This type of procedure is useful when creating distributions as you iterate over new texts

Functions

- bigrams

```
>>> list(bigrams(text1[:10]))
[('[', 'Moby'), ('Moby', 'Dick'), ('Dick', 'by'),
('by', 'Herman'), ('Herman', 'Melville'),
('Melville', '1851'), ('1851', ']'), (']', 'ETYMOLOGY'),
('ETYMOLOGY', '.')]
```

- collocations

```
>>> text1.collocations()
Building collocations list
Sperm Whale; Moby Dick; White Whale; old man; Captain Ahab;
sperm whale; Right Whale; Captain Peleg; New Bedford;
Cape Horn; cried Ahab; years ago; lower jaw; never mind;
Father Mapple; cried Stubb; chief mate; white whale;
ivory leg; one hand
```

Exercises (1)

#7. Find the collocations in `text5`.
#18. Using list addition, and the `set` and `sorted` operations, compute the vocabulary of the sentences `sent1` ... `sent8`.
#19. What is the difference between the following two lines? Which one will give a larger value? Will this be the case for other texts?

```
>>> sorted(set([w.lower() for w in text1]))
>>> sorted([w.lower() for w in set(text1)])
```

#22. Find all the four-letter words in the Chat Corpus (`text5`). With the help of a frequency distribution (`FreqDist`), show these words in decreasing order of frequency.

Using your own data

Using `.read()`, you can read a text file as a string in Python

- With this string representation, you can use NLTK's utilities
- See chapter 3 of the NLTK book for downloading URL content, as well

Assume `raw` is *Crime and Punishment*, from Project Gutenberg (e.g., `raw = open("pg2554.txt").read()`)

```
>>> tokens = nltk.word_tokenize(raw)
>>> tokens[:10]
['The', 'Project', 'Gutenberg', 'EBook', 'of',
 'Crime', 'and', 'Punishment', ',', 'by']
```

word_tokenize

A different way to use `word_tokenize` is to use the `from` statement in importing:

```
>>> from nltk import word_tokenize
```

Then, instead of typing:

```
>>> tokens = nltk.word_tokenize(raw)
```

you could use:

```
>>> tokens = word_tokenize(raw)
```

Creating an NLTK text

```
>>> text = nltk.Text(tokens)
>>> type(text)
<class 'nltk.text.Text'>
>>> text[:10]
['The', 'Project', 'Gutenberg', 'EBook', 'of',
'Crime', 'and', 'Punishment', ',', 'by']
>>> text.collocations()
Building collocations list
Katerina Ivanovna; Pyotr Petrovitch; Pulcheria Alexandrovna;
Avdotya Romanovna; Marfa Petrovna; Rodion Romanovitch;
Sofya Semyonovna; old woman; Project Gutenberg-tm;
Porfiry Petrovitch; Amalia Ivanovna; great deal; ...
```

Other ways to tokenize

NLTK allows for you to specify your own tokenization using regular expressions

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)    # set flag to allow verbose regexps
...     ([A-Z]\.)+        # abbreviations, e.g. U.S.A.
...   | \w+(-\w+)*        # words with optional internal hyph
...   | \$?\d+(\.\d+)?%?  # currency and percentages, e.g. $1
...   | \.\.\.            # ellipsis
...   | [][.,;"'?():-_`]  # these are separate tokens
... '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

More on regular expressions in a couple weeks ...

Stemming (prep)


```
>>> raw = """DENNIS: Listen, strange women lying in ponds
... distributing swords is no basis for a system of
... government.  Supreme executive power derives from
... a mandate from the masses, not from some farcical
... aquatic ceremony."""
>>> tokens = word_tokenize(raw)
```

Stemming

There are options for normalizing words, as well

```
>>> porter = nltk.PorterStemmer()
>>> lancaster = nltk.LancasterStemmer()
>>> [porter.stem(t) for t in tokens]
['DENNI', ':', 'Listen', ',', 'strang', 'women', 'lie', ...]
>>> [lancaster.stem(t) for t in tokens]
['den', ':', 'list', ',', 'strange', 'wom', 'lying', ...]
```

Lemmatization


```
>>> wn1 = nltk.WordNetLemmatizer()
>>> [wn1.lemmatize(t) for t in tokens]
['DENNIS', ':', 'Listen', ',', 'strange', 'woman',
 'lying', 'in', 'pond', 'distributing', 'sword', 'is',
 'no', 'basis', 'for', 'a', 'system', 'of',
 'government', '.', ...]
```

(It gets better if you pass in a part-of-speech)

Exercises (2)

#18. Read in some text from a corpus, tokenize it, and print the list of all *wh*-word types that occur. (*wh*-words in English are used in questions, relative clauses and exclamations: *who*, *which*, *what*, and so on.) Print them in order. Are any words duplicated in this list, because of the presence of case distinctions or punctuation?

#30. Use the Porter Stemmer to normalize some tokenized text, calling the stemmer on each word. Do the same thing with the Lancaster Stemmer and see if you observe any differences.

Built-In Corpora

NLTK has a variety of built-in corpora, which allow you to work with different kinds of (somewhat standard) data

- Gutenberg Corpus
- Web and Chat Text
- Brown Corpus
- Reuters Corpus
- Inaugural Address Corpus

There are functions to load in your own corpus

- Note: a corpus typically has sub-structure & meta-data, whereas a text is simply a text

(http://www.nltk.org/book/ch02.html)

Brown Corpus


```
>>> from nltk.corpus import brown
>>> brown.categories()
['adventure', 'belles_lettres', 'editorial', ...]
```

Note that these categories correspond to sections (e.g., section *f* (or *cf*) is “lore”)

- See section 1.3 of chapter 2 of the NLTK book

Brown Corpus

Data access

Access words (with `.words()`) in various ways:

```
>>> brown.words(categories='news')
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
>>> brown.words(fileids=['cg22'])
['Does', 'our', 'society', 'have', 'a', 'runaway', ',', ...]
>>> brown.words(fileids=['cm01','cm02'])
['Now', 'that', 'he', 'knew', 'himself', 'to', 'be', ...]
```

Access sentences in the same ways but with `.sents()`, e.g.:

```
>>> brown.sents(categories=['news', 'editorial', 'reviews'])
[['The', 'Fulton', 'County'...], ['The', 'jury', 'further'...
```