LNMIIT

NLP Project Round - 1

TEAM NAME: Mega Expedition

Team Members:

NAME	Roll No
Mohmmed Abul Haq	21UCS129
Mohmmed Atif Khan	21UCS130
Devavarapu Sai Ruthvik	21UCS174
Bishnu Kumar Shil	21UCS249

BOOK NAME: Gulliver’s Travels

GitHub link: https://github.com/Md-Atif-Khan/NLP-Project-.git

Abstract:

This project involves an analysis and processing of text covering important aspects. It starts by extracting text from a PDF document using the pyPDF2 library. Then follows a series of steps. These include preparing the text breaking it into tokens, removing words thoroughly studying the distribution of tokens and creating a word cloud to visualize key terms and performing Part of Speech (PoS) tagging using a Hidden Markov Model trained on the Brown Corpus.

Furthermore the program creates a bi-gram table of probabilities for the largest chapter in the book while also including stop words. This table is then used in a "Shannon game" to fill-in blank parts within a chapter enabling a thorough assessment of how accurately the completed text aligns with the original content. This project serves as a showcase of techniques for processing natural language and analyzing text, offering valuable insights, into the internal mechanisms behind the given text.

Introduction:

The code project is an exploration of analyzing and processing text. It starts by extracting written content, from a PDF document, which forms the basis, for operations. The process of going through this code can be divided into the following stages:

Text Preprocessing and Frequency Distribution:

At first the code processes the extracted text by performing steps to prepare it for analysis. Elimination of patterns and elements those are not essential, to the main story, such as copyright notices and chapter headings. Next the text is organized into units of meaning(tokens). Unnecessary words (stopwords) are removed to focus on relevant words for analysis. The frequency distribution of these tokens is then examined to identify the used terms in the text. A word cloud is created to represent these mentioned terms giving a clear overview of the most important concepts.

Part-of-Speech (PoS) Tagging:

Next POS tagging is done by the help of hmm model. Hidden Markov Model is then trained on the Brown Corpus to tag words with their respective grammatical categories. At last a frequency graph is plotted to visualize the result of POS tagging.

Bi-gram Probability Table:

A bi-gram consists of pairs of consecutive words and the Bi-gram Probability Table approximates the probability of a word given all the previous words by using only the conditional probability of one preceding word. Largest chapter from is novel is selected for performing the bigram modelling and stopwords are included as well. This is a critical component for subsequent steps in the project.

The Shannon Game:

Shannon game involves selecting a chapter, excluding the one used for creating the bi gram probability table and filling in missing words or phrases, within the text. To predict and insert words into these blanks we utilize the generated bi gram probability table. Afterwards we evaluate the accuracy of these predictions by comparing the filled text with the content. This step not only demonstrates how we can practically apply the probability table but also serves as an exercise, in assessing how well our language model performs.

1. TEXT PREPROCESSING

Step-1:

The code assigns the content of a variable “pdf_text”to another variable “copy_pdf_text”. This essentially creates a copy of the text content, allowing forfurther manipulation or analysis while preserving the original text in “pdf_text”.

Step-2:

Now we take a text and removes specific phrases and patterns like "Free eBooks at Planet eBook," non-standard characters, instances of "Gullivers Travels" with optional numbers, ".com" with spaces, "RICHARD SYMPSON," and text sections marked as "Chapter" followed by Roman numerals. These changes clean and refine the text. Following this the text which didn’t have space between special character and text was also given a blank space in between so as to help in getting accurate bigram probability distribution table. All the above steps were done after considering various results.

Step-3:

The code manipulates the “copy_pdf_text” variable to refine its structureand readability. Initially, it discards the first six lines of the text, effectively removing unwanted content from the beginning. Next, it performs a series of substitutions. Line breaks (newlines) that do not follow a hyphen are removed, creating more coherent text flow. The code also eliminates hyphens within the text. It then replaces line breaks with spaces, making the text more compact.

Step-4:

Additionally, it ensures there is a space after periods, specifically when they are followed by a word character, which is a common typographic convention. Lastly, it surrounds characters enclosed in parentheses with spaces to improve text clarity. These alterations contribute to a cleaner and more structured version of the `copy_pdf_text`.

2. FREQUENCY DISTRIBUTION

Step-1:

We tokenized the cleaned and filtered text, which means we split it into individual words or tokens to make it easier for analysis. We then went on to remove common stopwords, which are words that typically don't carry significant meaning in the analysis.

