Github Repo :
https://github.com/20-gogo/NLP-project

Team Name : Optimizers

Gaurank Maheshwari -19ucs179

Gunit Varshney - 19ucs188

Nikita Saboo - 19dcs012

NLP PROJECT ROUND 1

OVERVIEW

In this assignment, we will perform text analysis on two of the Gutenberg works that we have chosen. "Pride and Prejudice" and "A Tale of Two Cities" are two classic novels. Following that, we'll proceed to apply POS Tagging to both books. All of this will be accomplished utilizing natural language processing techniques and Python modules.

BOOKS USED

Pride and Prejudice

By Jane Austen

A Tale of Two Cities

By Charles Dickens

GOALS

1. Import the text from two books, let's call it as T1 and T2.
2. Perform simple text pre-processing steps and tokenize the text T1 and T2.
3. Analyze the frequency distribution of tokens in T1 and T2 separately.
4. Create a Word Cloud of T1 and T2 using the token that you have got.
5. Remove the stopwords from T1 and T2 and then again create a word cloud.
6. Compare with word clouds before the removal of stopwords.
7. Evaluate the relationship between the word length and frequency for both T1 and T2.
8. Do PoS Tagging for both T1 and T2 using anyone of the four tagset studied in the class and Get the distribution of various tags

SPECIFICATIONS

Python Libraries used in this project

Urllib - Used to fetch text data from Gutenberg URLs

NLTK - Used for Tokenizing, Lemmatization and Removing Stopwords

Re - Used to remove URLs and Decontract Contractions in English Language

Wordcloud - Used to create WordClouds from Tokenized Data

Inflect - Used to replace numbers with words

Maplotlib - Used to Visualize our text data

MILESTONES

Data Description

After Importing from URL, sample book data would look like :

```
b'\xef\xbb\xbf\r\nThe Project Gutenberg EBook of Pride and Prejudice, by
Jane Austen\r\n\r\nThis eBook is for the use of anyone anywhere at no cost
and with\r\nalmost no restrictions whatsoever.  You may copy it, give it
away or\r\nre-use it under the terms of the Project Gutenberg License
included\r\nwith this eBook or online at
www.gutenberg.org\r\n\r\n\r\nTitle: Pride and Prejudice\r\n\r\nAuthor:
Jane Austen\r\n\r\nRelease Date: August 26, 2008 [EBook #1342]\r\nLast
Updated: November 12, 2019\r\n\r\n\r\nLanguage: English\r\n\r\nCharacter
set encoding: UTF-8\r\n\r\n*** START OF THIS PROJECT GUTENBERG EBOOK PRIDE
AND PREJUDICE ***\r\n\r\n\r\n\r\n\r\nProduced by Anonymous Volunteers, and
David Widger\r\n\r\nTHERE IS AN ILLUSTRATED EDITION OF THIS TITLE WHICH
MAY VIEWED AT EBOOK\r\n[# 42671 ]\r\n\r\ncover\r\n\r\n\r\n\r\n\r\n
Pride and Prejudice\r\n\r\n      By Jane Austen\r\n\r\n
CONTENTS\r\n\r\n         Chapter 1\r\n\r\n         Chapter 2\r\n\r\n
Chapter 3\r\n\r\n         Chapter 4\r\n\r\n         Chapter 5\r\n\r\n
Chapter 6\r\n\r\n         Chapter 7\r\n\r\n         Chapter 8\r\n\r\n
```

We infer that :

- The book contains legal documentation which is of no use to us (Before "START OF THIS PROJECT" and after "END OF THIS PROJECT")
- There are numbers that are not meaningful to us directly, so we need to convert them to words
- There are a lot of contractions and punctuations in english and we need to convert them to meaningful data
- There are a lot of special characters, URLs
- There are no emojis, emoticons, or chat words

Data Preprocessing Steps

1. **Discard Useless Portion of Book**

   We will discard the documentation part of the book that is of no use to us.

```
def discard_useless_part (text):
  sidx = text.find('*** START OF THIS PROJECT ')
  eidx = text.find('*** END OF THIS PROJECT ')
  print("Discarding Before - ", sidx)
  print("Discarding After - ", eidx)
  text = text[sidx:eidx]
  return text
```

2. **Convert all data to lowercase**

   We will convert all text data to lowercase, as the case does not contribute much to the meaning of data.

```
def to_lower(text):
  return text.lower()
```

