NLP Lab Session Week 4
September 17, 2014

Reading and Processing Test, Stemming and Lemmatization

Getting Started

In this lab session, we will use two saved files of python commands and definitions and also work together through a series of small examples using the IDLE window and that will be described in this lab document. All of these examples can be found in python files on the blackboard system, under Resources.

Bigram.py, processtextfile.py, Labweek4examples.py, desert.txt, Smart.English.stop

Put all of these files in one directory (and remember where that is!) and open an IDLE window.

This section uses material from the NLTK book, Chapter 3, where they recommend to start the session with several imports:

```
>>> from __future__ import division
>>> import nltk, re, pprint
```

Reading Text from the web or from a file

So far, we have used a small number of books from the Gutenberg Project that were provided in the NLTK. There is a larger collection available online, and you can see what is available in

http://www.gutenberg.org/catalog/

Text number 2554 in the catalog is the book "Crime and Punishment" by Dostoevsky, and we can access this text directly from Python by using url libraries to connect to the url and to read the content.

```
>>> from urllib import urlopen
>>> url = "http://www.gutenberg.org/files/2554/2554.txt"
>>> raw = urlopen(url).read()
>>> type(raw)
<type 'str'>
>>> len(raw)
1176893
>>> raw[:100]
```

Note that the text obtained from this web page is pretty clean, that is, it does not contain html tags or other web formatting that would interfere with text processing. So from the raw text, we could proceed with tokenization and further steps of text processing.

But if there is html formatting, we may have to take further steps to clean up the text. Let's look at the example given in the NLTK book: http://news.bbc.co.uk/2/hi/health/2284783.stm

```
>>> blondurl = "http://news.bbc.co.uk/2/hi/health/2284783.stm"
>>> html = urlopen(blondurl).read()
>>> html[:1000]
```

Note that there is a lot of tags and formatting that is not text. One simple thing is to use the NLTK function clean_html to just remove html tags. If we wanted to do more sophisticated clean-up, we could investigate the python library BeautifulSoup.

```
>>> braw = nltk.clean_html(html)
>>> btokens = nltk.word_tokenize(braw)
>>> tbokens[:100]
```

For further cleanup, we could try to eliminate text representing menus and ads by hand, either on the string or on the tokens. Here we try to just pick out the tokens representing the text portion by inspecting the text and doing some trial and error.

```
>>> braw = nltk.clean_html(html)
>>> btokens = nltk.word_tokenize(braw)
>>> btokens[:100]

>>> btokens = btokens[96:399]
>>> btokens[:100]

>>> btokens = btokens[5:]
>>> btokens[:100]
```

Finally, another way to get text in is for you to collect the text into a file and read this file. To see this, I have collected the text of a news stories by copying it directly from the web page and pasting it into a text file: desert.txt

Now reading a file is very simple in Python and the only difficulty is making sure that Python can find the file in the right directory. One option is to put the file that you want to read in the current directory of IDLE. To find out what that directory is, we can do:

```
>>> import os
>>> os.getcwd()
```

Then we could put desert.txt in that directory and just open it:

```
>>> f.open('desert.txt')
```

Or instead we could add the path to the directory of the file by adding to sys.path, as shown later in this lab.

Or we can just read the file by giving the full path. Here you must substitute the path to the file desert.txt on your own lab machine.

```
>>> f = open('H:\\NLPclass\LabExamplesWeek4\desert.txt')
>>> text = f.read()
```

For Mac users, the path will use forward slash. For example, this is the path that I use on my mac laptop:

```
>>> f = open('/Users/njmccrac1/AAAdocs/NLPfall2014/labs/LabExamplesWeek4/desert.txt')
>>> text = f.read()
```

Module of Bigram functions

When we processed bigrams from text last week, we defined a function to filter out tokens consisting entirely of non-alphabetical characters. However, in the Emma text, we saw that there were weird tokens containing special characters and starting with underbars that were given high scores by the mutual information (PMI) scoring function.

So instead, we might filter out the tokens that don't consist entirely of alphabetical characters. This will filter out words like 'Syracuse-based', but it will also filter out those annoying tokens with many underbars at the beginning or end, which skews out mutual information findings.

To make it easier to use any functions that we have defined, we can place them into one python file, which python would call a Module. The name of the file is Bigram.py and if the file is in the same directory as another Python program, you can say "import Bigram" to access the function definitions. Then you use the name of the module to use the function names, for example, Bigram.alpha_filter.

In your IDLE window, use the File menu to open the Bigram.py file. Now if you want to use the functions in the Bigram file in your IDLE window, you will need to add the path to the Bigram file to the system path.

To see which directories the system will search to find programs for the IDLE window:

```
>>> import sys
>>> sys.path
```

Suppose that you added the Bigram.py file to a directory on the H: drive called NLPclass. To add that directory to your path:

```
>>> sys.path.append('H:\\NLPclass\ LabExamplesWeek4')
```

For those working on a Mac, my path on my laptop is:

```
>>> sys.path.append('/Users/njmccrac1/AAAdocs/NLPfall2013/labs/LabExamplesWeek4')
```

Now you could use the functions such as alpha_filter in your IDLE window without copying the definition to the window.

Mutual Information

Review the definition of association ratio from the Church and Hanks paper, and the mutualinfo function. I have put this function, along with the other unigram and bigram frequency distribution functions into the Bigram.py file.

From your IDLE window under the File menu, use Open to open the Bigram.py file into another IDLE window. Read through this module through to see the function definitions.