Step-2:

After this step, we created a frequency distribution of the remaining tokens, essentially counting how often each word occurs in the text. To present this data more clearly, we converted it into a DataFrame with two columns: "Token" and "Frequency." This DataFrame was then sorted by word frequency in descending order. We also dropped the three most common words (particularly punctuations: “,”, “.”, “;”) since they were potential outliers in the dataset.

Step-3:

Now created a set of visualizations to better understand our text analysis results. First, we designed a bar plot to showcase the 35 most common words in the text.

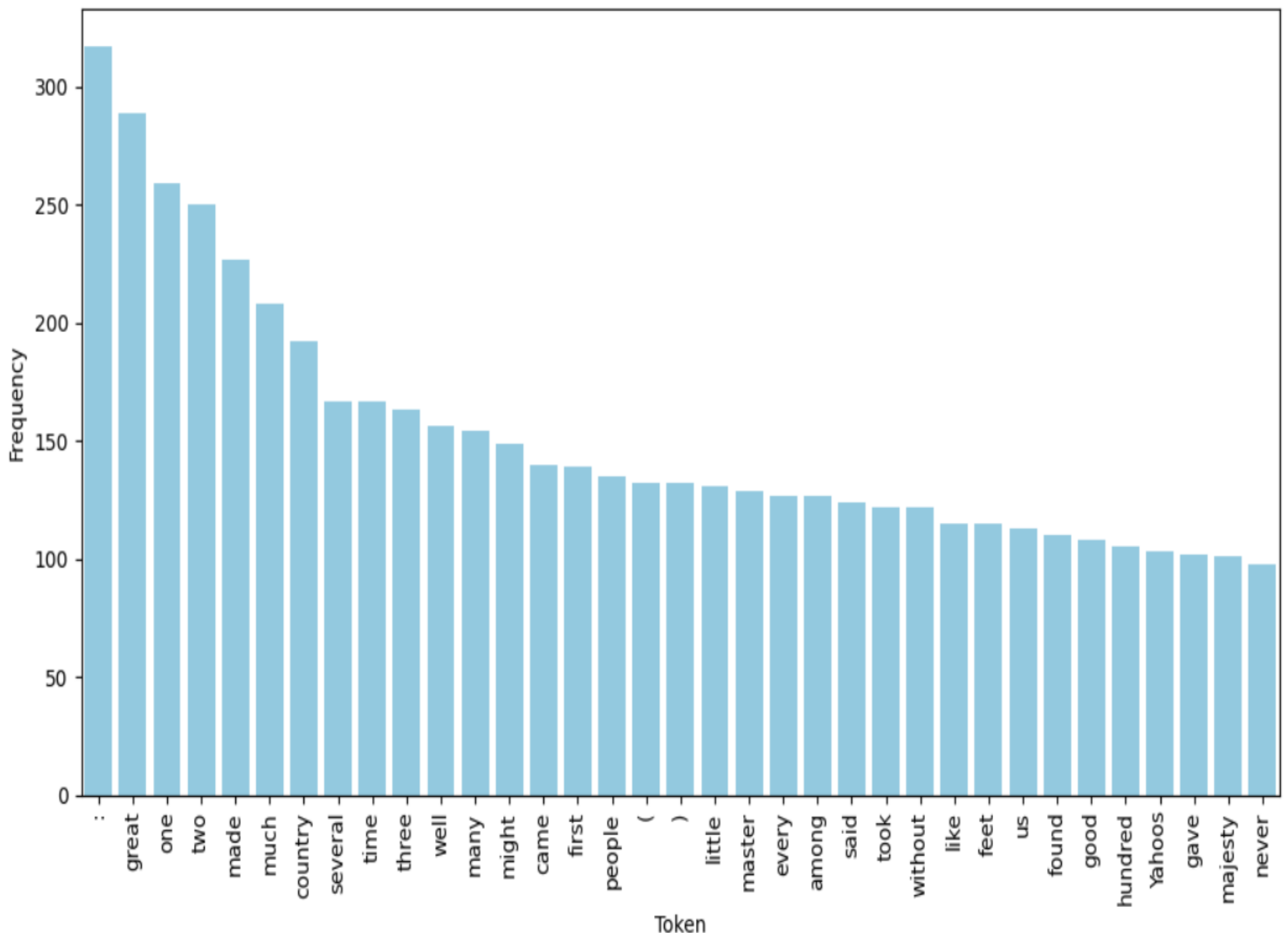
Index	Token	Frequency
0	,	9920
1	.	2679
2	;	1551
3	could	391
4	upon	374
5	would	370
6	:	317
7	great	289
8	one	259
9	two	250