3. **Converting Number to Words**

   For this, we use inflect Python Library which has a function p.number_to_words that will give us the english equivalent of a number using basic mapping techniques.

```
def num2word(text):
  list_of_words = text.split()
  modified_text = []

  for word in list_of_words:
    if word.isdigit():
      number_in_word = p.number_to_words(word)
      modified_text.append(number_in_word)
    else:
      modified_text.append(word)

  return ' '.join(modified_text)
```

4. **Removing Contractions and Punctuations**

We will do this using a Python Library `re` that will help us apply regular expressions on our data as desired.

```
def decontracted(text)
    # specific
    text = re.sub(r"won\'t", "will not", text)
    text = re.sub(r"can\'t", "can not", text)

    # general
    text = re.sub(r"n\'t", " not", text)
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"\'s", " is", text)
    text = re.sub(r"\'d", " would", text)
    text = re.sub(r"\'ll", " will", text)
    text = re.sub(r"\'t", " not", text)
    text = re.sub(r"\'ve", " have", text)
    text = re.sub(r"\'m", " am", text)
    return text
```

```
def remove_punctuation(text):
  tokens = word_tokenize(text)
  words = [word for word in tokens if word.isalpha()]
  return ' '.join(words)
```

5. **Removing URLs**

Again, we would do this using `re`

```
def remove_URL(text):
    return re.sub(r"http\S+", "", text)
```

6. **Lemmatization**

We do Lemmatization with the help of `WordNetLemmatizer()` function from `nltk.stem` which gives the lemmatized form of all verbs

```
def lemmatize_word(text):
    word_tokens = word_tokenize(text)
    lemmas = [lemmatizer.lemmatize(word, pos ='v') for word in word_tokens]
    return ' '.join(lemmas)
```

Data Preparation

We apply all the functionalities we added above and Prepare our data for analysis.

```
def PreProcessedBook(url):
  book = read_book(url)
  print_book_title_and_length(book)
  text = decode_book(book)
  text = discard_useless_part(text)
  text = to_lower(text)
  text = remove_URL(text)
  text = decontracted(text)
  text = num2word(text)
  text = remove_punctuation(text)
  text = lemmatize_word(text)
  return (text)
```

```
book1_text = PreProcessedBook(url1)
book2_text = PreProcessedBook(url2)
```

Problem Statements and Inferences

- **Analyze the frequency distribution of tokens in T1 and T2 separately**

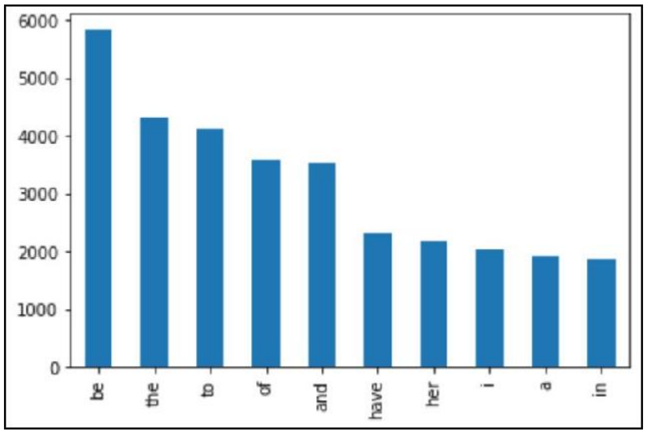
  For this step, we take help of python library `pandas`

  First we tokenize the given data, and then we plot a histogram of top 10 most frequent words.

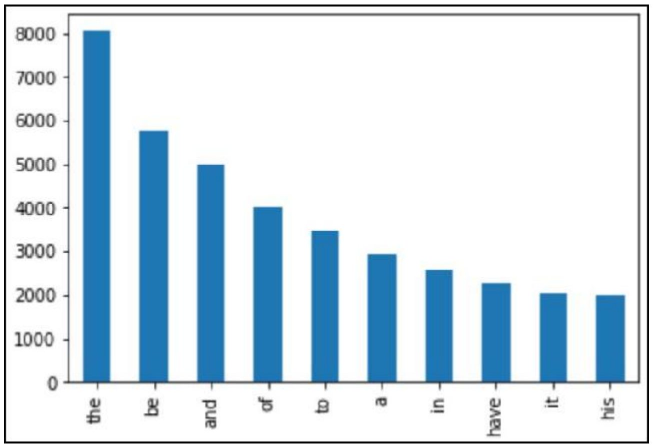
```
T1 = word_tokenize(book1_text)
pd.Series(T1).value_counts()[:10].plot(kind='bar')
```

```
T2 = word_tokenize(book2_text)
pd.Series(T2).value_counts()[:10].plot(kind='bar')
```

