NLP Project Round-1

Team: The Markovs

Team Members

Durvesh Umesh Sangle: 20ucs068

Hasmit Chauhan: 20ucs078

Himanshu Sharma: 20ucs080

Manav Jangid: 20ucs111

Introduction

- **Data Description:**

  Data used for the project is the book, “A Concise Introduction to Software Engineering”, by Pankaj Jalote.

- **Data Preprocessing Steps**

  Steps undertaken before processing the data are:-

  1. Removal of images, tables, figures, running sections and chapter name
  2. Removal of extra space

- **Data Preparation**

  After performing preprocessing steps, we tokenize the text data.

  1. Tokenization
  2. Lemmatization
  3. Stemming

- **Problem Statement**

  Do PoS Tagging for the tokens of the “A Concise Introduction to Software Engineering”, by Pankaj Jalote” Book using TreeBank Tagset.

Data Preprocessing Steps

1. Removal of Images, Tables, Figures , Running section and Chapter Name :
    - To remove the images, tables and figures we extracted paragraphs from the book. Thus, only text data was left in the docx.
    - To remove the running sections and chapter names, we wrote a regular expression for strings having some text followed by a single digit and '.' for eg(1. Chapter name). Thereafter we searched the text data to match the regex and did not include those strings in our newly formed doc file. The regular expression for the following task is as mentioned below:

      `'[1-9] [A-Za-z ]+'`
    - Labels for the figures, given in our Data is of the form "Figure 1.1 : Figure name".To eliminate the Figures Labels, Labels were matched with the regular expression and were not included in the processed Data file.The regular expression for the following task is as mentioned below:

      `'Figure [0-9].[0-9]:[A-Za-z .]+'`
    - Labels for the Tables, given in our Data is of the form "Table 1.1 : Table name".To eliminate the Table Labels, Labels were matched with the regular expression and were not included in the processed Data file.The regular expression for the following task is as mentioned below:

      `'Table [0-9].[0-9]:[A-Za-z .]+'`

2. Removal of extra space :
    - Extra spaces in our raw data were removed by excluding the paragraph having less than 50 characters:

```
if len(text)>50:
    doc.add_paragraph(text)
```

Python Libraries used here are:

- **docx** - It provides functions to handle document files. Main functions include, creating a new docx file, adding headers and paragraphs to it, etc.

- **re** - It helps with regular expressions. A regular expression (or RE) specifies a set of strings that matches it; the functions in this module let you check if a particular string matches a given regular expression (or if a given regular expression matches a particular string, which comes down to the same thing).

```
#******Removing figures, images, tables, headings from book******##
import docx
import re
doc = docx.Document()
def readtxt(filename):
    doc1 = docx.Document(filename)
    for para in doc1.paragraphs:
        newtext = para.text
        text = re.sub(r'[1-9] [A-Za-z ]+', '', newtext)
        text = re.sub(r'Figure [0-9].[0-9]:[A-Za-z .]+', '', text)
        text = re.sub(r'Table [0-9].[0-9]:[A-Za-z .]+', '', text)
        if len(text)>50:
            doc.add_paragraph(text)

readtxt('data.docx')
doc.save('preProcessedData.docx')
```

Figure: Code for Pre-Processing the Data

Data Preparation Steps

1. *Tokenization:-*

```
#**********Tokenization**********##
import nltk
fullTokens = []
def tokenization(filename):
    doc1 = docx.Document(filename)
    for para in doc1.paragraphs:
        nltk_tokens = nltk.word_tokenize(para.text)
        for token in nltk_tokens:
            fullTokens.append(token)

tokenization('preProcessedData.docx')
with open("tokens.txt", "w",encoding='utf-8') as outfile:
    outfile.write("\n".join(fullTokens))
```

Figure: Code for Tokenization

Tokenization is performed for each paragraph in our text data file using the nltk library's word_tokenize function, and all tokens are saved in a list called fullTokens.

2. *Lemmatization:-*

```
##**********Lemmatization**********##
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
punctuations="?:!.,;)([\"}{*"
lemmaList = []
for word in fullTokens:
    if word not in punctuations:
        lemmaList.append(wordnet_lemmatizer.lemmatize(word))

with open("tokensAfterlemma.txt", "w",encoding='utf-8') as outfile:
    outfile.write("\n".join(lemmaList))
```

Figure: Code for Lemmatization

The tokens from list ‘fullTokens’ are given as input to the WordNetLemmatizer, it returns the tokens after lemmatization, we add that token to the lemmalist.

