Report

Data

The file is a **plain text** version of “Bram Stoker’s Dracula”, downloaded for free from **Project Gutenberg**. I found it interesting because of my personal bias and fascination with vampire stories. Beyond that, I noticed that it contained a lot of punctuations like “ and ’ which didn’t match the standard encoding which python recognizes (contained in `string.punctuation`) and some other instances of text like _3 May and —Left which makes it a challenging piece of text for pre-processing.

Methodology

In developing the software I used `python` with `nltk` library for natural language processing as suggested by the professor. The software processes large volumes of textual data and produces and output showcasing each token from the text alongside the number of times it has appeared. To generate figures, I used the `matplotlib` library which visually showcases the distribution of word frequencies in the dataset using a bar graph.

I first **broke the assignment into subtasks** which were **handling command-line arguments, reading the text file, tokenizing the text, preprocessing the text, sorting the words based on their frequency and plotting them**. I started by researching how to pass arguments to a Python script using `sys.argv`, which allowed me to handle optional flags for various preprocessing steps. For converting the text into tokens I tried using `.split()` but googled a bit and found out that `nltk.tokenize()` is a much better way to do so because the text I used was a bit complex and `.split()` couldn’t capture certain words which were combined with numbers and punctuations. I also applied methods like lowercasing and punctuation removal, which didn’t require the use of `nltk` .

The remaining tasks, which were **sorting based on word frequency** and **plotting** them were fairly straightforward. I first counted the text by looping through the tokens, adding them to a dictionary and updating the value if they key already existed. To sort the words according to their count, I just googled the best way to sort dictionary items by their values. For plotting the data I used `matplotlib` library due to my previous experience with it. I plotted the **top 25** tokens with the highest word count and the **bottom 25 tokens** with the lowest word count. I didn’t have to use log-scaling since I had no trouble in viewing the top 25 and bottom 25 tokens, but when I wasn’t limiting myself to that range I indeed had to use log-scaling without which the bar graph wasn’t very clear.

Preprocessing options for users

There are various preprocessing options that a user can use to preprocess the text. To invoke the preprocessing methods, the user must specify the appropriate flags in the terminal, following the Python script and text file names. **The command syntax is as follows:-**

```
python normalize_text.py <text_file> [<options>]
```

Where:

- **<text_file>** is the name of any text file that the user wants to process.

- **<options>**
  - -l (optional) enables the conversion of text to lowercase
  - -lr(optional) enables lemmatization
  - -s(optional) enables stemming
  - -st(optional) enables removal of stopwords
  - -p(optional) enables removal of punctuations

For example, to process a text file with stemming, lemmatization, and punctuation removal, you would run:

```
python normalize_text.py dracula.txt -s -lr -p
```

An **additional option** for preprocessing that I have added beyond the defined requirements would be the **removal of punctuations**. As stated earlier, I noticed that my data contained a lot of punctuation, and before adding this option, the most common word, to my surprise, was a comma. In fact, the top 5 words were mostly punctuations which is not very useful in my opinion.

Furthermore, I added **pos tagging** as a part of my lemmatization process (-lr). Even though pos tagging itself is considered a preprocessing option of its own, I incorporated it as part of my lemmatizer because without appropriate pos tagging it wasn't converting nouns and verbs to their proper base forms, and I wasn't getting any useful results in my data. Rather than counting this as an additional option, I consider this an **augmentation** of the original lemmatizer option.

Sample Output

The output for the input `python3 normalize_text.py dracula-novel.txt -lr -p -st` on my macOS machine is as follows:-

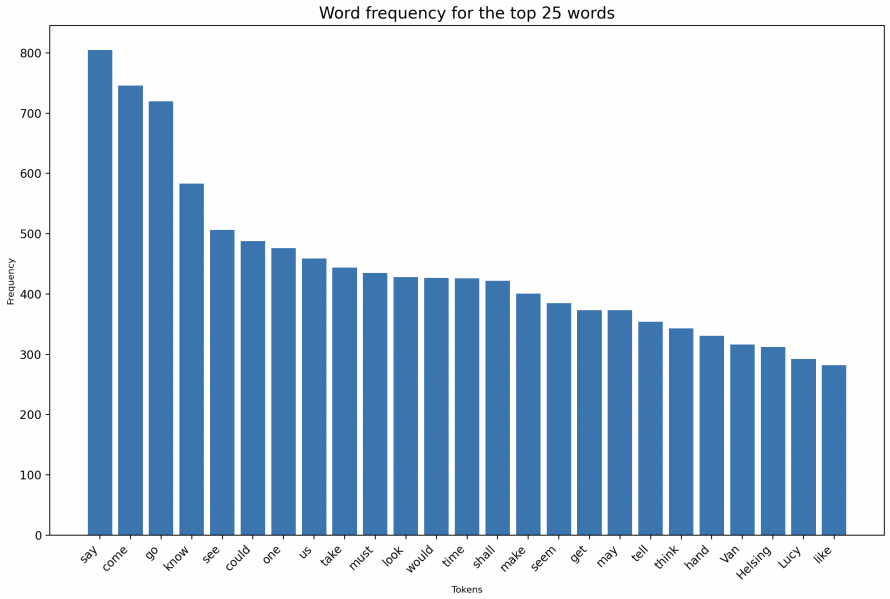
Standard Output(stdout)

```
The top 25 tokens with the highest word count:
             ---------------
say 805
come 746
go 720
know 583
see 506
could 488
one 476
us 459
take 444
must 435
look 428
would 427
```

