Project Gutenberg and Zipf's Law

Paige Pfenninger

February 25, 2016

1 Project Overview

I used Project Gutenberg, and downloaded four books: *Frankenstein*, *The Odyssey*, *The Wizard of Oz*, and *Peter Pan*. I then compared the word frequency distribution in those books to the word frequency distribution predicted by Zipf's Law. I hoped to learn how Zipf's law applies to books written during various time periods, and how accurate it is for various sizes of texts.

2 Implementation

The program has three major components. The first component takes a file name of a Project Gutenberg book and converts it to a usable string, the second component makes a histogram of all of the words in the string from the first portion, and the last component takes the histogram and sorts it from most frequent to least frequent and returns a list. The file itself isn't useful because it contains a lot of other things besides works like punctuation, numbers, and line breaks which is why it is necessary to convert the file into a usable string. The dictionary data structure is especially useful for creating a histogram because the dictionary is able to rapidly find each word and increment the word count. The dictionary is then transformed into a list of tuples, because the list can easily be sorted whereas the dictionary cannot be easily sorted.

In an ideal world, the program would have been able to display graphs of the actual word frequency and the word frequency predicted by Zipf's Law. Unfortunately, it seemed more time efficient to do the graphing in Matlab instead of in Python. The python program produces a cvs file of sorted numbers where the n^{th} number corresponds to the frequency of the n^{th} most frequent word. The Matlab code then takes the cvs file and plots the first 1000 data points along with the data poins predicted by Zipf's Law. CVS files tend to be easier to use in Matlab, which is why the python program outputs a CVS file instead of another type such as an excel file.

3 Results

Zipf's law states that the frequency of a word is almost inversely proportional to its rank in a frequency table. This can be shown by the equation

$$P_n \propto \frac{1}{n^a} \tag{1}$$

where a is close to 1. For most languages this transforms into the equation

$$P_n = \frac{1}{r\ ln(1.78 * R)} \tag{2}$$

where R is the number of different words.

Zipf's Law is generally a fairly goof predictor for large language samples. The plots of the word frequencies of *Frankenstein*, *The Odyssey*, *The Wizard of Oz*, and *Peter Pan*, are all very close to the word frequencies predicted by Zipf's Law.

