Gregory Clancy

10/30/2019

Project Overview:

I chose to analyze 4 texts written by Charles Dickens and compare them to a 5th text written by Leo Tolstoy. All the texts were obtained through Gutenberg.org. I processed the texts by grabbing them from the url with a script, removing the escape and special characters along with 200 of the most popular words from the English language, and then finding the 10 most frequent words for each text.

Implementation:

My implementation strategy had a few key elements where I branched out from what we had done in class. For starters, I decided to use the pickle library to get around the download limitation from Gutenberg.org. The key change that I made here from the example code was my decision to store the texts in a dictionary by title before dumping them to pickle. This was advantageous because it allowed me to pass only a single dictionary to most of my functions which made my code easier to understand and troubleshoot. Secondly, I chose to work with a new library in order to process the text. I used the re (regular expressions) library to efficiently remove escape characters from the utf-8 format, which I was struggling to do with just for loops. By combining this new library with my knowledge of dictionaries I was able to take a dictionary with titles as keys and the entire text as a value in string form, and create a second dictionary that kept the titles as keys but had a second dictionary as a value, which was a histogram of the word count. From here I was able to pass this dictionary of dictionaries to a function that returned a list of tuples of the 10 highest key value pairs for each text. This final step could have been very simple, but in order to better evaluate the content of the book I decided to remove about 200 words that are very common in the English language such as “the” “at” and “and”. These words did not help me gain any insight about the content of the texts, so I made a text file full of common words like these, and omitted them from my top 10 frequency rankings so that I could look at more relevant words instead.

When I was designing my implementation strategy I decided that dictionaries would be the best way to evaluate multiple texts because you only need to write the function to analyze a single key value pair, and then map it to the rest of the dictionary. This means that my program can analyze any number of texts as long as the user provides an initial dictionary of title and url key value pairs. I was worried about hitting my limit on Gutenberg.org, so I kept it to a conservative 5 texts, but it is completely scalable. I feel that dictionaries were a better choice than using lists for this project because many of the list techniques to do with frequency are slower than dictionaries. For instance in order to make a histogram from a list you would have to use a loop to run a count() on every unique element of the list. This would result in a lot more calculations, and a much slower program. In addition, there are no keys in a list, so if I had decided to use lists instead of dictionaries it would be easy to mix up which text I was working on or accidentally overwrite part or all of a list. With dictionaries, this was not a risk because I was able to use nested dictionary to contain everything.

Results:

My results found stark differences between the style of Tolstoy and Dickens. For example, in each of Dickens' 4 texts "little" ranked in the top 10 most frequent words, and was number 1 twice. Old was present in the top 10 3 out of 4 times. On average across all 5 texts, Dickens used the word 'old' 1.2 more than Tolstoy per word written, and the word 'little' 2.6 more times than Tolstoy. Similarly, Tolstoy was much more likely to use the names of his characters than Dickens, with 4/10 of his top words being related to character names, and 6/10 referring to either names or titles (prince, princess, etc.). This points to a more person oriented writing style from Tolstoy that focuses on character development and interaction, and a more descriptive, illustrative style from Dickens. These results show how even very basic text mining can be useful for comparing two or more entities in almost any field. Similar comparisons could be used for something like measuring political topics per time spent debating or at speeches, or to compare two soccer players or teams at different periods in time.

Reflection:

From a process point of view I think I did a good job at branching out and learning new techniques and new libraries within python. I also think I did a good job of building a central structure around dictionaries, and thinking about how each of my functions would handle my chosen data structures. However, some things certainly could have gone better. I ran into significant struggles trying to deal with Tolstoy's text because some other the characters had accent marks, which were getting removed by my re.sub function. Ultimately I was not able to resolve this issue, and even though it did not significantly impact my results, it ate up a lot of time and limited the scope of my project. My analysis was more limited than I would have liked it to be, but that was partially because of working solo, and having lots of work outside this project as well.

I think the main takeaways for me from this project are pickle (super useful!) and re. These libraries took a couple hours to really understand and use effectively, but once I got the hang of them I could turn dozens of for loops into a handful of lines of code. The thing I wish I knew before starting this project was how to better deal with stripping escape characters. I lost several hours trying to figure this out, and I knew deep down that there must be a simple way to do it. In the future I hope I can be better about focusing on the big ideas like trying to try multiple new types of analysis, and less on trying to optimize and streamline my code before the framework is complete. I hope to go back and work on some of these techniques now that my traveling for interviews is over and I can devote the time it takes to practice coding.