3. *Stemming:-*

```
56  # ##**********************Stemming************************##
57  from nltk.stem import PorterStemmer
58  stemList = []
59  ps = PorterStemmer()
60  for w in lemmaList:
61      stemList.append(ps.stem(w))
62
63  with open("tokensafterStem.txt", "w",encoding='utf-8') as outfile:
64      outfile.write("\n".join(stemList))
65
```

Figure: Code for Stemming

The tokens from list ‘lemmaList’ are given as input to the PorterStemmer(), it returns the tokens after stemming, we add that token to the stemList.

Python Library Used:-

nltk - NLTK, or Natural Language Toolkit, is a Python package that you can use for NLP. A lot of the data that you could be analyzing is unstructured data and contains human-readable text. Before you can analyze that data programmatically, you first need to preprocess it.

Functions used are:-

1. *Word_tokenize()* – To tokenize the text.
2. *WordNetLemmatizer()* - To lemmatize the tokens.
3. *PorterStemmer()* - To do stemming on tokens.

Graphs and visual inference

- ***Calculating the frequency of tokens:-***

```
66
67  # ##************frequency distribution*****************##
68  freq = {}
69  def freqCount(List):
70      for token in list:
71          if(token in freq):
72              freq[token] +=1
73          else:
74              freq[token] = 1
75
76  freqCount(stemList)
77  file = open('freq.txt','w',encoding='utf-8')
78  for key, value in freq.items():
79      file.write(key+" :"+str(value)+"\n")
```

Figure: Code for calculating frequency of Tokens

Frequency is calculated by iterating over all tokens in stemList. It is stored in a dictionary named '*freq*'.

- ***Bar Plot for Top 50 frequent words:-***

```
82  #******************bar plot for top 50 frequent words******************#
83  import collections
84  import itertools
85
86  sorted_x = sorted(freq.items(), key=lambda kv: kv[1],reverse=True)
87  sorted_dict = collections.OrderedDict(sorted_x)
88
89  top = dict(itertools.islice(sorted_dict.items(), 50))
90  import matplotlib.pyplot as plt
91  sizes = list(top.values())
92  labels = list(top.keys())
93  plt.barh(labels,sizes)
94  plt.yticks(fontsize=7.5)
95  plt.title("Word vs Frequency")
96  plt.savefig("barGraph"+".png", bbox_inches='tight')
97  plt.show()
98
```

Figure: Code for generating Bar plot of Top 50 frequent words (including stopWords) and its frequency

'freq' dictionary is sorted in the decreasing order of the magnitude of frequency values which is stored in *'sorted_x'.*

Python Library Used:-

collections - They are containers used to store data, commonly known as data structures. Lists, tuples, arrays, dictionaries, etc. Python has a built-in collections module that provides additional data structures for data collections.

itertools - It is a Python module used to iterate over data structures that can be skipped in for loops. This module serves as a fast and memory efficient tool, used alone or in combination to form iterator algebras.

matplotlib.pyplot - It is a comprehensive library for creating static, animated and interactive visualizations in Python. Matplotlib makes simple things easy and hard things possible. Create interactive characters that can zoom, pan and update.
Functions used are:-

1. *collections.OrderedDict()* - to track the order in which items were added.
2. *itertools.islice()* - to handle iterators for top 50 frequent words.
3. *plt.barh(labels,sizes)* - to specify that the bar chart is to be plotted by using the labels column as the Y-axis, and the sizes as the X-axis.
4. *plt.yticks()* - to specify the font size of words on Y-axis.
5. *plt.title()* - to specify the title to the bar chart.
6. *plt.savefig* - to save the graph figure in the storage.
7. *plt.show()* - to show the output of the bar chart.

