Project Overview

In this project, the code that I wrote enables a few powerful computational tools to analyze text from Project Gutenberg. What I set out to learn was whether - and if so, to what extent are - the products of generative text algorithms different from each other when trained on different texts. Text acquisition from Project Gutenberg has been made extremely simple: my code allows for the user to simply type in the name of a book and its author, and the book is automatically downloaded and processed for computation. Instead of just comparing the similarity of raw text files from Project Gutenberg, the code generates markov chains from these texts, and is capable of comparing any number of markov chains from the same text as well as between two different texts. The tools used for comparing the similarity are using the cosine difference between texts which is calculated through the TF-IDF values for words. In the code that I wrote, no APIs were used for any of this calculation, so the equations can be fine tuned for the weighting desired for the analysis. Furthermore, these results are easily visualized using a scatter plot.

Implementation

1. Setup
The first step of the program is to ensure that the necessary index of Project Gutenberg is present and parsed into a Python dictionary; if this has not happened yet, the code automatically takes care of this. Following this, the code gets various inputs from the user that instruct how it will carry out computation - e.g. whether the program utilize the book names that are hard-coded or prompt the user to input the name(s) of the books. Following this, the code will either download the necessary information from Project Gutenberg, parse it, and process it (into dictionaries for term frequencies and dictionaries used for markov chains), or load this information if it has already been stored (and pickled) on the computer's disk.
2. Markov Chain Generation
After all of the necessary "helper" data has been created for each book, the code immediately generates a few markov chains to display in the terminal. These are short in length and are meant to demonstrate each of the different types of markov chains used (which are described more in depth below). Then, depending on the parameters for the number and length of markov chains, the code generates more markov chains and passes them into the make_similarity_matrix function.
3. Similarity Matrix
The similarity matrix' values are calculated as the cosine similarity between each of the respective markov chains. Prior to this calculation, the vectors of words must be turned into a vector of floating point numbers. These floating point numbers are the TF-IDF values, which are calculated in two parts: the (augmented) term frequency multiplied by the inverse document frequency. Here, I had to make a design decision for which formula for TF-IDF I should use; of the three different TF-IDF weighting schemes (as described on Wikipeia), I chose to implement the second one because it did not require any context about all of the texts in the corpus to

calculate the term frequency, and when calculating the inverse term frequency, having a word not be in a document wouldn't result in the computer trying to take the log of 0. Both of these were important because they allowed me to more easily calculate the (augmented) term frequencies, as well as allowing me to calculate the inverse document frequency for any word in the context of the entire corpus.

For each pair of markov chain vectors, the a simple function is used to calculate the cosine similarity between the vectors, which is filled in as a floating point in the similarity matrix. After all of the values in the matrix have been calculated, the similarity matrix is rearranged (using the example code from the MP3 website) into a matrix that can be plotted as a 2D scatter plot. The points on the scatter plot are color coded depending on what they represent.

Results

The markov chain generators use two different methods of generating the chains: the first method (that I call the random method) chooses the next word based on the raw frequency of the word, whereas the second method (that I call the assisted method) chooses the next word based on the augmented term frequency of the word. I also used a third method that I call the control method, but this method just snips a bit of text out from the book to compare to the other methods. Here are a couple example outputs:

- Random markov chain for Watersprings, by Arthur Christopher Benson:
    - a little commendation he rose and said jack just as if you said that a wife a little silence but you were within--just as before seen a lot to be
- Assisted markov chain for Watersprings, by Arthur Christopher Benson:
    - undergraduate smiled and convenient to provide for marriage but you hated it no sign not affect or been attracted elsewhere how romantic when am most need to pieces and kindness
- Random markov chain for Frankenstein, by Mary Wollstonecraft (Godwin) Shelley:
    - causes of a transport of sensations it an early years before the various feelings he shall not seen for agitation of which i could not sink from beautiful in the

Empirically, one could look at these markov chains and identify that yes, there is indeed a difference between the markov chains of Mary Shelley and Arthur Benson, but is there a quantifiable way to show this difference? By increasing the length of the markov chain and calculating 2 chains for each of the described markov chain methods, we can produce the graph seen in figure 1.

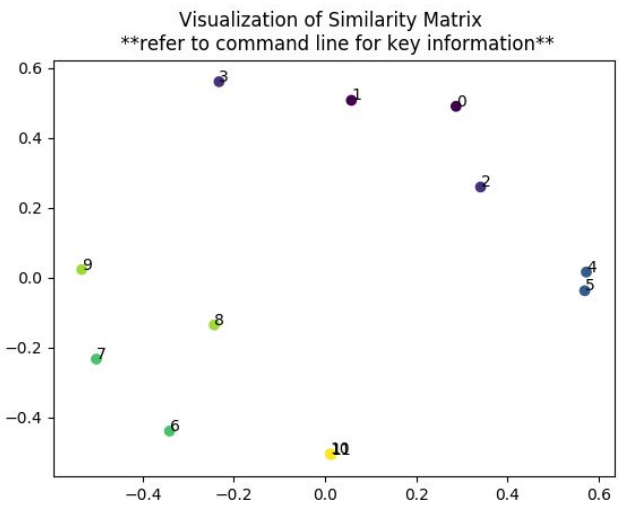

Fig. 1: A plot of the similarity between markov chains generated for Frankenstein (blue/purple) and Watersprings (yellow/green) with a markov chain length of 30,000 words. Axis units are dimensionless. Points labeled 0-1 and 6-7 represent the control markov method, points labeled 2-3 and 8-9 represent the random markov method, and points 4-5 and 10-11 represent the assisted markov method.