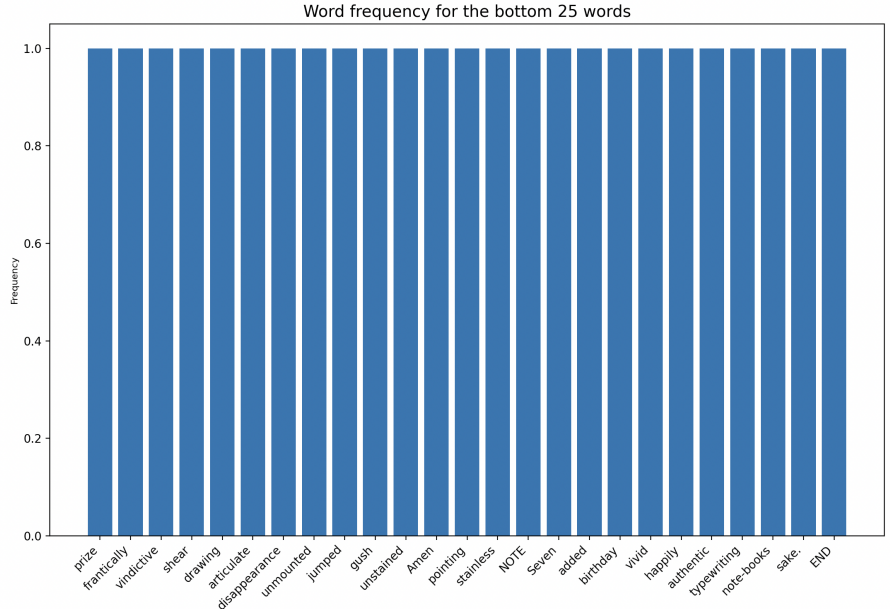
```
time 426
shall 422
make 401
seem 385
get 373
may 373
tell 354
think 343
hand 331
Van 316
Helsing 312
Lucy 292
like 282

The bottom 25 tokens with the lowest word count:
             ---------------
prize 1
frantically 1
vindictive 1
shear 1
drawing 1
articulate 1
disappearance 1
unmounted 1
jumped 1
gush 1
unstained 1
Amen 1
pointing 1
stainless 1
NOTE 1
Seven 1
added 1
birthday 1
vivid 1
happily 1
authentic 1
typewriting 1
note-books 1
sake. 1
END 1
```