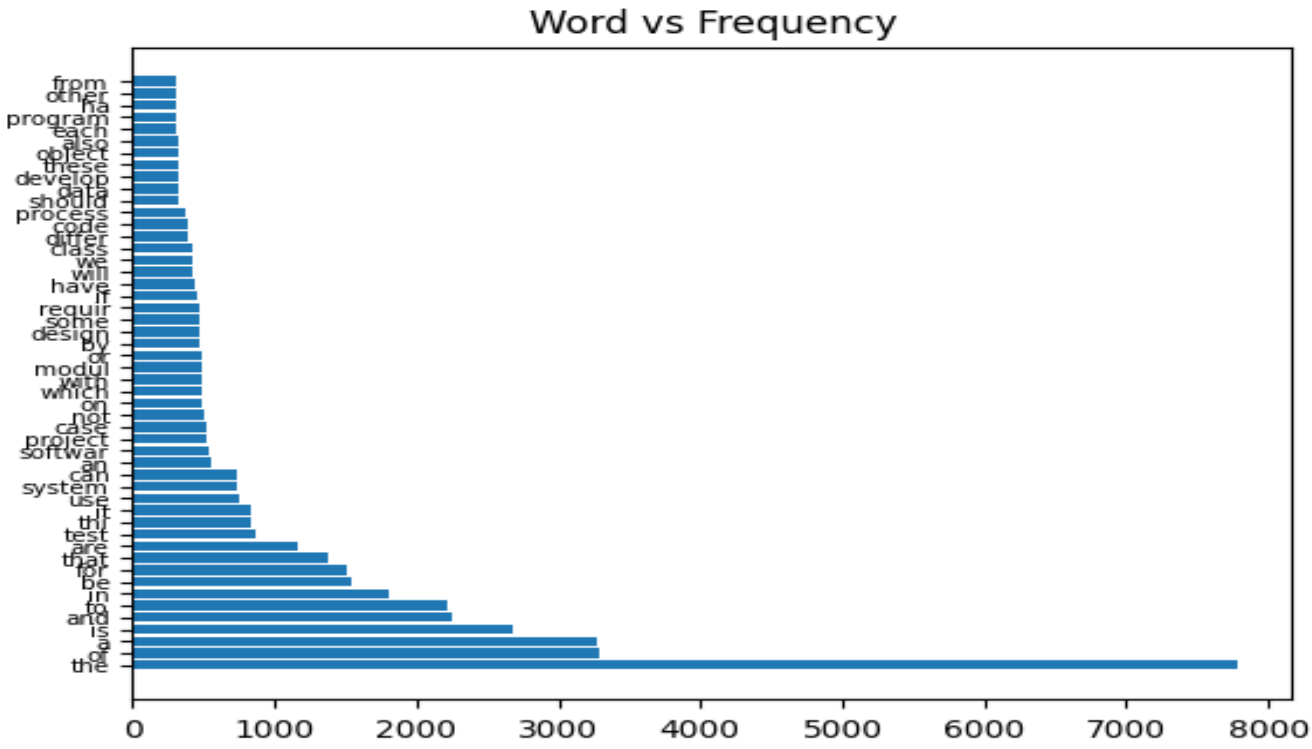


Figure: Bar Plot for Top 50 frequent words including stopWords and its frequency

- ***Word Cloud for all the tokens in our Data file:-***

```
# ##********************Word Cloud************************##
from wordcloud import WordCloud
wcloud = WordCloud().generate_from_frequencies(freq)
plt.figure()
plt.imshow(wcloud, interpolation="bilinear")
plt.axis("off")
plt.savefig("WordCloudWithStopWords"+".png", bbox_inches='tight')
plt.show()
```

Figure: Code for Generating Word Cloud of all the Tokens(Including StopWords)

The *'freq'* dictionary is used to create the word cloud for the set of words **with stop words included**.

Python Library Used:-

WordCloud - WordCloud is a technique that displays the most frequently used words in a given text. We can create word cloud by importing this library and then by selecting the Dataset and selecting the Text and amount of text for Word Cloud by using the function *WordCloud.generate_from_frequencies(freq).*

Function(s) Used:-

1. *WordCloud.generate_from_frequencies(freq)* - take a dictionary of words and their frequencies (here dictionary name is *freq*) and create a word cloud from the counts.
2. *plt.figure()* - used to create a new figure.
3. *plt.imshow()* - matplotlib function imshow() creates an image from a 2-dimensional numpy array.
4. *plt.axis()* - used to set some axis properties to the graph.
5. *plt.savefig()* - to save the graph figure in the storage.
6. *plt.show()* - to show the output of the bar chart.

Figure: Word Cloud of all the Tokens(including Stop Words)

- ***Removing Stop Words:-***

```
# # ##********************Remove Stop Words********************##
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
filteredList = []
for w in stemList:
    if w not in stop_words:
        filteredList.append(w)
with open("tokensWithoutStopWords.txt", "w",encoding='utf-8') as outfile:
    outfile.write("\n".join(filteredList))

```

Figure: Code for removing StopWords

- ***Frequency Distribution After Removing Stop Words:-***

Frequency is calculated by iterating over all tokens in stemList after removing stop words. It is stored in a dictionary named '*freq1'.*

```
# # ##**************Frequency Distribution After Removing Stopwords**************##
freq1 = {}
def freqCount(list):
    for token in list:
        if(token in freq1):
            freq1[token] +=1
        else:
            freq1[token] = 1
freqCount(filteredList)

