NLP Lab Session Week 2
January 23, 2011

Starting a Python and an NLTK Session

Open a Python 2.7 IDLE (Python GUI) window by going to All Programs->Python 2.7 -> IDLE (Python GUI).

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/, 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 Idle window, or you can type the example in.

Getting Started in Python and NLTK

First let's verify that the nltk-data is now correctly loaded into the lab computers.

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

This command loaded 9 of the text examples available from the corpora package (only a small number of them!). It has used the variable names text1 through text9 for these examples, and already assigned them values. If you type the variable name, you get a description of the text:

```
>>> text1
```

The variables sent1 through sent9 have been set to be a list of tokens of the first sentence of each text.

```
>>> sent1
```

Note that the first sentence of the book Moby Dick is "Call me Ishmael." and that this sentence has been already separated into tokens in the variable sent1.

Searching Text

The text data structure has a number of functions to operate on text. One is called "concordance", and it will search for any word that you give to the function and show you the occurrences and some surrounding context.

```
>>> text1.concordance("monstrous")
```

Observe the use of the arrow keys with the enter key to select and modify previous lines in Python, and try a similar example.

```
>>> text2.concordance("affection")
```

Another function is “similar” which finds all the words that are used in the same context as the one given, where the context is the word before and the word after.
>>> text1.similar("monstrous")

We can use this to compare how the same word is used differently in other texts.
>>> text2.similar("monstrous")

Counting Vocabulary

Each text from the books was separated into a list of tokens, and this is one of the first NLP processing steps. The tokens usually consist of words and all the punctuation and other symbols occurring in the text. To further investigate text, we can count the occurrences of words.

We start by using the Python length function, “len” to tell us how many things are in a list. (Strictly speaking, each text variable is an object of type nltk.text.Text, which contains the text string and some other functions, but we’re trying not to explain much programming here.)
>>> len(text3)

>>> len(text4)

Now this is the total number of tokens, and we might also want to find out how many unique words there are, not counting repetitions. The Python “set” function removes the repetitions, and we can apply the “sorted” function to that, returning the resulted sorted list of tokens. If we type the following, lots of words will flash by on the screen.
>>> sorted(set(text3))

Or we can just find the length of that list.
>>> len(sorted(set(text3)))

Or we can specify just to print the first 30 words in the list of sorted words:
>>> sorted(set(text3))[:30]

Now let’s compute the ratio of the total number of tokens to the number of unique tokens and we’ll get an average of how many repetitions there are for each word. First we get a division operator that uses real arithmetic (aka floating point) instead of integer and then we divide to get the ratio.

>>> from __future__ import division

>>> len(text3) / len(set(text3)
(On average, each word is used about 16 times.)

Now let's search for and count occurrences of particular words and compare that to the total number of words.

```
>>> text3.count("smote")
```

Compute the fraction of the number of occurrences of the word compared with the total number of words and then multiply by 100 to get a percentage.

```
>>> 100 * text3.count('smote') / len(text3)
```

How does this compare with a more common word, such as the word "a"?

```
>>> 100 * text3.count('a') / len(text3)
```

[Can everyone copy and paste these examples?]

Try it Out:

1. How many times does the word "lol" occur in text5? What is the percentage of its occurrences in the text? [Warning: text5 is uncensored chat]

Think of another word to find occurrences and get the number of occurrences and its percentage in the text. Save the word, the number of occurrences and its percentage in the text to post at the end of class.

Processing Text

In the first part of this lab, we counted words from text that had already been tokenized, i.e. separated into words. Now we'll look at some text examples that we will need to tokenize.

In addition to the examples that we imported for the NLTK book above, the NLTK has a number of other corpora, described in Chapter 2. In order to see these, type in

```
>>> import nltk
```

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

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

For purposes of this lab, we will work with the first book, Jane Austen's "Emma". First, we save the first fileid (number 0 in the list) into a variable named file1 so that we can reuse it:

```
>>> file1 = nltk.corpus.gutenberg.fileids( ) [0]
>>> file1
```

We can get the original text, using the raw function:

```
>>> emmatext = nltk.corpus.gutenberg.raw(file1)
```

```
>>> 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 that separates by white space and also by special characters (punctuation):

```
>>> emmatokens = nltk.wordpunct_tokenize(emmatext)
>>> len(emmatokens)

>>> emmatokens[:50]
```

Let's decide that we want to count upper case words such as "They" to be the same as a lower case "they", and to simplify things, we won't worry about proper names. So we convert all the tokens to their lower case version and call them words:

```
>>> 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.

Frequency Distributions

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. These are described in the NLTK book, at the end of Chapter 1. (Note: we already imported FreqDist when we did the command >>> from nltk.book import *)

One way to make a Frequency Distribution, is to create one 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.

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

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 (most frequent) words.

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

Note the special syntax of Python for a multi-line statement: The first line must be followed by an extra ":", and each succeeding line must be indented by some number of spaces (as long as they are all indented by the **same** number of spaces.)

Optional, if there's time:

In Python, we can make a function that can take any argument, process it and return a result. For an example function, we make a function called "makeAlphaFreqDist" that takes a list of words as an argument and returns a Frequency Distribution which only contains words with all alphabetical characters.

Copy and paste this entire function definition into the Idle window.

```
>>> def makeAlphaFreqDist(words):
  adist = FreqDist()
  pattern = re.compile('.*[^a-z].*')
  for word in words:
    if not pattern.match(word):
      adist.inc(word)
  return adist
```

Line-by-line explanation:
Make a new empty frequency distribution called adist.
Make a regular expression pattern that will match any word that contains a non-alphabetical character.
Loop over all the words
    If the pattern doesn't match the word, then it doesn't contain any non-alphabetical characters and we add it to the frequency distribution .inc,
        which adds the word and adds one to its count
After all the words are processed, return the new distribution as the result.

Apply the new function to emmawords and look at the first 50 words:

```
>>> adist = makeAlphaFreqDist(emmawords)
>>> adist.keys()[:50]
```

Print out the 30 top words:

```
>>> for key in adist.keys()[:30]:
   print key, adist[key]
```

Try it out:

The text that we first imported from NLTK book are already separated into lists of words. Create a frequency distribution for the words in text1 from the NLTK book (Moby Dick) by applying FreqDist directly to text1. For example:

```
>>> mbdist = FreqDist(text1)
```

Use text1 as shown or pick one of the other texts and make a FreqDist. Print out some portion of the keys and pick several words and look at the frequencies.

If we did the optional part with makeAlphaFreqDistr, try it on one of the texts.

Exercise to submit this week:

Go to Blackboard and find the Discussion for the second week exercises (renamed from the first week). Create a post in which you:

1. State which text, word, frequency and percentage you got in the first "try it out".
2. State which text you used in the second "try it out" and give the frequencies of two of the words.