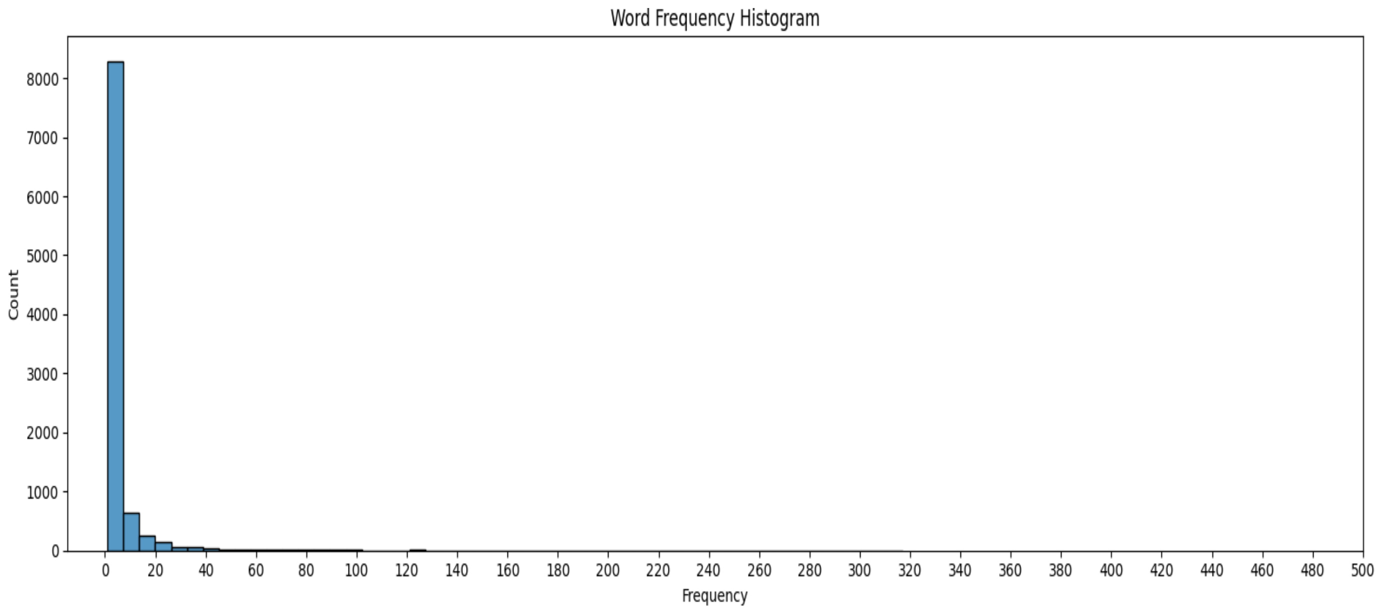
Step-4:

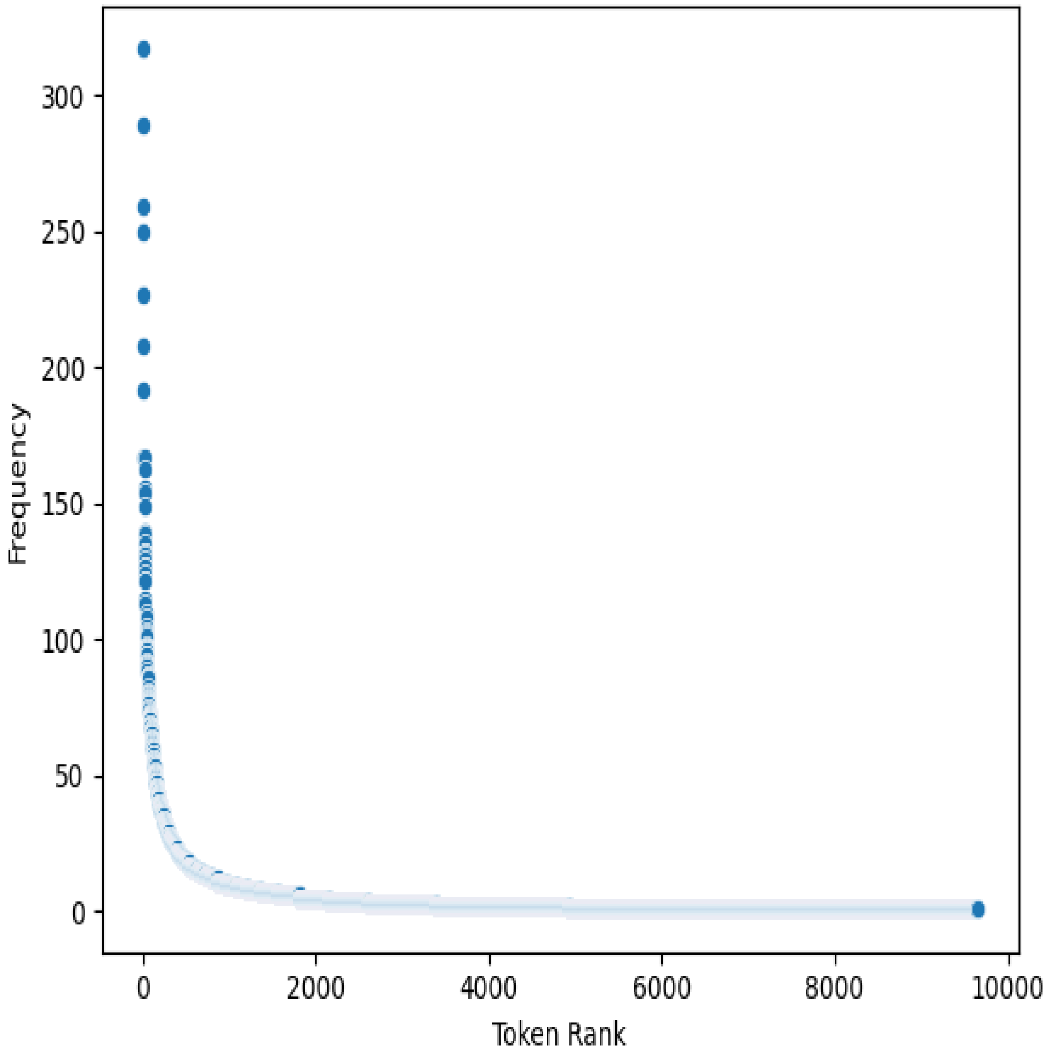
Next, we constructed a histogram. The x- axis has frequency of words from 0 to 500 in increments (no word has more frequency than 500; word ‘could’ has max frequency of 391). The graph of first 20 words of maximum frequency is shown below.

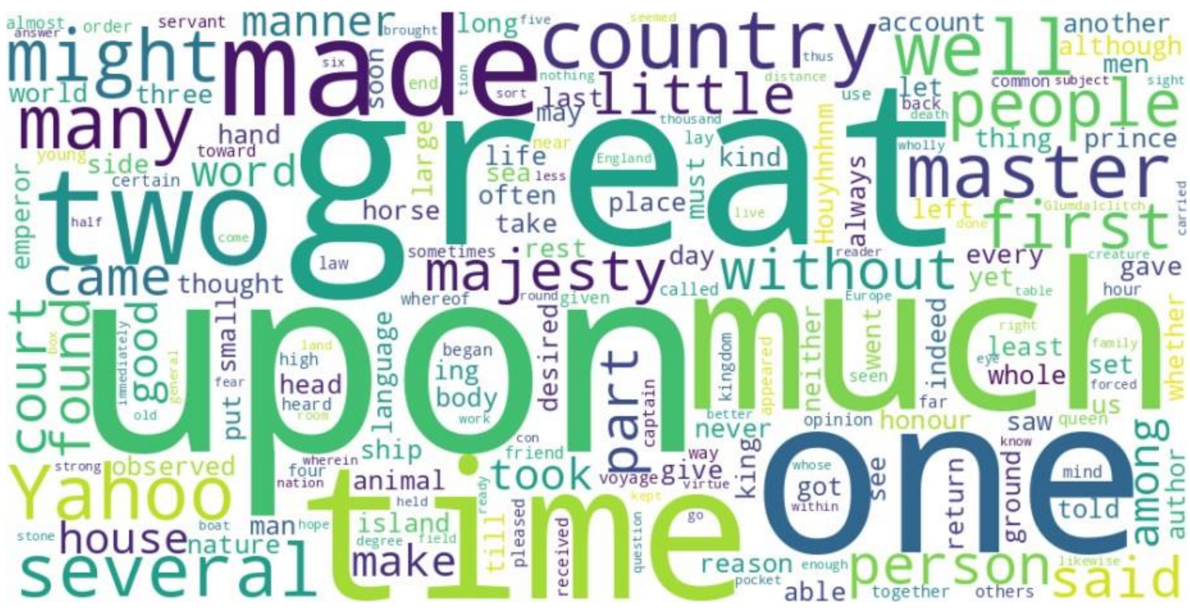
Step-5:

We have also created a scatter plot to visualize the relationship between the rank of words and their frequencies. This basically gives us a overall visual representation of how to frequency of words are being present in the text.