```

Figure: Code for frequency distribution after removing stopwords

- ***Bar Plot for Top 50 frequent words after removing stop words:-***

'freq' dictionary is sorted in the decreasing order of the magnitude of frequency values which is stored in *'sorted_y'.*

```
136  import collections
137  import itertools
138
139  sorted_y = sorted(freq1.items(), key=lambda kv: kv[1],reverse=True)
140  sorted_dict1 = collections.OrderedDict(sorted_y)
141
142  top1 = dict(itertools.islice(sorted_dict1.items(), 50))
143  import matplotlib.pyplot as plt
144  sizes1 = list(top1.values())
145  labels1 = list(top1.keys())
146  plt.barh(labels1,sizes1)
147  plt.yticks(fontsize=7.5)
148  plt.title("Word vs Frequency")
149  plt.savefig("barGraph2"+".png", bbox_inches='tight')
150  plt.show()
```

Figure: Code for Bar Plot for Top 50 frequent words after removing stop words

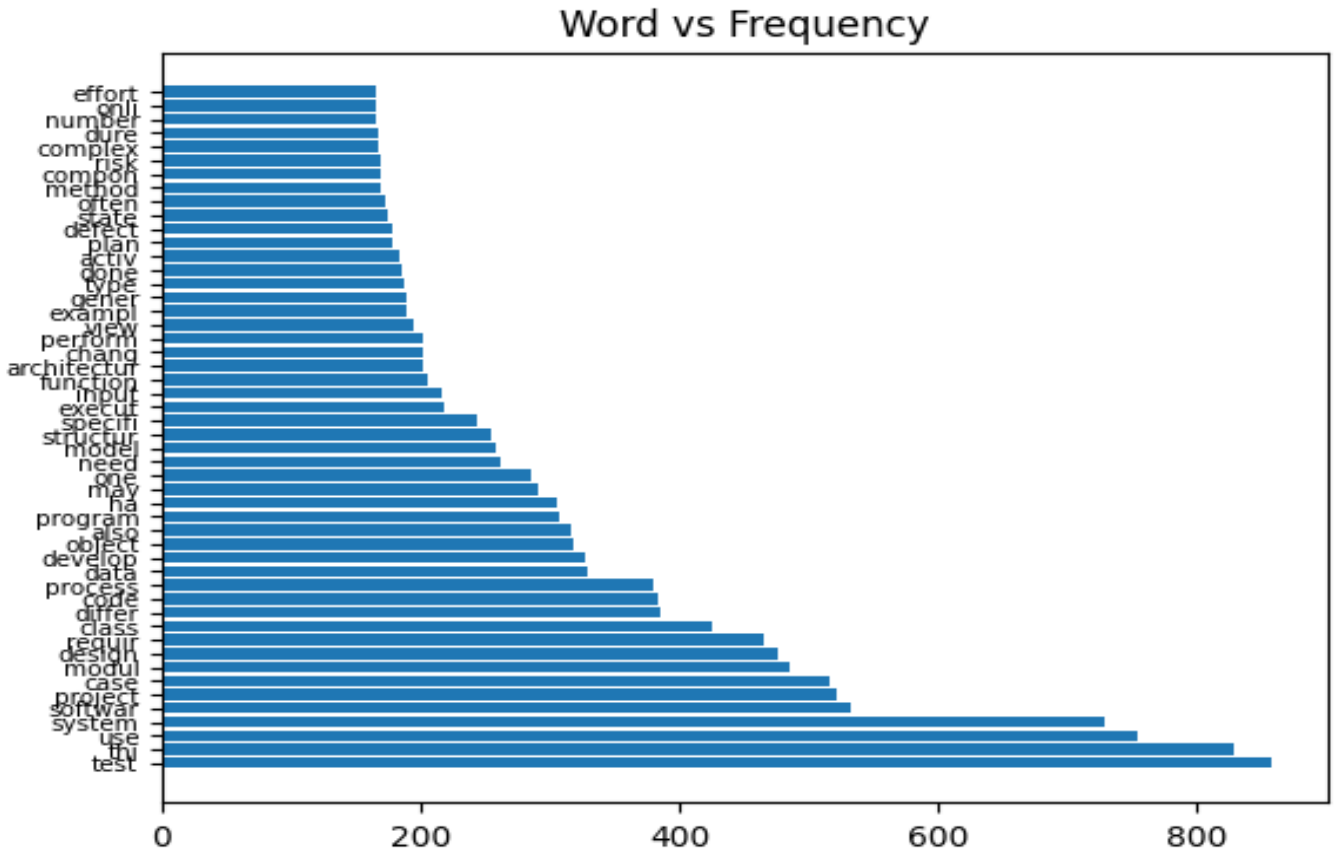


Figure: Bar Plot of Top 50 frequent words(excluding stopWords) and their Frequency

- ***Word Cloud for all the tokens in our Data file after removing stop words:-***