Now we will use the text that we got from Project Gutenberg from the novel Crime and Punishment. Check that your text is still in a variable called raw. Then tokenize the text, but this time, we won't use lowercase.

```
>>> raw[:200]
>>> crimetokens = nltk.word_tokenize(raw)
>>> len(crimetokens)
```

In order to use our filter functions, we can
```
>>> import Bigram
```

Now we can run our text processing steps using filter definitions from the module.

```
>>> from nltk.collocations import *
>>> bigram_measures = nltk.collocations.BigramAssocMeasures()
>>> stopwords = nltk.corpus.stopwords.words('english')
```

Make a bigram finder and apply the non-alphabetical and stopword filters.

```
>>> finder = BigramCollocationFinder.from_words(crimetokens)
>>> finder.apply_word_filter(Bigram.alpha_filter)
>>> finder.apply_word_filter(lambda w: w in stopwords)

# score by bigram frequency
>>> scored = finder.score_ngrams(bigram_measures.raw_freq)
>>> scored[:30]

# Or for NLTK 3.0, sort the results first:
>>> sorted(scored, key=itemgetter(1), reverse=True)[:30]
```

Now run the scorer again to get bigrams in order by Pointwise Mutual Information.
```
>>> scored = finder.score_ngrams(bigram_measures.pmi)
>>> scored[:30]
```

```
# Or for NLTK 3.0, sort the results first:
>>> sorted(scored, key=itemgetter(1), reverse=True)[:30]
```

We can run this all again with the other alphabetical filter.

[Note that if you make changes to the Bigram module, you should save it and then use the reload function, instead of importing it again.

```
>>> reload(Bigram)
```

]

Stemming and Lemmatization

For this part, we will use the crime and punishment text from the Gutenberg Corpus as we did above.

Note that crimetokens has words with regular capitalization and we can make crimewords to have lower-case words with no capitalization.

```
>>> crimewords = [w.lower( ) for w in crimetokens]
```

NLTK has two stemmers, Porter and Lancaster, described in section 3.6 of the NLTK book. To use these stemmers, you first create them.

```
>>> porter = nltk.PorterStemmer()
>>> lancaster = nltk.LancasterStemmer()
```

Then we'll compare how the stemmers work using both the regular-cased text and the lower-cased text.

```
>>> crimePstem = [porter.stem(t) for t in crimetokens]
>>> crimePstem[:200]

>>> crimeLstem = [lancaster.stem(t) for t in crimetokens]
>>> crimeLstem[:100]
```

Try the same examples with the crimewords to see lower case version of the stemmers.

The NLTK has a lemmatizer that uses the WordNet on-line thesaurus as a dictionary to look up roots and find the word.

```
>>> wnl = nltk.WordNetLemmatizer()
>>> crimeLemma = [wnl.lemmatize(t) for t in crimetokens]
>>> crimeLemma[:200]
```

Processing Text from Files

Open the processtextfile.py and read through it to observe how it reads text from the file desert.txt and how it creates a stop word list from the file Smart.English.stop. Look at the Smart.English.stop words file in a text editor. Note that processtextfile then runs the bigram finder and scorers.

Run the processtextfile by selecting Run Module from the Run menu. The results will appear in your main IDLE window (which restarts the session and erases all the definitions that you already made).

Modify the code to try the mutual information function with and without stopwords, by using the comment character # to switch between the two statements. Also try thresholds of 3, and perhaps 5.

Note that another list of stopwords is available in NLTK: nltk.corpus stopwords. This list has 127 words and can be accessed as follows:

```
>>> from nltk.corpus import stopwords
>>> nltkstoplist = nltk.corpus.stopwords.words('english')
>>> nltkstoplist
```

For your homework, if you develop a file similar to the processtextfile in lab, then you can rerun your examples without as much typing into IDLE.

Note that you can also run this python file by using python directly from a terminal/command prompt window. That is, on a PC, open a command prompt window and change directory until you get to the LabExamples4 directory. Then run (I think):

```
%  processtextfile.py
```

On a Mac, open a terminal window and change directory until you get to the LabExamples4 directory. Then run:

```
% python processfile.py
```

Ideas for Homework

For documents that come from NLTK corpora, read the NLTK book sections from Chapter 2 on the different corpora.

Also note that Chapter 2 discusses how to load your own text with the PlainCorpusReader, or you can just read text from files as we did in the lab example with desert.txt.

Example of using word frequencies to analyze text:
Nate Silver's analysis of State of the Union Speeches from 1962 to 2010.

http://www.fivethirtyeight.com/2010/01/obamas-sotu-clintonian-in-good-way.html
Here is a statement of the question that he is trying to answer by looking at word frequencies to compare the SOTU speech in 2010 with earlier speeches:
"What did President Obama focus his attention upon and how does this compare to his predecessors?"

[And you may find it interesting to note also the criticism of the technique at http://thelousylinguist.blogspot.com/2010/01/bad-linguistics-sigh.html. I am not expecting you to be experts in how you categorize the words or bigrams that you use in your analysis (and I actually think that Nate Silver was not too far off the mark in picking categories of words to answer his question.)]

How is this example different from your assignment? It is different because it does more comparisons that you are required to do and only uses word frequencies while you must also look at bigram frequencies and mutual information. But if you choose the analysis option, this is the type of thing that you are aiming for. Also, you can just use word lists and frequencies/scores without making colored graphs.

Also note that you can collect your own text from just about anywhere. For a (too short) example of this, see the potato chip marketing example.

Discuss the programming option.

There is no exercise for today.