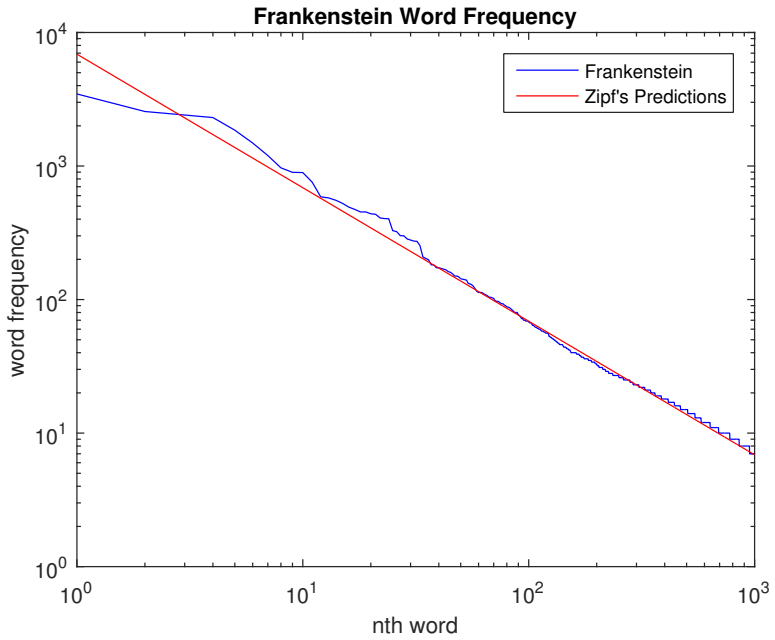

Figure 1: Comparison of Frankenstein and Zipf's Law

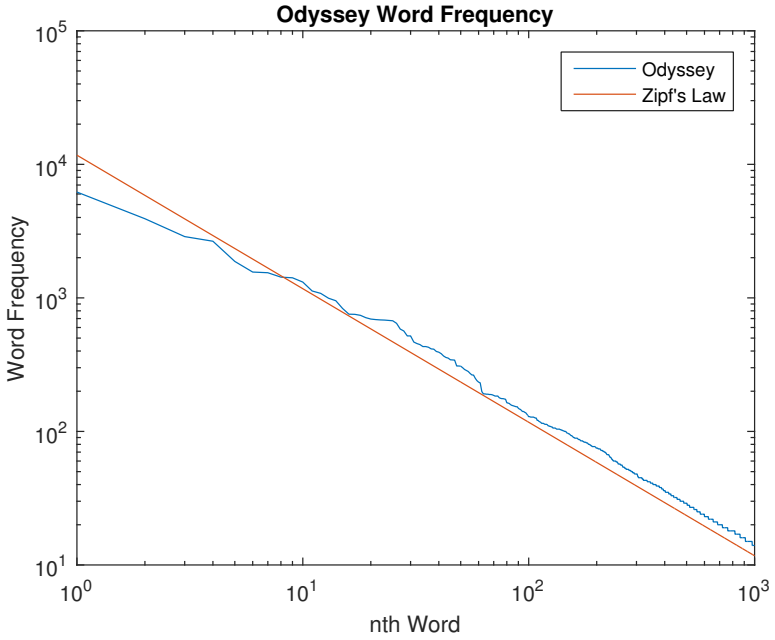

Figure 2: Comparison of The Odyssey and Zipf's Law

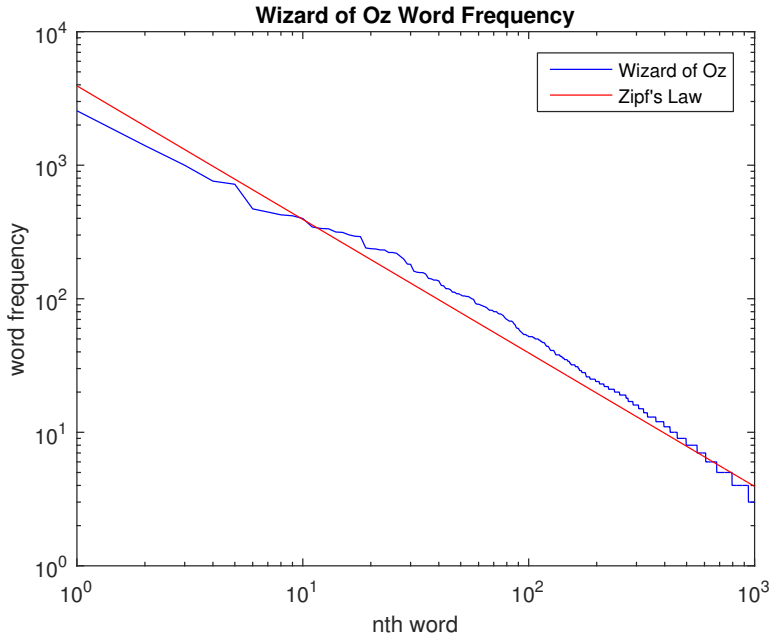

Figure 3: Comparison of The Wizard of Oz and Zipf's Law

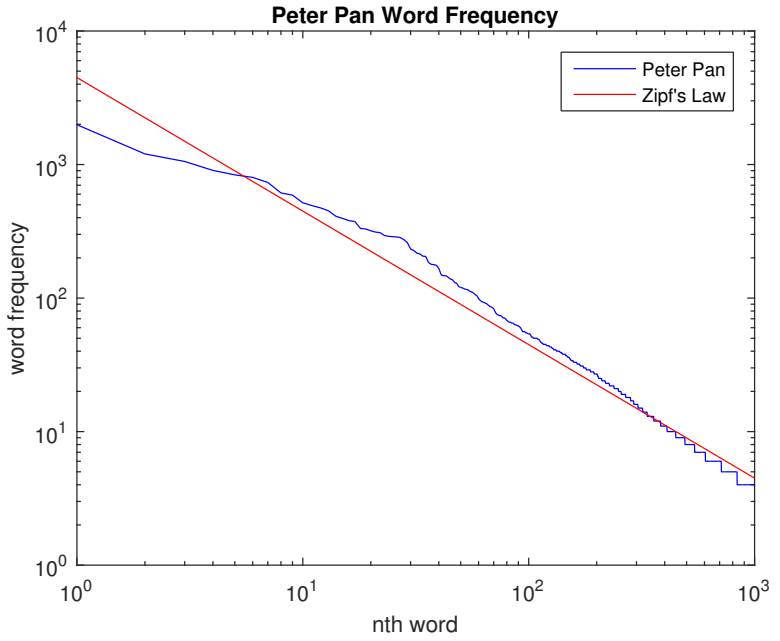

Figure 4: Comparison of Peter Pan and Zipf's Law

The word frequency of each book is very similar to the word frequency predicted by Zipf's Law. It is interesting Zipf's Law works for a book as old as the Odyssey (granted I used the English version, not the original Greek version) as it does for more modern books. For all of the books, Zipf's Law tends to be a better predictor from the 100^{th} to the 1000^{th} most frequent words than it is for the words that appear more frequently. The most common word in every book was 'the',

but the top 25 most common words varied a bit from book to book. There was even a name in the most common 25 words of the Odyssey. Names did not appear in the most common 25 words of the other three books.

4 Reflection

The python program itself was fairly easy to write, and saving the books from Project Gutenberg was also simple. I was successfully able to write unit tests for my three functions. One of the bigger problems with my function that turns the book file into a string is that if only works for Project Gutenberg books because the function deletes the preface and licsene at the end of the book and uses markers that are specific to Project Gutenberg books. I wish that I had gotten more data from Project Gutenberg, and had been able to find a way to combine data from several books. I don't think this project is as interesting as it could have been. It would have been more interesting if I had been able to find books in different languages and tested to see if Zipf's law applied as well to other languages as it does to the English books. If I had had more time, I would have liked to figure our how to make nice graphs in Python rather than using Matlab, but exporting the data from my Python program into Matlab seemed like the best way to make graphs at the time.