Introduction to Text Analysis with Python

Data Visualization for Architecture Urbanism and the Humanities

March 31, 2017

Michelle McSweeney

Major steps in doing text or language analysis

1. Getting the data
2. **Turning the data into numbers**
3. Analyzing the data

Question

What parts of language (spoken, written, texted) can you count?

What numbers can you come up with beyond counting?

Text Editor Functions

- Word counts
- Character Counts
- Word-in-context (text wrangler & sublime)

Moving Beyond the Text Editor with Python + NLTK

The Natural Language ToolKit

Open the Terminal or Command Prompt

In [ ]:
```
import nltk
from nltk.book import *
```

Concordance: Words in their contexts

In [ ]:
```
text1.concordance("love")
```

Similar: Words that appear in a similar environment as the target word

Might do this with two different texts

In [ ]:
```
text1.similar("hope")
```

In [ ]:
```
text2.similar("hope")
```

Common Contexts: Two words that appear in similar environments as each other

In [ ]:
```
text1.common_contexts(['whale','monster'])
```

In [ ]:
```
text1.dispersion_plot(["whale", "monster"])
```

The Dispersion Plot is often **BEHIND** the shell.

If you did not get a Dispersion Plot:

```
 * put up your sticky note
 * type into the next line: import matplotlib as plt
```

CLOSE THE DISPERSION PLOT BEFORE WE MOVE ON

Now we are going to calculate a bunch of things related to my text without NLTK - just using Python

Count a specific word - how many times does this sequence of characters occur in my document?

In [ ]:
```
text1.count("love")
```

How many **tokens** are in my text?

tokens are unique sequences, let's start with an example:

"love", "bowie", "Bowie", "!" and ":)" are all unique **tokens**

In [ ]:
```
len(text1)
```

How many **unique** words are in my text?

- first make a set that groups all the "words" together (numbers, punctuation sequences, etc.) - this groups together **types**.
- Token = instance
- Type = more general ("bowie" and "Bowie" are different types - why?)

In [ ]:
```
set(text1)
```

I like to **sort** my **set** so I know what I have:

In [ ]:
```
sorted(set(text1))
```

Now count the number of items in that set to find the number of unique words

In [ ]:
```
len(set(text1))
```

Lexical Density: the number of unique tokens divided by the total number of words.

This is a descriptive measure of language register or grade level approximations.

In [ ]:
```
len(set(text1))/len(text1)
```

Frequency Distribution is a probability object that Python deals with. We will use it to make a graph of the most common words.

In [ ]:
```
my_dist = FreqDist(text1)
```

Since nothing appears, I like to go check for it

In [ ]:
```
type(my_dist)
```

Now let's **plot** the graph

In [ ]:
```
my_dist.plot(50,cumulative=False)
```

It may be a little easier to look at as a list

In [ ]:
```
my_dist.most_common(10)
```

That actually doesn't tell us very much.

PREVIEW
*We need to remove the **stopwords** to learn more.*

In [ ]:
```
love_words = ['love', 'joy', 'hope', 'amor']
```

In [ ]:
```
my_list = []
for word in love_words:
    if word in text8:
        my_list.append(word)
    else:
        pass
```

In [ ]:
```
print(my_list)
```

Let's pull a book in from the Internet

Project Gutenburg is a great source! www.gutenberg.org

Text 37472 is Zanzibar Tales translated by George W. Bateman

To make a book into a Text NLTK can deal with, we have to:

- open the file from a location
- read it/decode it
- tokenize it (go from a string to a list of word)
- nltk.Text()

In [ ]:
```
#import the urlopen command
from urllib.request import urlopen
#set the url to a variable
my_url = "https://www.gutenberg.org/files/37472/37472.txt"
```

In [ ]:
```
#open the file from the url
file = urlopen(my_url)
#read the opened file
raw = file.read()
```

In [ ]:
```
#specify which decoding to use. (usually utf-8)
zt = raw.decode('utf-8')
```

In [ ]:
```
#check the type to be sure it worked. I expect a string now.
type(zt)
```

If this doesn't work, or you are dealing with messier text, check out the ftfy library Fixes Text For You

http://ftfy.readthedocs.io (http://ftfy.readthedocs.io)

In [ ]:
```
#split the string into words with word_tokenize (uses spaces to distinguish words
)
zt_tok = nltk.word_tokenize(zt)
```

In [ ]:
```
#check to make sure it worked
type(zt_tok)
```

In [ ]:
```
#get an idea of how big the file is
len(zt_tok)
```

In [ ]:
```
#look at the first 10 words to be sure its correct
zt_tok[:10]
```

Yuck! That just looks like metadata!

