Zipf's Law in Classic Literature

Jaden Noronha

January 2022

1 Introduction

In 1949, American Linguist George Kingsley Zipf proposed the principle of least effort, also known as Zipf's Law. [1] This law stated that **the frequency of a given word is proportional to the inverse of its ranking in a corpus.** [2] That is,

$$f(w) \propto \frac{1}{r^a} \tag{1}$$

where *f(w)* represents the frequency of a word w, r represents the word's ranking, and a is a value close to 1.

The law roughly described the distribution of words in human languages: there are a small number of words used very frequently, a large number of moderately used words, and several low frequency words. [1] This means that the frequency of the second most common word in a corpus of text is expected to have roughly 1/2 the frequency of the most common word, the third most common word has 1/3 the frequency of the first, and so forth. [2]

Zipf's law has been applied to numerous other languages and disciplines, including psychology, sociology and economics. [1] To explore and deepen my understanding of this law, I decided to apply Zipf's law to various works of classic literature, as found on Project Gutenberg, an online catalogue of classic written works.

2 Writing the Program

To analyze Ziph's law, I wrote a program in Python 3 which opens all of the text files in the same directory as the program file, prints the total word count and top 10 most common words (with their respective frequencies), and produces a graphical visualization of the law using the matplotlib library.

2.1 Importing Libraries and Defining Global Values

First, I imported the required libraries for the program. The first module imported was **string**, which provided a list of punctuation that was used to remove punctuation from the consumed text. Next, I imported the **os** module, which allowed me to consume the text files. I also defined a variable *location* which contained the directory of the program file, as the program consumes all of the text files in its directory. Finally, I imported **matplotlib.pyplot** (as **plt**), which provided me a rich array of graphing tools.

I then defined three important Global values:

- *words* is a dictionary where the keys represent words found in the corpus and the values represent their word counts.
- *wordcounts* is a list containing tuples which represent (word, word count)
- *wordcount* is a Natural number representing the total number of words in the corpus

2.2 Function 1: consume_files()

The first function was called *consume_files()*, which acts as the "main" function to the program. This function iterates through each of the files in the same directory as the program file. If the file is a text file (ends in *.txt*), the file is opened, and the function *calculate_wordcounts(file)* is run, consuming the file. After all of the files in the directory are iterated through, the function *print_final_values()* is run.

2.3 Function 2: calculate_wordcounts(file)

The next function was called *calculate_wordcounts(file)*, which consumes a file from *consume_files()*. This function counts the words in the text files, and updates the global values *words* and *wordcount*. This happens by counting each line in file, stripping the whitespace and punctuation, and converting the line into a list of lowercase words. Then, each word in the line was iterated through, stripping any characters that are not letters, and then adding the word to the *words* dictionary and updating the *wordcount*.

2.4 Function 3: plot_zipf()

plot_zipf() is the function that plots the data collected from the global list *wordcounts*, which contains tuples representing the words and their corresponding word counts (sorted in decreasing order).

According to Zipf's law, when the ranking of a word (based on its frequency relative to the other words in the corpus) is graphed against its frequency (word count in the corpus) on a LogLog scale, The slope of the frequencies should be close to -1 [3].

To graph this relation, I created 3 sorted lists: one list contains the ranks of the words in ascending order (based on *wordcounts*), the next contained the frequencies of the words in descending order (also based on *wordcounts*), and the last contained the expected frequencies of the words in descending order. This was calculated by taking the frequency of the first word in *wordcounts*, and then dividing it by the corresponding rank (see equation (1)). All of this data was then graphed using the **matplotlib** library on the same graph.

2.5 Function 4: print_final_values()

This function produces the final values based off the data obtained from the text files. First, the key-value pairs from *words* were added to *wordcounts* as tuples, and then *wordcounts* was sorted in decreasing order of word frequency. Then, the total word count (from *wordcount*) and the 10 most frequent words (from *wordcounts*) were printed to the console. Finally, *plot_zipf()* was called to produce the graph.

