Project 3 Write-up and Reflection

Oscar Zhang

Project Overview

Having been curious about old english literatures, I wanted to analyze the familiar story of Romeo and Juliet and compare the frequency of speech between the two houses Capulets and Montague as well as the non-house characters. From a software design aspect, I also wanted to practice working with text files and build a function program from scratch. To avoid running the restriction imposed by project gutenberg, I used pickle to save and pre-load the data for analysis without having to pull the text and query the api every time.

Implementation

Describe your implementation at a system architecture level. You should **NOT** walk through your code line by line, or explain every function (we can get that from your docstrings). Instead, talk about the major components, algorithms, data structures and how they fit together. You should also discuss at least one design decision where you had to choose between multiple alternatives, and explain why you made the choice you did.

To begin, in my textMining.py file, I used Python's pickle module to store a local copy of the text for it to be loaded, parsed and analyzed in the textProcessing.py file. I chose dictionaries to represent my processed data as best represent key-value pairs with increments. Intuitively, the key is the name of the character in the play, and value is the house he/she/they belong to and the the number of times they speak. The structure of my code is divided into 5 components each with their individual function. The first is to create a dictionary of characters and their associated houses. The second component is for parsing the text file/script loaded in to pickle and handle some special cases. The third is to refer back to the dictionary and output the number of characters from each house. The fourth is to calculate the number of times each character speaks. The fifth and last is to use the output from the fourth step to compute the averages of the speech frequency for each house.

Since text from Gutenberg or texts in general varies significantly depending on the genre and writer, I made specific parsing targeted solely for the chosen text. For example, when I came across a few character names that consist of two words rather than one, I needed to hard-code my function to recognize them. Furthermore, I was able to use the feature of plays that it is composed mostly of dialogues and the name of the character comes conveniently before they speak. Running the program on another text, especially one that’s not a play, it would not yield accurate results.

Results

Program output:

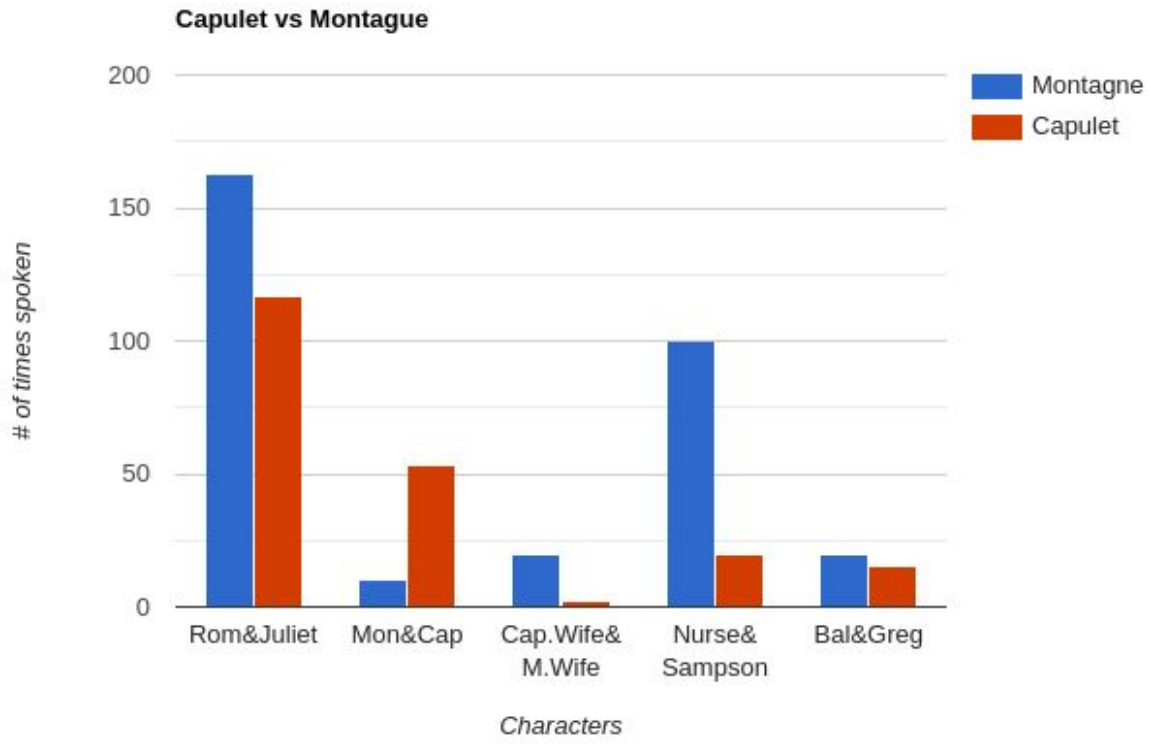
> There are 6 Montagues and 13 Capulets in Romeo and Juliet.{'Chor.': ('Neither', 2), 'Samp.': ('Capulet', 20), 'Greg.': ('Capulet', 15), 'Abr.': ('Montague', 5), 'Bal.': ('Montague', 9), 'Ben.': ('Neither', 64), 'Tyb.': ('Capulet', 17), 'Officer.': ('Neither', 2), 'Citizens.': ('Neither', 2), 'Cap.': ('Capulet', 53), 'Wife.': ('Capulet', 20), 'Cap. Wife.': ('Capulet', 0), 'Mon.': ('Montague', 10), 'M. Wife.': ('Montague', 2), 'Pri.': ('Neither', 0), 'Prin.': ('Neither', 0), 'Rom.': ('Montague', 163), 'Par.': ('Capulet', 23), 'Nurse.': ('Montague', 100), 'Jul.': ('Capulet', 117), 'Mer.': ('Neither', 62), 'Friar.': ('Neither', 52), 'Laur.': ('Neither', 4), 'John.': ('Neither', 5), 'Peter.': ('Capulet', 5), 'Apoth.': ('Neither', 4), '1. Serv.': ('Capulet', 3), '2. Serv.': ('Capulet', 2), '3. Serv.': ('Capulet', 1), '2. Cap.': ('Capulet', 2)}
>
> {'Montague': (6, 289), 'Capulet': (13, 278), 'Neither': (11, 197)}
>
> {'Montague': 48.166666666666664, 'Capulet': 21.384615384615383, 'Neither': 17.90909090909091}