Word Frequency Histogram
Count
Frequency
0
1000
2000
3000
4000
5000
6000
7000
8000
0
20
40
60
80
100
120
140
160
180
200
220
240
260
280
300
320
340
360
380
400
420
440
460
480
500


Frequency
Token Rank
0
50
100
150
200
250
300
0
2000
4000
6000
8000
10000



WORD CLOUD

3. POS TAGGING

Step-1:

Firstly, Brown corpus was imported from NLTK library along with all of the available categories like ‘news’, ‘military’, ‘fiction’, etc. These were then combined and was named as ‘combined_corpus’.

Step-2:

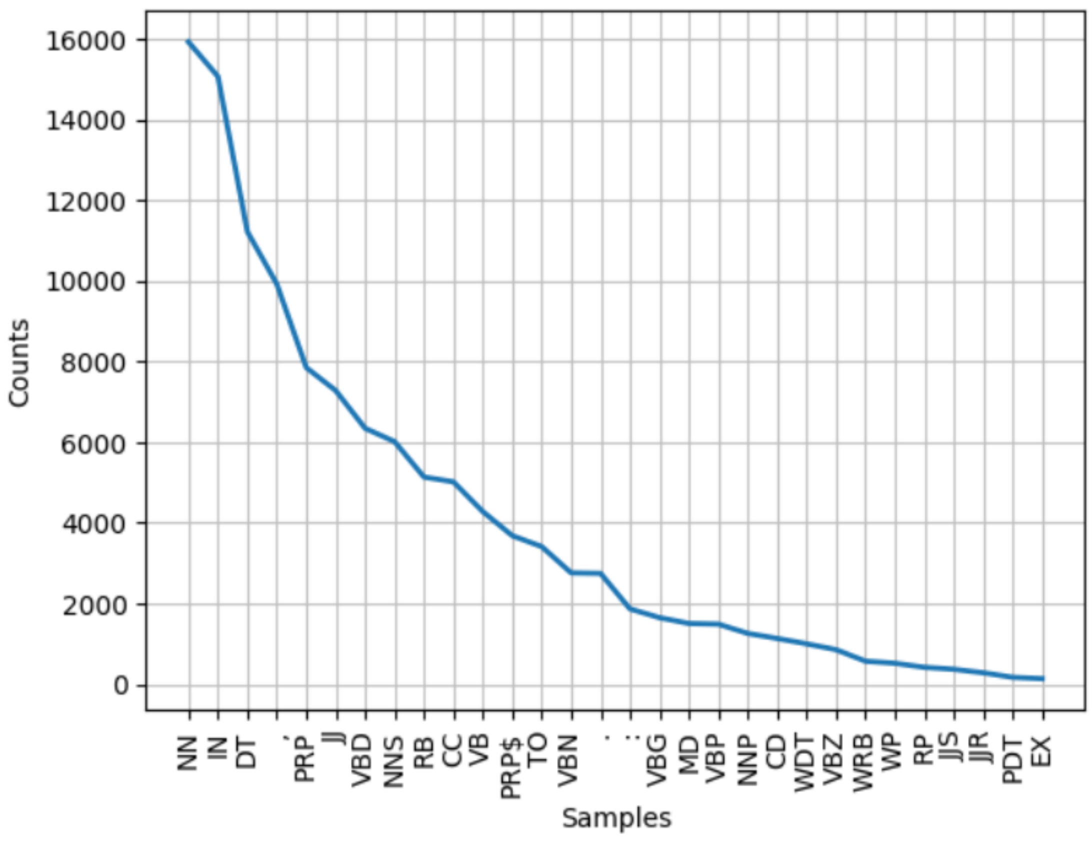
Then using the in-built POS tagger of NLTK library and the ‘combined_corpus’ made using Brown corpus all the ‘tokens’ were tagged with appropriate parts of speech.