3 Corpus Selection

For my corpus, I chose the 10 most popular E-books found on Project Gutenberg over the past 30 days (As of January 6th, 2022). The works chosen were:

1. *A Christmas Carol in Prose; Being a Ghost Story of Christmas* by Charles Dickens
2. *Pride and Prejudice* by Jane Austen
3. *Frankenstein; Or, The Modern Prometheus* by Mary Wollstonecraft Shelley
4. *The Adventures of Sherlock Holmes* by Arthur Conan Doyle
5. *Alice's Adventures in Wonderland* by Lewis Carroll
6. *The Scarlet Letter* by Nathaniel Hawthorne
7. *Moby Dick; Or, The Whale* by Herman Melville

8. *A Tale of Two Cities* by Charles Dickens
9. *The Brothers Karamazov* by Fyodor Dostoyevsky
10. *The Picture of Dorian Gray* by Oscar Wilde

All of the E-Books were downloaded as text files (.txt) and placed in the same directory as the program file so that they could be read by the program. At the beginning and end of each E-Book, there is text describing the book's information and licensing information. ***To ensure that the data collected only represents words found in the written texts, these sections were manually deleted before the program was run.***

```
The Project Gutenberg eBook of Alice's Adventures in Wonderland, by Lewis Carroll

This eBook is for the use of anyone anywhere in the United States and
most other parts of the world at no cost and with almost no restrictions
whatsoever. You may copy it, give it away or re-use it under the terms
of the Project Gutenberg License included with this eBook or online at
www.gutenberg.org. If you are not located in the United States, you
will have to check the laws of the country where you are located before
using this eBook.

Title: Alice's Adventures in Wonderland

Author: Lewis Carroll

Release Date: January, 1991 [eBook #11]
[Most recently updated: October 12, 2020]

Language: English

Character set encoding: UTF-8

Produced by: Arthur DiBianca and David Widger

*** START OF THE PROJECT GUTENBERG EBOOK ALICE'S ADVENTURES IN WONDERLAND ***
```

Figure 1: An example of a section that was manually removed. Everything before the *** START OF THE PROJECT GUTENBERG EBOOK ... *** line was removed.

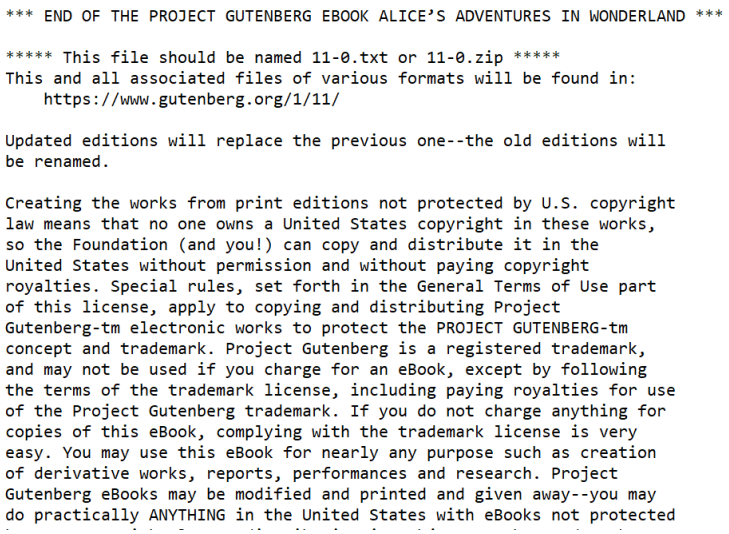

Figure 2: Another example of a section that was manually removed. Everything after the *** END OF THE PROJECT GUTENBERG EBOOK ... *** line was removed.

This was done for scalability of the program, so that it could be applied to all kinds of text files, not just E-books from Project Gutenberg. While this method does remove some error, other text added within the texts (such as chapter headings and transcript information) was not removed, however their inclusion should not influence that data by a significant amount.

All of the original E-Books with all of the the original formatting and liscencing information can be found in the folder *RAW FILES*.

4 Results

```
Total word count: 1217980

Total unique words: 37427

The 10 most common words are:
the 63995
and 38986
of 33373
to 31579
a 25362
i 21511
in 20501
that 17621
he 17512
it 16605
```

Figure 3: The raw output into the python console.

