Exploring Text

Python can be run either on the command line to run Python programs (that have been written in a text editor) 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/IPython/Spyder window.

You will probably want to work by having an IDE 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 many examples are taken from.

In the following, examples for you to try are given following the Python Idle prompt of `>>>`. If you use the iPython interpreter, your prompt would be `[n]:` instead.

Getting Started in Python and NLTK

Start by typing a couple of examples of arithmetic into the Python interpreter. For example:

```
>>> 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.

Next, you use the Python "import" statement to load the data used in the book examples into the Python environment:

```
>>> 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.

```
>>> 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)
```

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]
```

We probably want to use the lowercase versions of the 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 – Regular Expressions to the Rescue! (as in xkcd _), but we'll work on that later.