The *'freq'* dictionary is used to create the word cloud for the set of words **after removing stop words**.

```
#****************Word Cloud after removing stop words************************#
wcloud = WordCloud().generate_from_frequencies(freq1)
plt.figure()
plt.imshow(wcloud, interpolation="bilinear")
plt.axis("off")
plt.savefig("WordCloud"+".png", bbox_inches='tight')
plt.show()

```

Figure: Code for Word Cloud of all the Tokens excluding StopWords

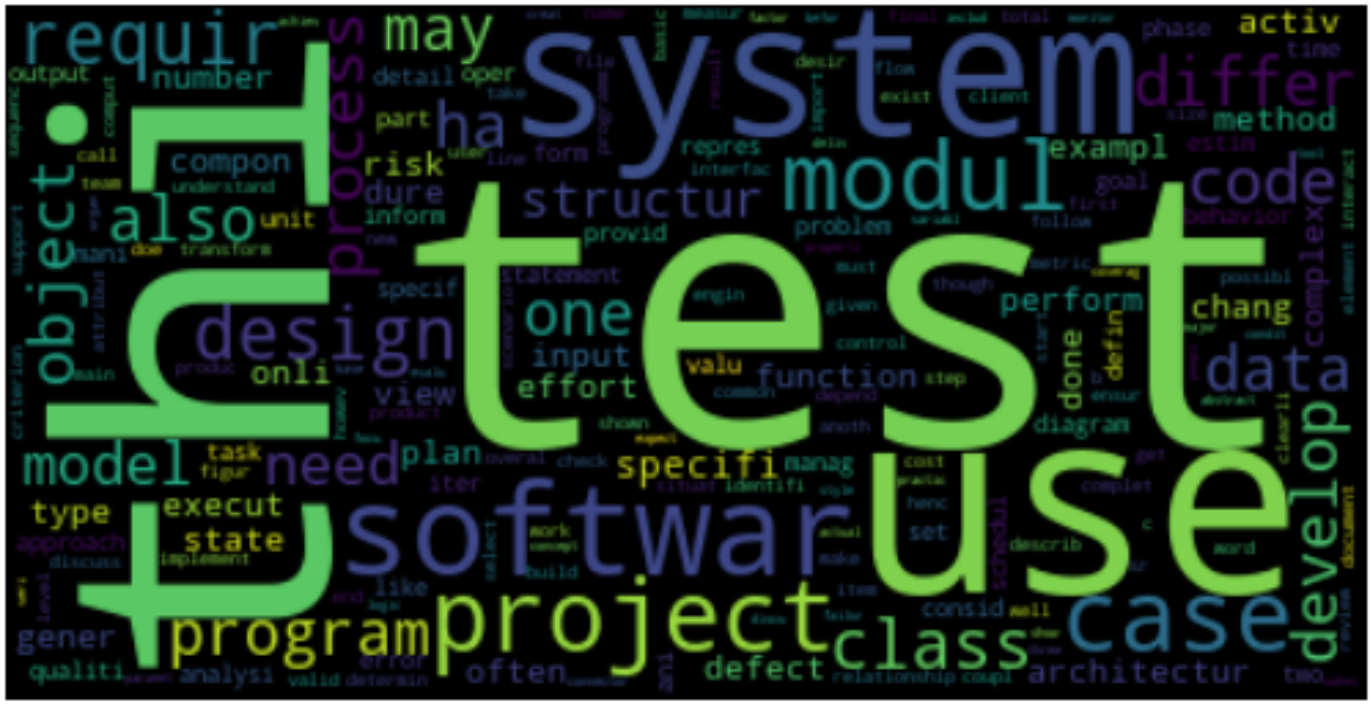


Figure: Word Cloud of all the Tokens(excluding StopWords)

- ***Relationship between word length and frequency and It's Graph Plotting:-***

This function uses stemList and stores length of words with its corresponding frequency of occurrence and stores it in a txt file named freqWithLength.

```
# ##*****************RelationShip b/w word Length and frequency*******************##
from collections import Counter
counts = Counter(len(word) for word in stemList)
wordLengths = []
frequencies = []
file = open('freqWithLength.txt','w',encoding='utf-8')
file.write(" Len     Freq")
for length in range(1, max(counts.keys()) + 1):
    file.write("\n")
    wordLengths.append(length)
    frequencies.append(counts.get(length,0))
    file.write(f'{length:4d} {counts.get(length, 0):6d}')

#******************Graph Plotting*****************************#

plt.barh(wordLengths,frequencies)
plt.yticks(fontsize=7.5)
plt.title("Word length v/s Frequency")
plt.savefig("barGraph"+".png", bbox_inches='tight')
plt.show()
```

Figure:Code for generating the bar plot for word length and frequency

<u>Python Library Used:-</u>

Counter - It is a dict subclass for counting hashable objects. A collection that stores items as dictionary keys and their counts as dictionary values.