Montague character average # of speaking times: 48.166666666666664

Capulet character average # of speaking times: 21.384615384615383

Average # of speaking times of non-house characters: 17.90909090909

Despite playing equally important roles in the play, Montague characters speak on average significantly more times than Capulet characters, infact, more than twice as much. Surprisingly, non-house characters such as Benvolio speak more than 64 times in the play, hence the average of 17.9091 times comparable the averages times of speaking of Capulet characters.



In the graph, the characters with highest numbers of speaking times are visualized. the main driver of dialogues is unsurprisingly the main character Romeo and Juliet. And the most of the frequently spoken characters are Montagne characters. However, what stood out is the fact the Capulet(the character) has significant more lines than Montagne(the character), despite being an irreplaceable role, Montague(the character) only spoke 10 times. Only 5 sets of characters presented in the graph because there are no more pairable Montagne and Capulet counterparts, and all other house associated characters play a much smaller role.

Alignment

I had a broad learning goal of wanting to explore the process of text analysis. Initially, I wanted to analyze word usage of old English writers like Shakespeare and train a model. But then I realized it might be a more fruitful learning process if I hard code everything myself to perform analysis instead of relying on black-box algorithm like a neural network. Therefore, I started out trying to quantify the number of times each character speaks. I wondered what would be this best way to represent these data? Initially, I imagined an array would be easiest to implement and representative. Then, wanting to dive a little deeper, I wanted to analyze the frequencies by a certain feature, ending up choosing houses. This raised the other question of how would I show the disparity(if any) between the number of times spoken between the houses? For this part, I imagined the idea of using averages which is method I ultimately implemented.

To answer these questions, I rely on the features of python, the string library and the pickle package. When trying to represent the frequency of speech by characters, I ran into many bugs and errors trying to implement with array, wasting a lot of time. Remembering learning about dictionaries in reading journals, I realized dictionaries are a much better way to answer my question of what data structure best represents my results. Upon switching to dictionaries, the function and code required became simpler and more intuitive.

I have quite high confidence in my results. I did not use much tools other than pickle and string methods, with the rest being all python. The functions provided individual frequency of speech for each character, which is easily verified and concrete. With that information, the implemented algorithm that calculates the averages is also easily explainable and logical. Nevertheless, limitation resides since the visualization only included the most frequent speakers. Given more time, I would like to provide similar visualization to every character to provide more qualitative or in-context explanation to the disparity in averages between the houses.

Reflection

Overall, the project went well. As with most programming projects, I spent the majority of the time debugging and refactoring. The major switch as mentioned above, is the change of data structure for my input and output from an array to a dictionary. One major regret I have for this project is that I couldn't find an effective way to implement unit testing. Given the large volume of words in the output, it is difficult to perform doctests on my functions. However, I did a search & find for the characters frequency in google chrome and verified with the result of my python program. It is obvious that it is not the best way of testing my functions; but given the matching results as well as examining through all the matches in the text, the accuracy of the results is evident. Given more time, I would love to find a way to implement work arounds to unit test my functions.