Indeed, there is a significant difference between the markov chains generated between the two texts. Note that while these markov chains are unique to each running of the program, the pattern of separation that they produce is the same for dissimilar source texts. It is important to note that large markov chains (in this case, 30,000 words in length) are required to show this trend; smaller markov chains (see fig. 2) do not show this trend:

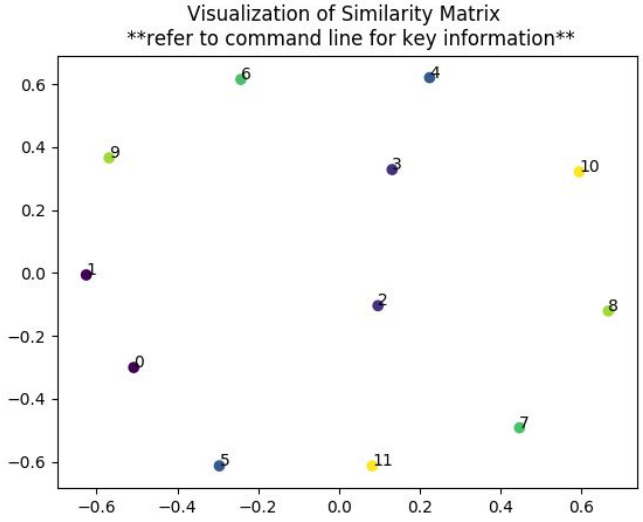

Fig. 1: A plot of the similarity between markov chains generated for Frankenstein (blue/purple) and Watersprings (yellow/green) with a markov chain length of 100 words. Axis units are dimensionless. Points labeled 0-1 and 6-7 represent the control markov method, points labeled 2-3 and 8-9 represent the random markov method, and points 4-5 and 10-11 represent the assisted markov method.

Alignment

There is a strong alignment between what I set out to explore and the data and tools that I used. Because I had set out to explore the distinction between generative text algorithms trained on different books, Project Gutenberg was the perfect place to quickly, easily, and legally acquire text files for books. The tools that I used for comparing similarity, too, aligned well because a similarity matrix populated with cosine similarity values calculated through tf-idf values is a very standard way to compare the similarity of texts, and indeed produced easily interpretable results.
What sparked my exploration was a desire to understand the differences between predictive text models trained on different sources, and although I had originally set out to explore this topic using machine learning models, markov chains served the purpose of answering my questions just as well.

What I had imagined for my results was a scatterplot of similarity that would show a distinct cluster for each type of markov chain, as well as a large difference between markov chains trained on difference source texts. Indeed, this latter result was observed, but the former was not. Only markov chains from the assisted method clustered together; the other two (the random and control methods) seemed to bear no resemblance of being closely related. While I did not expect this, it is a reproducible result and, when consideration is taken that a markov chain predicated on the raw frequency of words in a text should have the same amount of variability as text randomly selected from the book, is a result that makes sense. It also makes sense that less variability should arise in the assisted method because words are chosen based on the augmented term frequency, which de-emphasizes common words like “the” and emphasizes the less common words. This makes it easier for the assisted method to generate the less-common words which, by definition, have a smaller number of possible following words (hence the lower variability).

The computational tools that I employed served my analysis well: they produce reproducible results that align with my expectations; the tools that I created for accessing text from Project Gutenberg also serve my purposes well because they store data to reduce the amount of computation when possible, and they make it incredibly easy to acquire any text from Project Gutenberg. While the tools themselves aren’t limited to comparing two texts, the scaffolding of the code around them is, but I had never set out with the intention of comparing more than two texts at once. In addition to the reproducibility of expected results, I am confident in the accuracy of the tools that I employed because I either used unit testing or a separate and more manual method of testing a tool before integrating it into my project.

Reflection

From a process point of view, I feel like my research and implementation of computational tools used for text similarity went quite well. I went from having no understanding of these tools to

being able to successfully implement them without the use of any API (except for visualization). However, my process for developing a method for acquiring text could have gone better: it wasn't until I had built an entire method for scraping text from Project Gutenberg that I discovered that Project Gutenberg does not allow roboting of any kind and that I should have set up my code to scrape from a mirror of Project Gutenberg. Not only is this more ethical and safer from the standpoint of not having to worry about a blocked IP, this method proved to be an order of magnitude faster and more reliable, and is something that I should have done from the very beginning. Although I had set out to explore machine learning models, I feel like I made the right decision in choosing to scale my project back into a more in-scope project by just using markov analysis. Although not every function uses unit tests, important functions in the calculation of the similarity matrix do have unit tests, and all features of the code have been thoroughly tested.

A lot of what I learned from this project was centered around how to compute with text in a quantitative way, which, alongside my new skills of scraping text from the internet, is a skill that I can carry forward with me for future data oriented projects.