NLP Lab Session Week 1
Word Frequencies and Frequency Distributions from text in NLTK
August 31, 2016

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 labs, using the command line interpreter and Jupyter notebook, but we will also develop python programs for assignments.

For Windows: open a command prompt window
For Macs: open a terminal window

At the prompt (whatever it is), type python:

```
%  python
```

This starts a python interpreter session where you can type bits of python at the prompt:

```
>>>
```

First we go to the NLTK download page and make sure that we have nltk_data installed.

```
>>> import nltk
>>> from nltk.corpus import brown
>>> brown.words()
```

Alternatively, you could run an IDLE window on PCs, or if you already know about iPython notebook, that should be fine as well.

You will probably want to work by having a Python interpreter 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/. 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.

[In the future, we will also use the Jupyter Notebook for developing examples.]

Getting Started in Python and NLTK

Start by typing a couple of examples of **arithmetic** into the Python interpreter. For example, "1 + 2" is an arithmetic expression and Python will compute a result:

>>> 1 + 2

Note that if you want to type in a **string** of text, you surround the string with quotes.
>>> 'hello'

[In Python, you can usually also use double quotes. But note that whenever you copy/paste quotes from a Word of PDF document, you may get non-programming quotes, and you may have to type them directly from the keyboard.]

In programming, when we have a value of some type (like the number 3 or the string 'hello'), we can save that value by assigning it to a **variable.**
>>> num = 1 + 2
>>> num
In this example, the name of the variable is "num" and its value is 3.

Now define a string and assign it to a variable:
>>> sentstr = "I like NLP"

Print the contents of this variable in two ways, first by simply typing the variable name and pressing enter, then by using the print statement.
>>> sentstr
>>> print (sentstr)

Try adding the string to itself using sentstr + sentstr, or multiplying it by a number, e.g., sentstr * 3. Notice that the strings are joined together without any spaces. How could you fix this?

Define a list and assign it to a variable:
>>> sentlist = ["I", "like", "NLP"]

What happens when you try adding sentstr + sentlist?
How do you add "very much" at the end of sentstr and sentlist?

Although you don't explicitly declare types in python, it always assigns a type, aka 'class' to any value or variable. You can use the function 'type' to find out what it is.

>>> type(num)
>>> type(sentstr)
>>> type(sentlist)

In lab this week, we will also use some functions and operators on lists. First, we can use the length function "len" to find how many items are in a list.

>>> len(sentlist)

In order to get individual items from the list, we use indexing, where we put an index number in square brackets after the list name. The index numbers start at 0.

```
>>> sentlist[0]
>>> sentlist[1]
```

We get an error if we use an index number that doesn't have a value:

```
>>> sentlist[5]
```

We can also use index slicing to indicate a sequence of index numbers. Here we select the first two items in the list.

```
>>> sentlist[ :2]
```

You can learn more details about lists and also about the type dictionary in the readings for this week.

Python and NLTK Resources

Python WikiBooks: http://en.wikibooks.org/wiki/Python_Programming
Python tutorial from python.org: http://docs.python.org/2/tutorial/

NLTK book: http://nltk.org/book/

After today's lab session, if you want to know more about Python variables, strings and lists, there will be readings on these topics in the resources.

Working with Text from the NLTK Gutenberg corpora

The NLTK has a number of corpora, described in Chapter 2 of the NLTK book, that we can use to process text. 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 file0 so that we can reuse it:

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

We can get the original text, using the raw function. This returns the text as a string of characters, and the length function tells us how many characters.

```
>>> 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]
```

Processing Text

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 50 **tokens**

```
>>> emmatokens[:50]
```

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)
```

In emmatokens, the tokens "Miss" and "miss" are counted as two different words. After applying lowercase to emmatokens, the two are counted as the same word in the vocabulary.

We can further view the words by getting the unique words and sorting them into alphabetical order.

```
>>> 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.

[Side note on Python: Here we defined emmawords by using a Python list comprehension. It would be equivalent to define emmawords by starting with an empty list and using a "for" loop that kept appending the lower case of each word in emmatokens .

Note that a Python "for" loop can be used to iterate over every element of a list.

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.) Just hit enter to finish the statement.

Start with an empty list.
```
>>> emmawords = []
```
For each word w in the list emmatokens, add the lower case of that word to the emmawords list.
```
>>> for w in emmatokens:
   emmawords.append(w.lower())

>>> emmawords[:50]
```
End note.]

Now that we have a list (in order) of all the words in Emma, we may want to find the counts of different words in the text. The python function "count" will do that and here are some examples:

```
>>> emmawords.count('food')
3
>>> emmawords.count('love')
116
>>> emmawords.count('she')
2336
>>> emmawords.count('the')
5198
```

Using NLTK Frequency Distributions to Count Words

In order to see the frequencies of all the words, or to see the top frequency words, we are going to create an example of a Python dictionary. We will create a dictionary that has a list of (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.

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.

What we want is a list that has pairs such as:

[('food', 3), ('love', 116), ('she', 2336), ('the', 5198), . . .]

This list will be put into a dictionary where we can look up words and get their count.
The NLTK has a data structure called a Frequency Distribution, FreqDist, that creates such a dictionary from a list of words like "emmawords".

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.

```
>>> fdist = FreqDist(emmawords)
>>> fdistkeys = list(fdist.keys())
>>> fdistkeys[: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']
```

The FreqDist module has a function to return the most frequent words in the corpus. It is a list of (word, frequency) pairs, and we can use this list to print the most frequent words.

```
>>> topkeys = fdist.most_common(50)
>>> for pair in topkeys:
    print(pair)
```

In the future, for comparing documents, we will also show how to normalize frequencies with the length of the document.

Try it out:

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

- get the text with nltk.corpus.gutenberg.raw()
- get the tokens with nltk.word_tokenize()
- get the words by using w.lower() to lowercase the tokens
- make the frequency distribution with FreqDist
- get the 30 top frequency words with most_common(30) and print the word, frequency pairs

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 first 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.

Reading Assignment: Python Variables, Strings, Lists and Dictionaries

NLTK book: http://nltk.org/book/
Read the sections called "Indexing Lists", "Variables", and "Strings".
(Optional) Read the section 5.3, Mapping Words to Properties using Python Dictionaries.

Or
Python WikiBooks: http://en.wikibooks.org/wiki/Python_Programming
For more detail on these topics, read the sections on Variables and Strings, and on Lists, but note that these sections have more detail than you need to know for this course.
(Optional) Read the section on dictionaries, under Data Types in the menu.

It's o.k. if you just read the parts on variables, strings and lists, and we will discuss more on dictionaries next week.