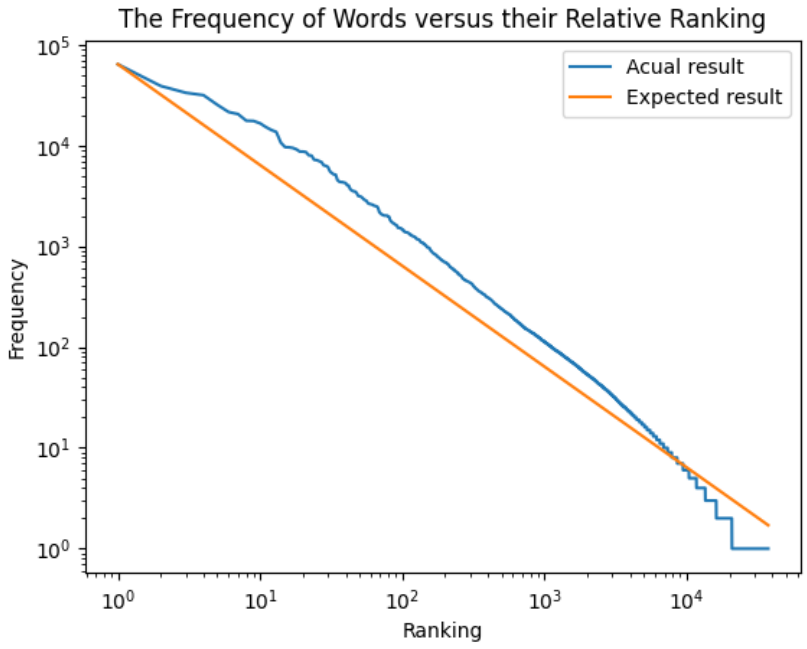

Figure 4: The graphical representation of the relative word count ranking of the words against their frequencies on a LogLog scale.

After running our program with our modified files, we have gained several insights about our selected corpus. In the 10 texts, there were a total of 1217980 words, with 37427 unique words."The" was the most common word with 63995 occurrences, followed by and, of, to, a, I, in, that, he and it (Figure 3). Looking at Figure 4, where the orange line represents our expected result and the blue line represents our data, our data generally follows the trend, however, the frequencies were mostly higher than expected. Towards the end of the ranking, the frequencies were significantly lower than expected.

The results of this analysis are consistent with our current knowledge, As the general shape of our graph was similar to the shape of the expected result. The sample size was relatively small, with only 10 texts and 1.2 million words in total. As the sample size increases, the data will become increasingly consistent with Zipf's law. The large diversions from the expected result for words with very low frequencies can also be explained. Referring back to equation (1),

$$f(w) \propto \frac{1}{r^a}$$

This formula states that the relative frequency of a word in a corpus should be proportional to roughly one over its rank. Since our sample size is small, let us observe the word that was ranked last, or ranked 37427th. From (1), our expected result for this word is (assuming a = 1 for simplicity):

$$f(w_{37427}) \propto \frac{1}{37427^1}$$

Multiplying by the 63995, the frequency of the most common word "the":

$$f(w_{37427}) = f(w_1) * \frac{1}{37427} = \frac{63995}{37427} \approx 1.709$$

Referring back to the graph in Figure 4, we can see that the lowest ranked value has a frequency of 10^0, which is equal to 1. This is a significant deviation from the expected frequency. Calculating the percentage error:

$$Error \approx |\frac{1 - 1.709}{1.709}| * 100\% \approx 70.9\%$$

This value can be reduced with a larger corpus with more diverse texts and a larger word count and word variety.

5 Testing Scalability

The core analysis only analyzed 10 files with 1.2 million combined words. However, the program can also analyze much larger data sets. In this section, we will preform the same analysis with the same texts, but on a much larger scale.

5.1 Modifying Corpus

Each EBook tested in the core analysis was duplicated until there were 10 copies of each EBook. This means that the values found in Figure 3 should increase by a factor of 10. Since each text is multiplied 10 times, the frequency axis on the resulting graph should start at 10^1 instead of 10^0.