Removing metadata involves using Regular Expressions

Regular Expressions are saved for another day. They are powerful but complicated

In [ ]:
```
#I want to get rid of that **intro** metadata!!

#I found the number 177 by copying this into Text Wrangler
#Text Wrangler counts words, characters, and spaces

#To do this for many files, you need Regular Expressions

zt_tok[177:188]
```

In [ ]:
```
#turn the list of words into a text nltk can recognize
zt_text = nltk.Text(zt_tok[177:])
```

In [ ]:
```
#check to make sure it worked
type(zt_text)
```

In [ ]:
```
#get an idea of how big the file is
len(zt_text)
```

One step further! **Part-of-Speech Tagging**

NLTK uses the Penn Tag Set.

There are better options (i.e., tree tagger and polyglot), but this illustrates the idea.

In [ ]:
```
#make a new object that has all the words and tags in it
zt_tagged = nltk.pos_tag(zt_text)
```

In [ ]:
```
print(zt_tagged[:10])
```

This doesn't look like a dictionary! What's going on??

In [ ]:
```
type(zt_tagged)
```

We have a **list of tuples**

I'm going to put it in a for-loop

Have to deal with this in a special way
(a, b) in my_list

In [ ]:
```
#function to determine what is the most common tag in Zanzibar Tales
def commontag(taggedbook):
#create an empty dictionary
    tag_dict = {}
#for every word/tag combo in my list,
    for (word, tag) in taggedbook:
        if tag in tag_dict:
            tag_dict[tag]+=1
        else:
            tag_dict[tag] = 1
    print(tag_dict)

commontag(zt_tagged)
```

I wish I would have made that return an ordered dictionary!!!

Let's add a line of code with OrderedDict in it'

In [ ]:
```
from collections import OrderedDict

def commontag(taggedbook):
#create an empty dictionary
    tag_dict = {}
#for every word/tag combo in my list,
    for (word, tag) in taggedbook:
        if tag in tag_dict:
            tag_dict[tag]+=1
        else:
            tag_dict[tag] = 1
    tag_dict = OrderedDict(sorted(tag_dict.items(), key=lambda t: t[1]))
    print(tag_dict)

commontag(zt_tagged)
```

How do you know to put all that other stuff in (i.e., sorted, lambda, etc)?!?

Read the docs

https://docs.python.org/3.1/whatsnew/3.1.html (https://docs.python.org/3.1/whatsnew/3.1.html)

So far, we have counted things in our texts by looking at

- Concordance
- Words in similar environments
- Words in common contexts
- Unique words
- Length of words

Then we performed some operations, but still counted things:

- Frequency Distributions
- Lexical Density
- Found words from a list in a text
- Part-of-Speech Tags

Now we will perform operations on the Text itself before doing those operations

To get a better idea of the content of a text, it's usually best to exclude stopwords

Stopwords perform grammatical functions, but have limited semantic content.

(the, a, at, in, of, with, etc.)

Two methods:

- Use a pre-defined list from nltk
- Make your own list and use a for-loop

In [ ]:
```
from nltk.corpus import stopwords
```

In [ ]:
```
#have to tell NLTK that you want the English stopwords
mystops = stopwords.words('english')
```

In [ ]:
```
nostop_text1 = []

#remove stop words
#go through all the words in text1 and save all the non-stopwords
for word in text1:
    if word not in mystops:
        nostop_text1.append(word)
    else:
        pass
```

In [ ]:
```
print(nostop_text1[50:150])
```

Now to:

- remove all that punctuation
- make everything lowercase

In [ ]:
```
#remove punctuation
#go through all the items in text1 (without stopwords), and save everything that
is alphabetic
nopunct_text1 = []
for word in nostop_text1:
    if word.isalpha():
        nopunct_text1.append(word)
    else:
        pass
```

In [ ]:
```
lower_text1 = []

for w in nopunct_text1:
    lower_text1.append(w.lower())

print(lower_text1[:50])
```

There's another way to write this, if this is easier to understand:

In [ ]:
```
new_text1 = [w for w in text1 if w not in mystops]
```

Now you have a nice clean text.

Let's look at the lexical density of that text

In [ ]:
```
len(set(new_text1))/len(new_text1)
```

That's much higher!!

This is the density of the whole book without the stop words, which better represents the variety of words used

What if I want to read in my OWN corpus?

In [ ]:
```
f = open("/Users/mam/books/hungerGames/catchingFire.txt", 'r')
my_file = f.read()
```

In [ ]:
```
type(my_file)
```

Going Forward

- Use a text editor to write complete programs
  - Run these in the terminal
- Use Spyder to write complete programs
- Often save the program you write in the same file as the file you will be working with to shorted the path.

How do I know where to go?!?

- http://www.nltk.org/book_1ed (http://www.nltk.org/book_1ed)
- http://www.nltk.org/ (http://www.nltk.org/)
- Play!
  - http://techblog.about.com/post/140231383537/analyzing-the-language-of-the-presidential-debates (http://techblog.about.com/post/140231383537/analyzing-the-language-of-the-presidential-debates)
  - http://andybromberg.com/sentiment-analysis-python/ (http://andybromberg.com/sentiment-analysis-python/)
  - etc.

In [ ]:
```

```