Step-3:

Then a frequency distribution was created along with a dataframe and a categorical bar graph along with a line graph shows the result.

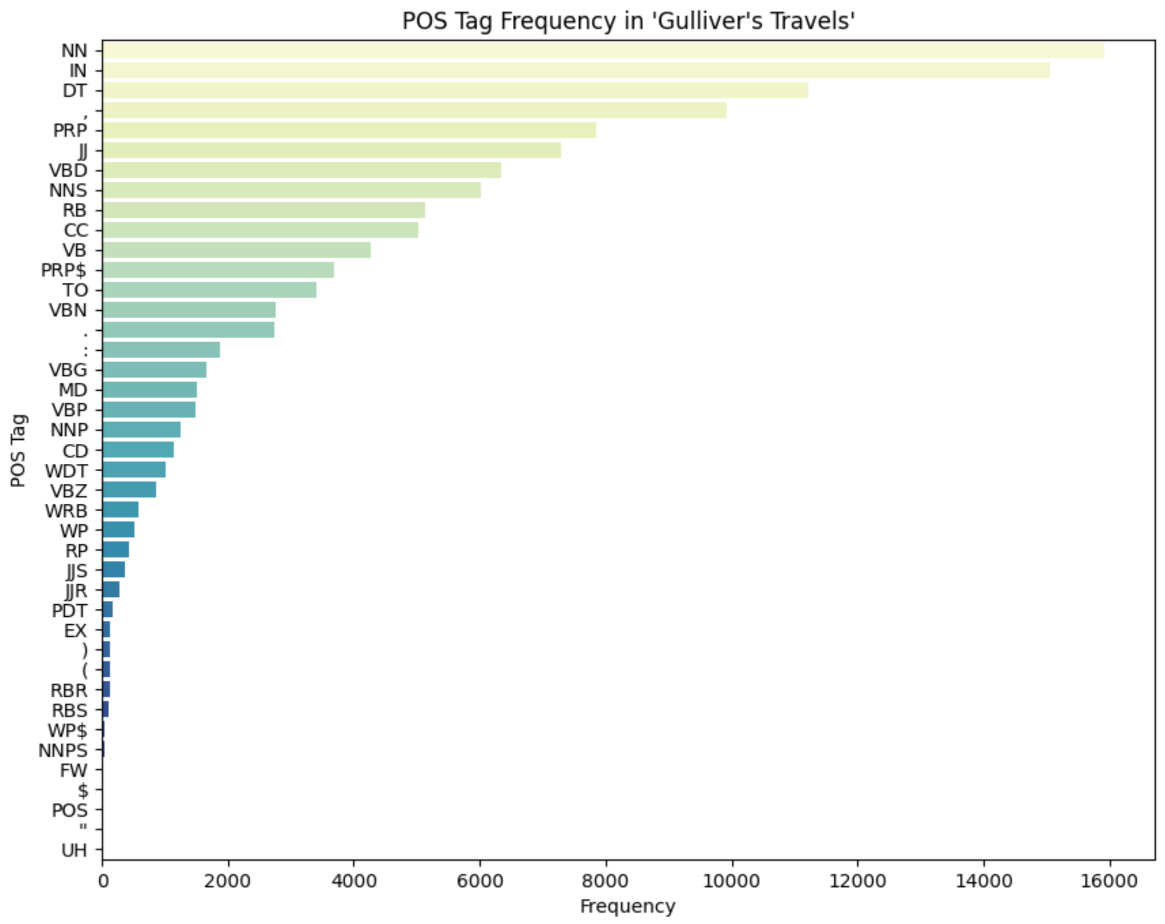
Insights:

Index	PoS Tag	Frequency
0	NN	15919
1	IN	15063
2	DT	11218
3	,	9920
4	PRP	7850
5	JJ	7280
6	VBD	6344
7	NNS	6015
8	RB	5136
9	CC	5018



Counts
16000
14000
12000
10000
8000
6000
4000
2000
0
NN IN DT , PRP JJ VBD NNS RB CC VB PRP$ TO VBN . : VBG MD VBP NNP CD WDT VBZ WRB WP RP JJS JJR PDT EX
Samples

```
<Figure size 640x480 with 0 Axes>
```



POS Tag Frequency in 'Gulliver's Travels'
POS Tag
NN IN DT , PRP JJ VBD NNS RB CC VB PRP$ TO VBN . : VBG MD VBP NNP CD WDT VBZ WRB WP RP JJS JJR PDT EX) (RBR RBS WP$ NNPS FW $ POS " UH
0 2000 4000 6000 8000 10000 12000 14000 16000
Frequency

4. BIGRAM MODELING

Step-1:

First, we copied the “pdf_text” which we got from the previous part of thecode into “sample_text”.
Now we were trying to find out the words which are in the format "Chapter [IVXLCDM]+" which is a regular expression.