Function(s) used:-

1. *Counter(len(word) for word in stemList)* - it returns the number of times an object appears in a list.

After this, the program is plotting the bar graph between word length and frequency.

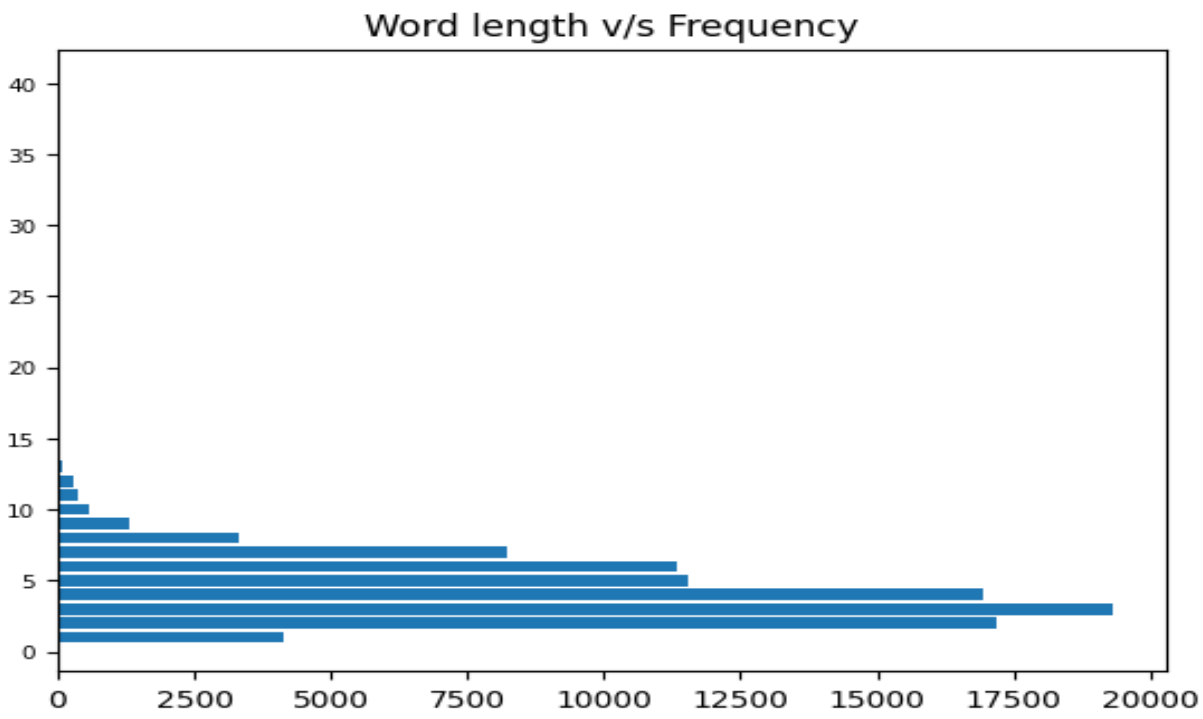


Figure: Bar Plot of Word length and Frequency

POS Tagging

This function uses fiteredList() for the final **POS tagging using TreeBank Tagset**, which is then stored in a txt file named as taggedTokens.

