Assignment: Lab2
By: Alena Borisenko
Created: September 25th, 2017
Submitted: September 27th, 2017
==========

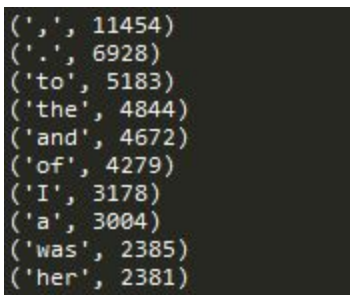
Authors selected:

A1 = Austen, A2 = Edgeworth, A3 = Melville

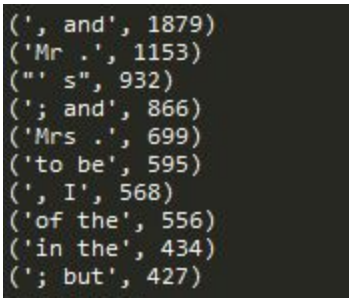
For some reason I assumed that the gutenberg corpus had a lot more books in it, so I spent a little too much time trying to find three writers with known love-hate relationships. I settled on Nabokov, Dostoevsky and Tolstoy because Nabokov openly despised Dostoevsky and was an admirer of Tolstoy's "Anna Karenina while Tolstoy regretted not being able to meet Dostoevsky prior to his death. Sadly, none of the three appeared in the corpus, so all that went out the window. In the end, I decided to make a female writer choose between lunch with a male writer or another fellow female writer.

Analytics performed:

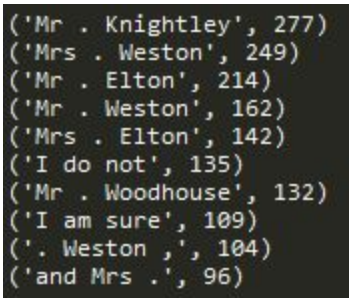
I decided to use dictionaries/hashing to create my n-grams. I took advantage of NLTK's sents function to read in each text as an array of sentences. I chose sentences over words so that words would rely only on the context of words in a given sentence. The following are the top tens of the initial counts I got from A1's book:



single words

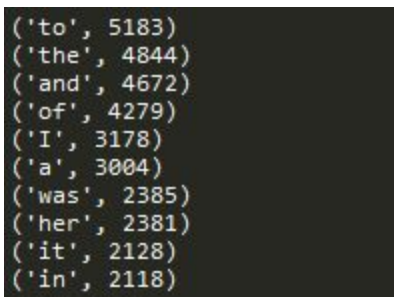


double-words seqs

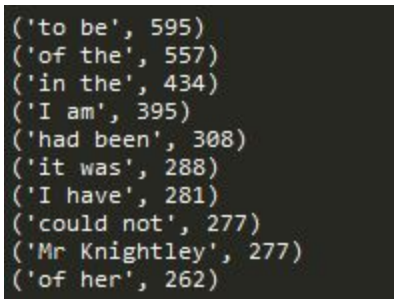


triple-word seqs

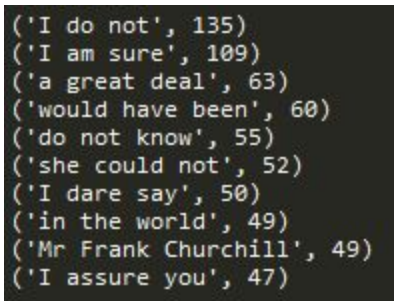
The obvious problem is the fact that punctuation is counted as words. The next step was removing those "fake" words from the counts and the updated result was much nicer:



single words



double-words seqs



triple-word seqs

After making sure that the counts/hashing made sense, I changed the way n-grams were set up to make them less readable but (hopefully) more useful. Now instead of a key-count correspondance each n-gram contained context-key (the n-1 words before current) and a dictionary of possible words to follow along with their counts. So, for 3-gram something like:

{‘Alena likes’ : {‘penguins’ : 100, ‘tea’ : 50, ‘sushi’ : 25},
‘weather is’ : {‘nice’ : 200, ‘awful’ : 400, ‘gross’ : 20}}

This was necessary in order to later assign probabilities to words during dialogue generation, which was done via the NumPy choose function.

In order to decide who A1 picks, I got the top 100 single words in A1’s unigram and looked it up in A1’s, A2’s and A3’s bigrams. If A2’s most common bigram result for a top 100 word matched with A1’s, A2 would get points equal to the number of occurrences (same for A3). The author that matched the most words that are most-likely-to-follow-a-top-100-word would win and have a dynamically generated 4-line dialogue with A1. I then repeated the process for the author that got left out.

Results:

A3 is the winner when A1 is choosing. A3 is also the winner when A2 is revenge-choosing. Sample program output should look something like this:

```
λ py -3 prose.py
-------------a1-------------
-------------a2-------------
-------------a3-------------
-------a1 is choosing-------
4690 6585
a3 wins
--------DIALOG START--------
A1:  Perry had said on the stairs by Jane herself was
A3:  the science of whales and dragons are strangely jumbled together
A1:  approaching Randalls and secured him the pleasantest proof of
A3:  it comes to that hated fish remnants and marching
---------DIALOG END---------

-------a2 is choosing-------
5220 7761
a3 wins
--------DIALOG START--------
A2:  her daughter s drawing out his knife and cuts it
A3:  foul line rammed down the promenade of the rotten line
A2:  that the basket not entirely however with a small room
A3:  conduct he kept it averted for some little chat with
---------DIALOG END---------
```

Things I would do differently:

POS-tagging would probably make for a stronger argument towards writing style similarity. In addition, POS patterns could result in more sensible conversations between the authors. In the interest of time, I simply used literal word/phrase matching.