NLP Lab Session Week 2
Word Frequencies from text in NLTK
September 9, 2015

Starting an NLTK Session

Python can be run on the command line to run Python programs (that have been written in a text editor or an IDE (Integrated Development Environment)) or as an interpreter, where you just type little pieces of Python on the interpreter line and it runs them for you. We will mostly be running Python in interpreter mode in an IDLE window.

For Windows:
If the Python IDLE icon is on your desktop, just double-click that to open the Idle window.
Or to open the Python IDLE window, go to All Programs->Python 2.7 -> IDLE (Python GUI)

For Mac:
Start a terminal session. Type either "python" or "idle".

It's ok if it's Python 2.6, but don't try to use any version of Python 3 with these lab notes.

Now download the files
LabWeek2examples.txt and gutenbergwords.py
and save them in a folder where you will keep materials for this class.

You will probably want to work by having the IDLE window open for testing NLTK and a browser window open with these instructions. You may also want to have a separate tab or window open to the NLTK book: http://www.nltk.org/book_1ed/, where these examples are taken from Chapter 1.

In the following, examples for you to try are given following the Python Idle prompt of >>>. You can copy and paste the Python example into the python interpreter, or you can type the example in.

Loading Text to Process from the Gutenberg Corpus in NLTK

In this section, we'll reload the emma text from the first book, Jane Austen's "Emma", in the Gutenberg Corpus in the NLTK.

```
>>> import nltk
```

You can then view some books obtained from the Gutenberg on-line book project:

```
>>> nltk.corpus.gutenberg.fileids()

>>> file0 = nltk.corpus.gutenberg.fileids( ) [0]
>>> file0
```

We can get the original text, using the raw function:
>>> emmatext = nltk.corpus.gutenberg.raw(file0)
>>> len(emmatext)

Since this is quite long, we can view part of it, e.g. the first 120 **characters**
>>> emmatext[:120]

NLTK has several tokenizers available to break the raw text into tokens; we will use one trained on news articles that separates by white space and by special characters (punctuation), but also leaves together some of these such as Mr.:
>>> emmatokens = nltk.word_tokenize(emmatext)
>>> len(emmatokens)

View the first 200 **tokens** and observe some of the tokenization rules used, and some of the errors with not separating the '.' at the end of the sentence.
>>> emmatokens[:200]

We probably want to use the lowercase versions of the words. We define the variable emmawords using a Python list comprehension: "[w.lower() for w in emmatokens]", where this expression is the list consisting of the all the items that are the lowercase of w "w.lower()" for all w that are in emmatokens.

>>> emmawords = [w.lower( ) for w in emmatokens]
>>> emmawords[:50]
>>> len(emmawords)

We can further view the words by getting the unique words and sorting them:
>>> emmavocab = sorted(set(emmawords))
>>> emmavocab[:50]

We can see that we will probably want to get rid of these special characters – Regular Expressions to the Rescue! (as in xkcd _), but we'll work on that next week.

Using a Python Frequency Dictionary to Count Words (Learning Exercise)

We will start to create a frequency list using a Python dictionary, as a learning exercise about Python dictionaries. In the next section, we'll use the NLTK Frequency Dictionary module instead.

Python dictionaries are described in the NLTK book, in section 5.3. Using dictionaries to count frequencies of words is described in section 1.3.

We will create a dictionary that has (key, value) pairs where the key is the word and the value is the frequency, or the count of the number of occurrences of each word. We start with an empty dictionary.
>>> emmadict = {}

We will then write a for loop to go through the words and count them. To do that, if the word is already in the dictionary, we add 1 to its count and if not, we start the count at 1. Note that we indent for the "for" loop and then we indent some more for the body of the "if" and the "else" parts.

```
>>> for w in emmawords:
   if w in emmadict:
      emmadict[w] += 1
   else:
      emmadict[w] = 1
```

How many unique words were counted? We can get the number of keys in the dictionary, where the keys are given by calling the keys() function on the dictionary.

```
>>> len(emmadict.keys())
```

We can look up individual words and print the frequency:

```
>>> emmadict['the']
5198
>>> emmadict['of']
4262
>>> emmadict['a']
3124
```

We don't really want to print the frequencies of all 9,000 words, but we can print 30 of them, using the first 30 keys, and we note that the keys of a dictionary don't occur in any particular order:

```
>>> for word in emmadict.keys()[:30]:
   print word, emmadict[word]
```

Now what we really want is the list of word frequencies in ranked order, i.e. the most frequent first. We can do that in Python, but it's a little awkward, so NLTK has conveniently defined a class to do that for us.

NLTK Frequency Distributions to Count Words

NLTK has a set of functions that use a data structure called a Frequency Distribution, FreqDist.

This structure is an extension of the Python dictionary structures. We can import it from the nltk probability module.

```
>>> from nltk import FreqDist
```

This class allows you to make a Frequency Distribution just by initializing it with a list of words. It will do all the counting for you and create a distribution in which the set of keys are all the words, and the set of values are the frequency (count) of each word. The keys( ) function produces the list of words in order of decreasing frequency. (This is true in NLTK 2.0 versions, but not in 3.0 versions.)

```
>>> fdist = FreqDist(emmawords)
>>> fdist.keys( )[:50]
```

We can treat the frequency distribution just like a Python dictionary and we can look at the frequencies of individual words:

```
>>> fdist['emma']
>>> fdist['the']
```

Or we can use a for loop to look at the frequencies of the first words, and this time the keys are sorted in the order of decreasing values.

```
>>> for word in fdist.keys()[:30]:
   print word, fdist[word]
```

Making a Python Program

Suppose that we want to look at the frequency distribution of one of the other Gutenberg books. It would be nice if we could save the steps of getting the raw text from the book file, getting the words and putting them into a Frequency Distribution. First look at the books available from Gutenberg.

```
>>> 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']
```

In the IDLE window, we'll create a new module (or new Python shell) and type or copy in the steps from the file gutenbergwords.py, with a variable that we can change to get different files/books.

```
# program to get text from Gutenberg files
import nltk
from nltk import FreqDist

fileno = 1

nfile = nltk.corpus.gutenberg.fileids( ) [fileno]
```

```
ntext = nltk.corpus.gutenberg.raw(nfile)
print 'First 200 characters of text', ntext[200]
print

ntokens = nltk.word_tokenize(ntext)
print 'First 100 tokens', ntokens[:100]
print

nwords = [w.lower( ) for w in ntokens]

ndist = FreqDist(nwords)

print '30 Top Frequency Words:'
for word in ndist.keys()[:30]:
   print word, ndist[word]
```

After putting this into the new module, if we select Run Module, it will restart the python shell and run these python steps. Warning: when it restarts the python shell, it loses all the definitions that you did previously.

An alternative way to run the python program is to open a Command Prompt (Windows) or Terminal (Mac) window. Use the change directory command 'cd' to move to the directory of the gutenberg.py program and run the program on the command line.

```
$ gutenberg.py
[or on a Mac $ python gutenberg.py]
```

Try it out:

Pick a different file number for the Gutenberg books, change the variable fileno to this number and rerun the program to create a frequency distribution for the book.

Save the list of top words and their frequencies to put in a post for this week.

Exercise to submit this week:

Go to Blackboard and find the Discussion for the second week exercises. Create a post in which you:

- Make a post with the title giving the text that you used for a frequency distribution
- In the body of the post, give the top 30 frequency words from the text that you chose in the "try it out" section.