5.2 Modifying Code

The initial program file was slightly modified to measure the time taken for the program to run. To measure this, I used the **time** module and calculated the time taken from when the program is first run, to when the window containing the graph opens. This was done by finding the difference between the time when the program was started (using the variable *start_time* in the global variables section) and the time when the graph window opens (using the variable *end_time* in the *plot_zipf()* function), which was done using the *perf_counter()* function from the **time** module.

5.3 Results

I tested the program on a moderately-powered laptop (Ryzen 5 Pro 3500U with 8 GB of RAM) that was plugged in to optimize performance. The test was run 5 times to gain more data points.

```
Total word count: 12179800

Total unique words: 37427

The 10 most common words are:
the 639950
and 389860
of 333730
to 315790
a 253620
i 215110
in 205010
that 176210
he 175120
it 166050
Time taken: 27.640844100000002
```

Figure 5: The raw output into the Python console for one of the scalability tests.

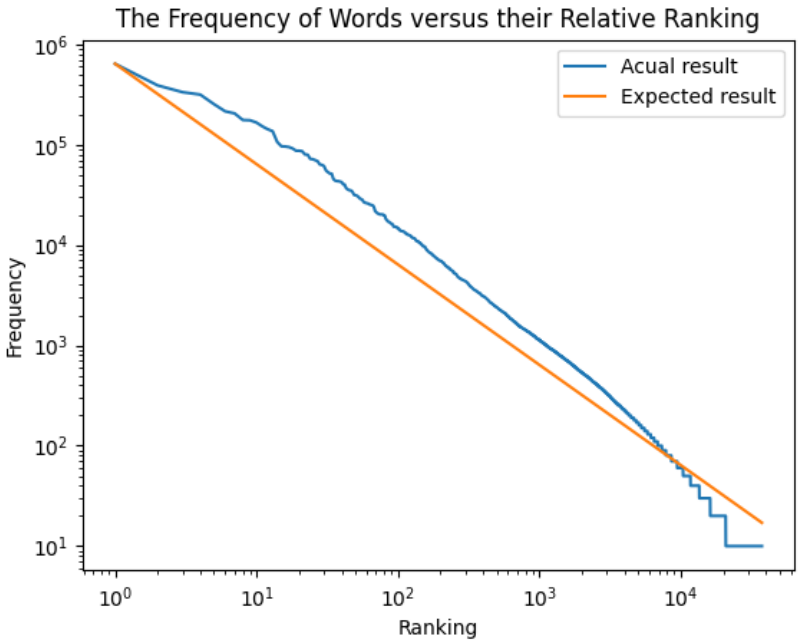

Figure 6: The graph from the scalability tests.

Test number	Time (Seconds)
1	32.0088673
2	27.640844100000002
3	27.234517200000003
4	26.792392500000002
5	27.0549411

Figure 7: A chart with the elapsed times for the tests.

The results from Figures 6 and 7 are expected, as the frequencies have been scaled by a factor of 10, while the ranking of the words was maintained. After five tests, the average time taken to analyze 12179800 words was about 28.1 seconds, or roughly 430 thousand words per second.

6 Extensions and Applications

Zipf's law has many implications regarding how we communicate. There are a few words that are very common, some middling-frequency words, and many words that are barely used. According to Zipf, this allows us to communicate with less effort, as both the listener and speaker frequently use a common set of words [1]. Complex words would make communication difficult for some, since they have not encountered that word often in their communications.

To extend this project, I would like to test a larger corpus, as a larger sample size would yield more accurate results. I would also like to test different types of literature, such as speech transcripts and news articles, as they are written for different audiences for different purposes.

7 References

1. https://www.thoughtco.com/principle-of-least-effort-zipfs-law-1691104
2. https://plus.maths.org/content/os/latestnews/may-aug08/food/index
3. https://www.ccs.neu.edu/home/ekanou/ISU535.09X2/Handouts/Review_Material/zipfslaw.pdf