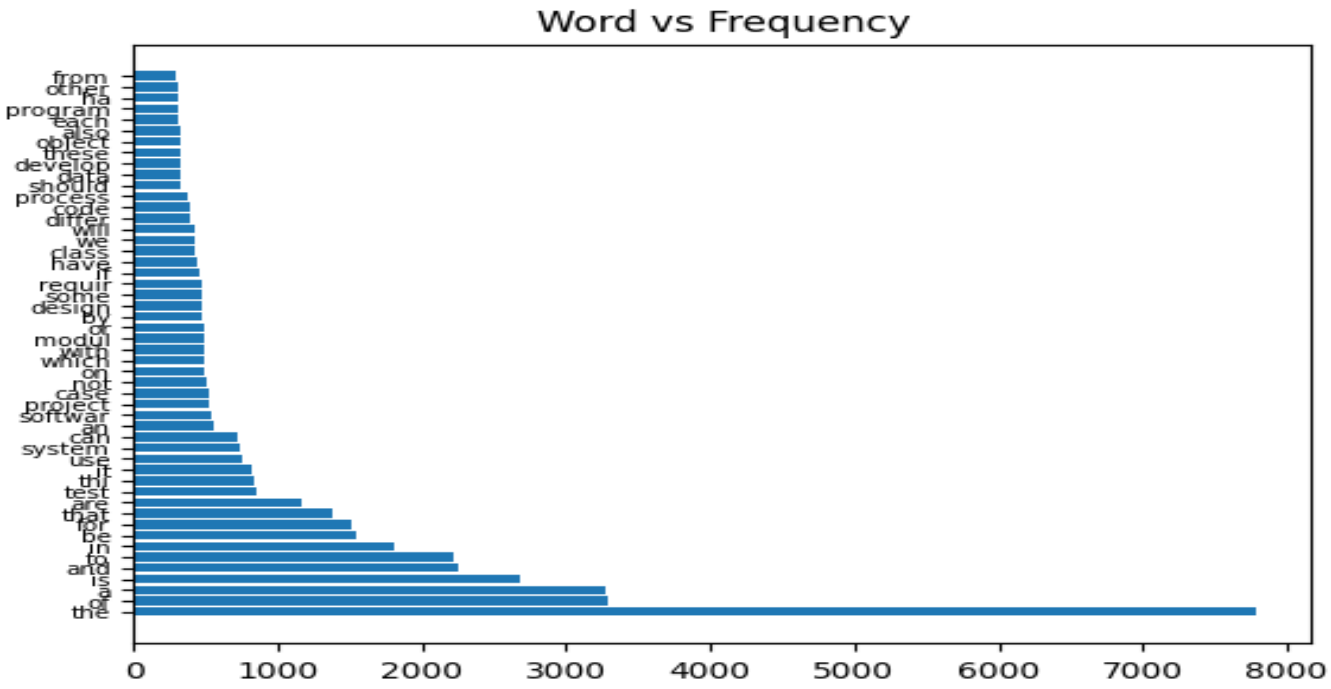
```
150  # ##********************POS Tagging**********************************##
151  from nltk import tag
152  taggedTokens = tag.pos_tag(filteredList)
153  with open("taggedTokens.txt", "w",encoding='utf-8') as outfile:
154      outfile.write("\n".join(" ".join(tup) for tup in taggedTokens))
```