Step-2:

Next we split the given text document into a list of chapters based on the specified regular expression pattern.

Step-3:

The following task involves finding the largest chapter among all the chapters to use as the training corpus. To find the largest chapter we iterate among all the chapters and compare the length of each chapter and then replace the larger chapter with the smaller one.

Step-4:

Now since we had found the largest chapter, we tokenize the “largest_chapter” using the function “word_tokenize” from NLTK library and store the tokens in “sample_tokens”.

Step-5:

Now we use the function bigrams()which also belongs to the NLTKLibrary,then save them in a list.

Now we use the counter() function from the collection module in the Python standard library by passing “sample_bigrams” to the function and save thecount in “bigram_counts”.
Now we also do the count for the “unigrams” using the counter() functionby passing “sample_tokens” into the function and save the count in the “unigram_counts”.

Step -6:

‘{ }’ denotes the dictionary comprehension and using the bigram_counts[bigram] and unigram_counts[bigram[0]].
The obtained result is a dictionary where the keys are bigrams, and the values are the probabilities of each bigram occurring, given the occurence of thefirst word.

Step-7:

We use the bigram_probabilities.items() function to retrieve the keyvalue pairs(basically the bigrams and their probabilities).
The whole data frame is stored in the variable df.

5. SHANNON GAME

Step-1:

The first step to the Shannon game would be tokenizing, first the selected chapter is saved in the “chapter_text”, the selected chapter comes from the random selection of the chapters where random is a function of the Python random module.

The tokenization is done by the function word_tokenize() from the NLTKlibrary.

We used it instead of split() because of the difference between word_tokenize and split() being that the word_tokenize recognises special characters like ‘.’ : ’,’ : ’!’ etc but the split()function doesn’t do that.

Step-2:

Now the blank positions are are mentioned in array, where we can replace the tokens with ‘***’
With method of iteration, we replace the mention positions with ‘***’.

Step-3:

Now we join the tokenized text and store it in “copy_chapter_tokens”, and print that which shows the text with few mentioned tokens replaced with ‘***’.

Step-4:

Now we use the function “fill_in_blanks” which takes input “chapter_tokens”, “blank_positions” and “bigram_probabilities”, thisfunction returns a tokenized text which then is joined and stored in “final_predicted_text”.

Step-5:

Now the “fill_in_blanks” function, copies the selected chapter tokensinto “filled_text”. Now we iterate through the “filled_text”,

Case-1: If it is a starting or ending word then skip that position.

Case-2: Now since it won’t be the start or end position now we store the previous word in the variable “previous_word”and use this the find all the possible next words and if there are any possible next words we choose the word with the maximum probability and store it in the “next_word”and replace the wordin the “filled_text” with the word in the “next_word” which contains the most probabilistic word.

This function (“fill_in_blanks”) returns the “ filled_text” whichcontains the updated text with the most probable words.

Step-6:

The final task for us is to find the accuracy, for that first we are saving the “selected_chapter” into “original_chapter”, and then tokenizing the words using the same function as we used before i.e word_tokenize().

Step-7:

We use the function “calculate_accuracy”function for finding the accuracy passing the values: “original_tokens”, “filled_text” and “blank_positions”.
The function calculate_accuracy does the following:

Case-1: Ambiguity

If the length of the “original_tokens” and “filled_text”are not the same then the code returns an ERROR.

Case-2:

Otherwise, it traverses along both the “original_text” and the “filled_text” and counts the number of correct and total predictions which are used to calculate the accuracy. After counting (excluding the start and end words) it divides the both (correct/total)and multiplying by 100, we get the accuracy which is returned.

Step-8:

Finally, the accuracy was printed, which shows most of the accuracy in between 25% to 50%.
This can be pointed out on the fact that the model was only trained on a particular chapter which was relatively larger than the other chapter. But certainly the training corpus is too small to predict any missing word in a sentence using probability apart from the very same chapter.