Josh Harkness

Assignment01 Deliverable

**Overview**

This deliverable will go through each question in the assignment and discuss how the solutions were obtained. All answers to the first portion of the assignment are in docs/Assignment01.txt and were programmatically generated using HomeworkWriter.py.

**Tokenization**

The first half of the assignment dealt with familiarizing myself with the nltk library and how to obtain tokens within a text document. The code for running this portion of the assignment exists in three locations.

* TextMining/src/main/python/Assignment01/MobyTokens.py – This is the brain of the code. All the logic for the assignment exists here.
* TextMining/src/main/python/Writer/HomeworkWriter.py – HomeworkWriter automated the process of writing the answers to a txt file.
* TextMining/src/main/assignment01Generator.py – The face of the assignment. All execution occurs here, but like a face, it just hides the brain so it looks prettier.

Throughout the assignment, many variables are needed multiple times. The first few calls in the script make these common requirements available for all the questions.

* fileLocation – The location of moby.txt relative to the script file within this repo.
* stringFile – A string representation of the entire file.
* mobyTokens – An unfiltered list of all the tokens (words and punctuation) within moby.txt. The tokens are generated using nltk’s word\_tokenize method.

1. *How many tokens and unique tokens in the text (words and punctuation symbols)?*

The method wordAndPunctuationTokenCounter() uses Python’s native len() function to get the total number of tokens and the nltk library to obtain the unique number of tokens through the FreqDist.

1. *Apply lemmatizations on the verbs in the text, recalculate the number of tokens and unique tokens.*

In getVerbLemmatizations, the nltk library is used again but this time it is the WordNetLemmatizer. To determine each token’s part of speech, the nltk’s pos\_tag function was used on the token list. To count only the verbs, a list of all known verb types in nltk was used as a filter. The WordNetLemmatizer was used on any tokens that passed the filter and were added to their own list. The resulting list after the filter was then passed to the same method used to count tokens in question one.

1. *What percentage of tokens is ' HISTORY' or 'history'?*

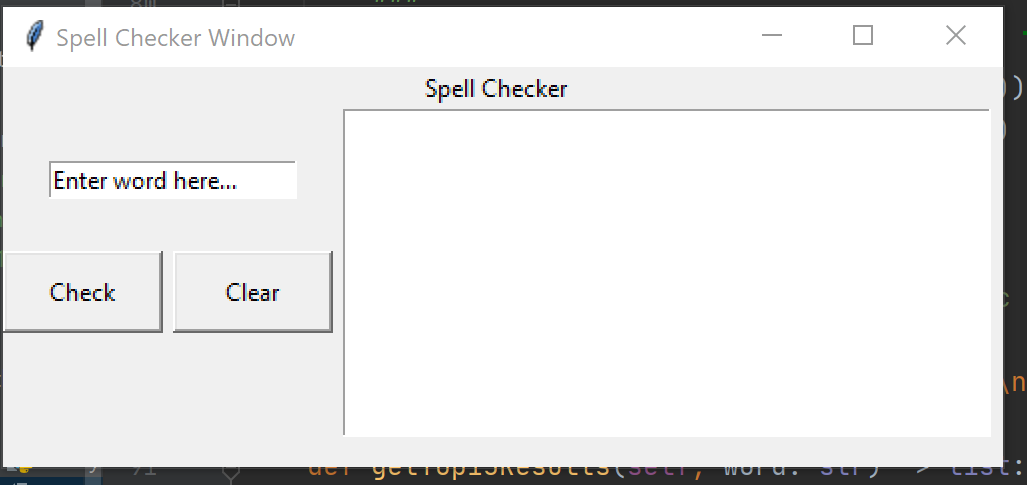
A list containing the two strings above was used as an identifier. FreqDist was used again to get the total number of occurrences of each word in the tokens. The desired words list (‘HISTORY’, ‘history’) was then iterated through, where each term in the list had its frequency percentage obtained with the built in freq() method. The percentages were then added together to receive the total percentage.

1. *What are the 10 most frequently occurring (unique) tokens in the text? What is their frequency?*

Once again, FreqDist was used to calculate the frequencies of each token in moby.txt. The list returned by FreqDist used Python’s native most\_common method to return the most frequent tokens along with their count.

**Spell Checker**

I had a little extra fun with this task and created a GUI for ease of use. Only one file was used for this this task, TextMining/src/main/python/Assignment01/SpellChecker.py, but it is still opened using the script. It should be self-explanatory on how to use.



Once the GUI opens, just type in the word you want to obtain close words from. The top 15 results will be printed in the Text on the right. Tkinter was used to generate the GUI, and some threading was done to ensure pressing the check button did not freeze the GUI. A little more work could be done to disable the buttons while the background function runs, but that is out of scope for this assignment.

All the logic occurs after pressing the “Check” button. Once pressed, a thread is created for the “spell checking” logic. The string entered is first stripped to ensure there are no whitespace characters. The string is then checked to ensure only alphabetic characters exist. If those two conditions were true, the closest 15 words are returned. The task to determine the top 15 seems cumbersome, but it works.

Every word in the nltk dictionary has its edit distance calculated against the entered string. It may have been better to only iterate through words within plus or minus 3 letters, but for ease of the assignment, every word is checked. The resulting dictionary, where key equals the word and the value is the edit distance, is then sorted to help with the following iteration.

The last step iterates through the first 15 terms in the sorted dictionary and returns each word to be printed in the Text.