Bar Graphs

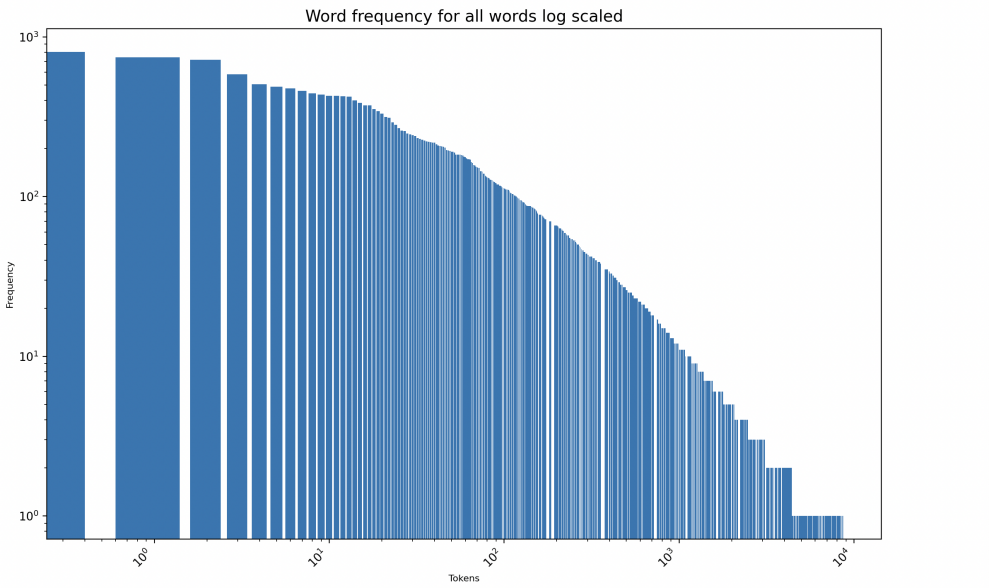
1. Top 25 words



2. Bottom 25 words



3. All words log-scaled



Discussion

Findings

For words at the top of my list, I noticed that if I didn't do any preprocessing, they were **mostly stopwords or punctuations**, with a comma (,) being the most common word occurring 11230 times and the being the second most common word occurring 7298 times. I feel most of these commonly occurring words had some grammatical function and didn't contribute much to the semantic context of the text. After preprocessing by removing stopwords and punctuations, I noticed a significant difference where these filler words were removed. Compared to 11230, my most common word occurred only 805 times. On the other hand, at the bottom of the list, without any preprocessing, I noticed that the bottom words either were capitalized or maybe didn't have much to do with the context of the novel. On adding lowercase preprocessing, I noticed some of the capitalized words that were occurring on the lower end weren't there anymore.

I read the Wikipedia page and felt that there were similarities between the description and my observations. In particular, at the beginning, they were saying that stopwords like "the" were some of the most common words in English text, which was true in my case as well since I noticed words like "the" and "and" were some of the most frequently occurring words in my case. On the lower end, I observed a stair-like pattern, especially in my log-scaled graphs, which also seems to be true.

I can see that the importance of removing stopwords in a corpus is significant. Regular nouns and verbs didn't occur nearly as frequently as them even when plotting them without preprocessing. Even without research, this observation seems intuitive since words like "the" and "and" are used much more than other words, even in my own daily usage of the English language. In regular usage of

English, these words can act as conjunctions and are extremely important, but their usage in text analysis is useless.

Reflections

There were plenty of things I learned from this assignment, the first of them being **report writing** itself due to the fact that I didn't use AI to write my report and wanted to learn how to write a technical report by myself.

The most important thing I learned from this assignment was the usage of the `nltk` library. The most interruptions I had while solving this assignment were due to my lack of knowledge of `nltk`. I had to use Google for the basic syntax of the `nltk` library, for example. I didn't know a stemmer object could be created using `stemmer = nltk.stem.PorterStemmer()` . Learning this library benefits me far beyond just the scope of this course since in some of my other courses or projects, it can be used to model CFGs and draw parse trees for an expression.

Moreover, I learned more about various **preprocessors** and their practical usage. Although I had theoretically learned about these preprocessors in class, learning their practical usage on a large dataset gave me better clarity on how they function. I **researched online** and found that **an article** by Spot Intelligence provided the most comprehensive guide about preprocessors. I learned more preprocessing methods and **used some snippets** from the article that were related to just the basic usage of some processors like **stemmers** and **lemmatizers** using `nltk`. One of the preprocessing methods called **Part-Of-Speech tagging** was mentioned separate from lemmatization, but I incorporated it with my lemmatizer since I found it to be essential before lemmatization. I made this decision due to an **interruption** I had while building this software, which was that the **lemmatizer wasn't functioning as planned** on my dataset. Theoretically I was aware that 'running' should be changed to 'run' and 'was' should be changed to 'be'. Instead, running was unchanged, but 'was' got changed to 'wa'. I **researched online** and found a **conversation post** where a user had the same problem as me, and some had replied to their question by pointing out that it was because the default pos for lemmatizer was a 'noun' and it is programmed to remove the letter 's' from a word. I resorted to **using ChatGPT for inquiring how to determine POS for each token** from the text, and it explained to me about the **POS tagging**, which helped me understand its actual function.

I also learned that some text can be encoded differently, which Python libraries might not recognize. In my text file, there were a lot of commas which were different than the standard encoding. I was already aware of the fact that punctuations were already a part of the `string` library in Python and used it to filter punctuations in my text only to find out that it wasn't removing most of them. Luckily, they were one of the most frequently occurring tokens, so I didn't have to manually scrape the data to locate them. I extended `string.punctuation` with the ones I found in the text, and it solved the problem.

Additionally, I learned the basic functionality of the `sys` library, which enabled me to read the inputs of the text filename and flags for the preprocessing methods.