Andrew Pickner
11/14/18
CSCI 2270
Sharon Gupta, Smith Gopal

Final Project Proposal

Although this may be a little bit out of my skill-level, I think it would be fun to write a program similar to Alex Curtis' data structures final project. It would take text in from a specific author, and then output a piece of text created by the computer that mimics the original author. The motivation behind the project is that I'm fed-up with our current President's twitter habits and I think it would be funny to write a program that could take in some of his Twitter data and output a tweet that resembles one of his in some way. I have some ideas on how I will do this, but those are for later on after the foundation is laid.

My program will begin by taking in two command-line arguments: 1. A file containing cleaned twitter data, and 2. The file's given author. Next, my program will build a Graph from the given text. It will do this by adding a Word-Vertex for each word, and adding a singly-directed weighted Edge from one word to the word that immediately follows that word and set the weight equal to 1. If the edge is already created for two given words, then the weight is incremented by 1. I will choose a word to begin the computer-generated string and then from there, use probabilities to determine which words will come next. More specifically, from this Graph, I will hopefully be able to read up on Markov chains and learn how to use them in the context of this problem. I will add the words to a Queue and then concatenate the final string together. From there I will output the message with an indication of who the "fake" author is.

I will be using Graphs to model an author's "speech-pattern" and use probabilities computed with the Graphs Edges. I can use a Queue for building my final text one word at a time. I'm sure I will be using more data structures once I begin determining how to handle the probabilities created. Furthermore, I may need to implement a HashTable of sorts that maps words to their associated Vertices in the graph so I can look up a node quickly for computing some of these probabilities.

The Graph is perfect for this type of problem because it is very flexible and the Edges allow for a good way to traverse through the words in a logical way. I assume that my program will implement a BFS of sorts that will iterate through the words in the Graph by choosing the next Word-Vertex through probabilities. Sometimes, I may want a specific Vertex in the Graph quickly so I may think about using a hash-table to store pointers of the Vertices. This would allow for constant look-up for a specified Vertex which could come in handy when I have chosen an initial word. Lastly, I may want to build the string by returning a single word from a function. In any case, if I choose to build the final string incrementally, I will use a Queue to store the individual words.

Although my application is mostly just to create some laughter for at least one other person than myself, I will actually really enjoy trying to make this thing as accurate as possible. I had to implement a simple text classifier using the Naive-Bayes Theorem in my intro Computer Science class, and it was fun trying to tweak my code bringing up the accuracy little by little. I'm sure my program could be re-engineered to take in a bulk of text and an author, and then a smaller amount of text, and then the output would be whether or not the smaller text is the given author or not.

I expect that the Markov chain part of the program will be difficult to get correct in forming messages that sound like the given author. It will also be a challenge to create custom messages that haven't been created by the author word-for-word. The Edges might be difficult initially, but once Graph is complete, the difficult time comes from the actual output formulation.