Figure: Code for POS Tagging using TreeBank Tagset

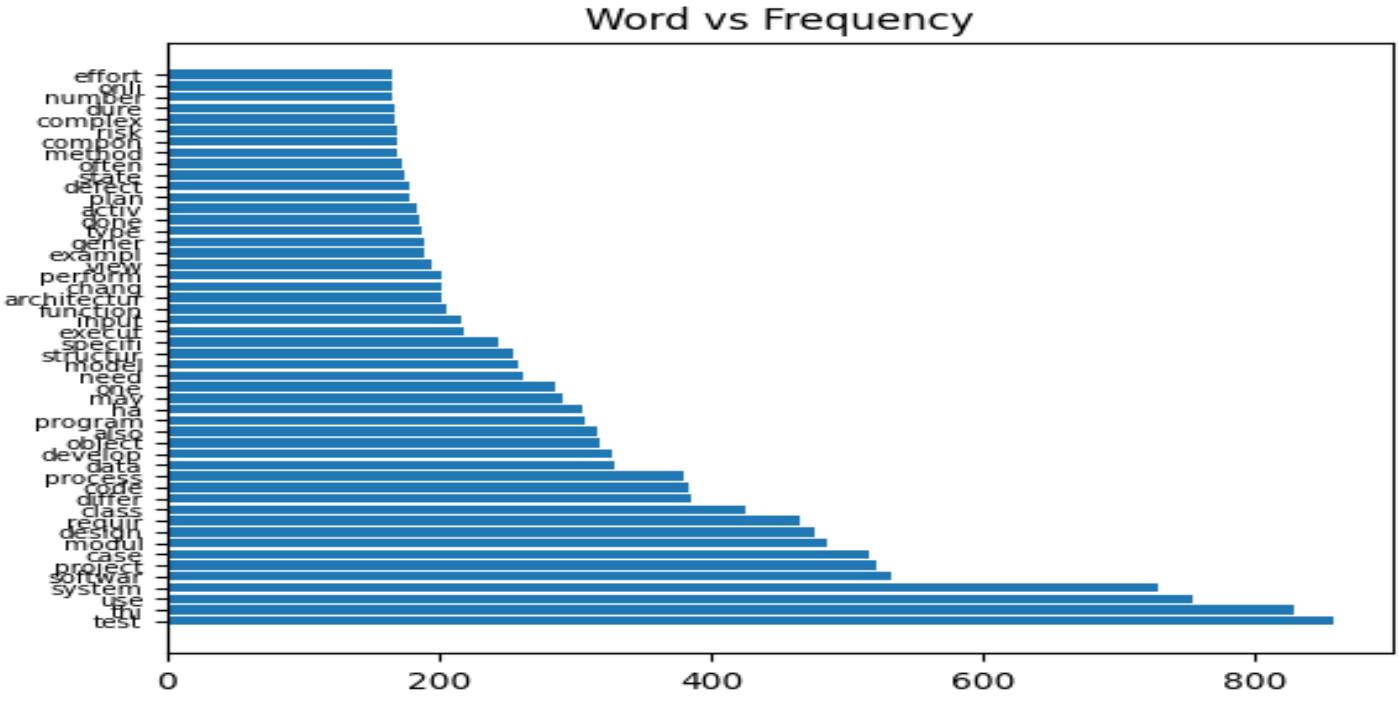
Observation

❖ **Difference between the Bar Plot of Word vs Frequency with stopWords and Without StopWords**

- *With stop words:-*



- *Without stopWords:-*

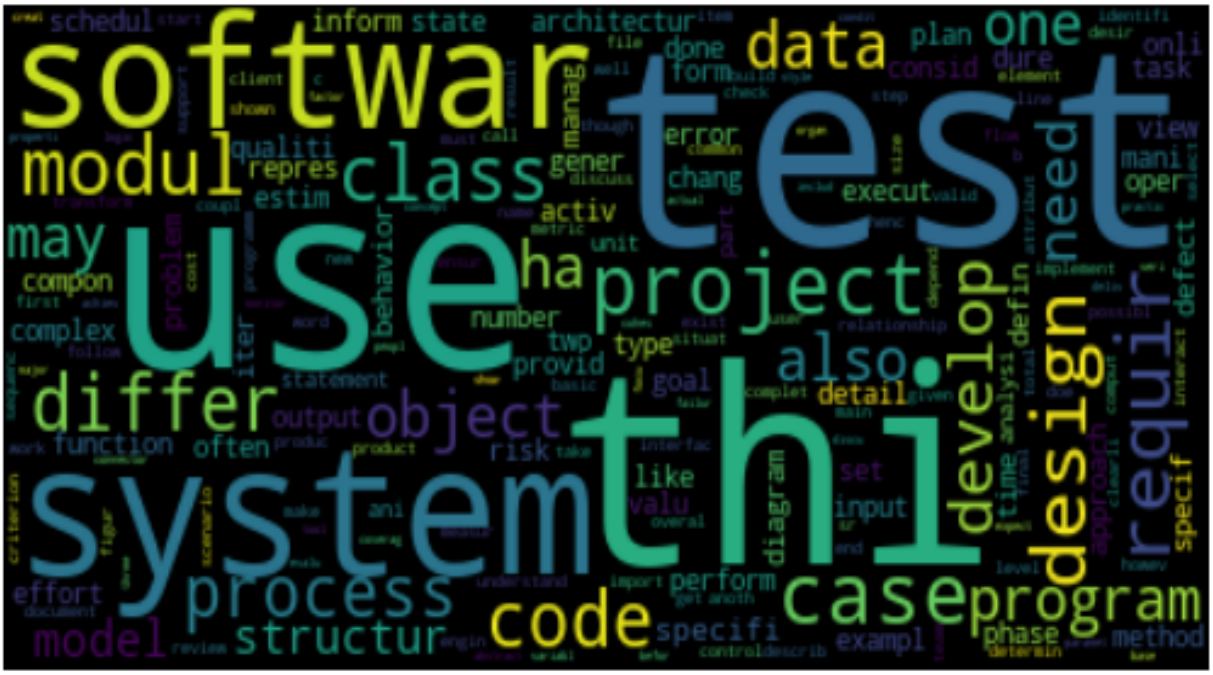


- *The Bar Plot of word vs frequency including stop words contains **'the'** with the highest frequency with count of 7800 approximately.*
- *The words such as **'of'**, **'a'**, **'is'**, **'and'**, **'to'**, **'be'**,.... are having the frequency around 1000-3500 which consist of a very large set of our corpus.*
- *The Bar plot of word vs frequency excluding stop words contains **'test'** with the highest frequency with the count of 820 approximately.*
- *By this observation we can conclude that most of the set of our data (65%) comprises stopWords.*

❖ ***Difference between Word Cloud with and without stop words***

- ***With stop words:-***

- ***Without stopWords:-***



DIfference:-

Before removing stopwords words like, ***'the', 'is', 'a', 'to'*** *, etc are having large font sizes in the word cloud. Whereas, after removing stop words,* ***'test', 'use', 'systems', 'thi'*** *,etc,. have large font sizes.*

❖ *GitHub Link:*

The project's code, generated text file and images is available in the following gitHub link:

https://github.com/manavjangid5/The_Markovs_NLP_Project