Frequency Distribution of T1



Frequency Distribution of T2



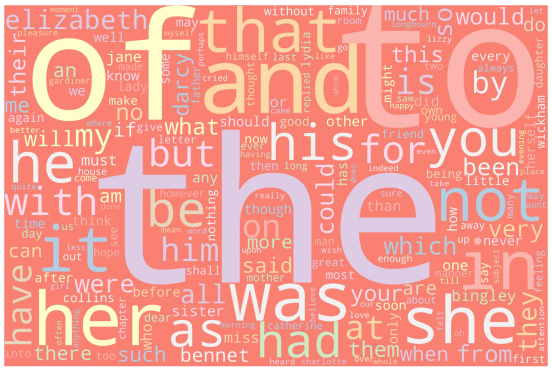
- **Generating a word cloud from T1 and T2**

For this we take the help of a python library `wordcloud` and its function `WordCloud`

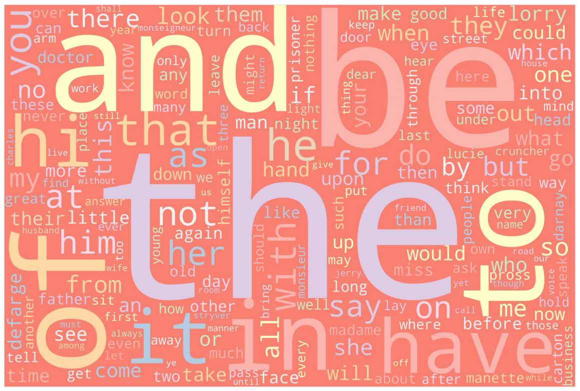
It helps in generating wordclouds from a list of Tags from Text data

```
# Generate word cloud
wordcloud = WordCloud(width = 3000, height = 2000, random_state=1,
                      background_color='salmon', colormap='Pastel1',stopwords= [],
                      collocations=False).generate(' '.join(T1))
# Plot
plot_cloud(wordcloud)
```

T1



T2



Inferences

We infer from the above visualizations that

- Words like 'of', 'to' and 'the' are the most frequently used words in T1
- Words like 'of', 'and', 'be' and 'the' are frequently used words in T2
- These words do not contribute to the meaning of the sentence and are mostly useless for us
- These words are known as 'stopwords' and we need to get rid of them

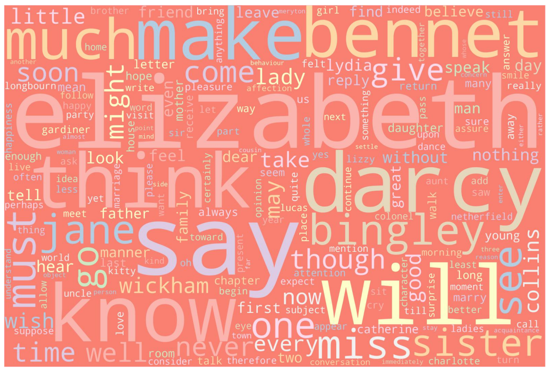
- **Generating new word clouds after removing stopwords**

To remove stopwords, we use an inbuilt function in `nltk` called `STOPWORDS`

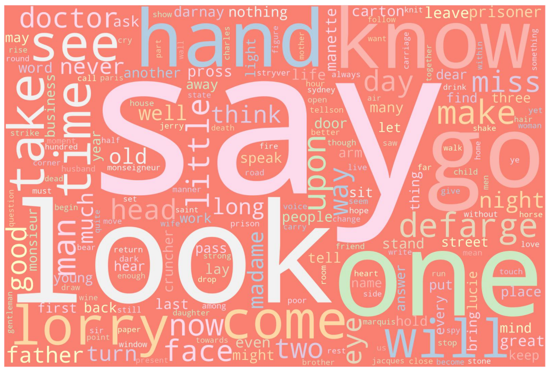
```
def remove_stopwords(tokens):
    return [word for word in tokens if word not in STOPWORDS]
```

```
T1 = remove_stopwords(T1)
T2 = remove_stopwords(T2)
```

T1



T2



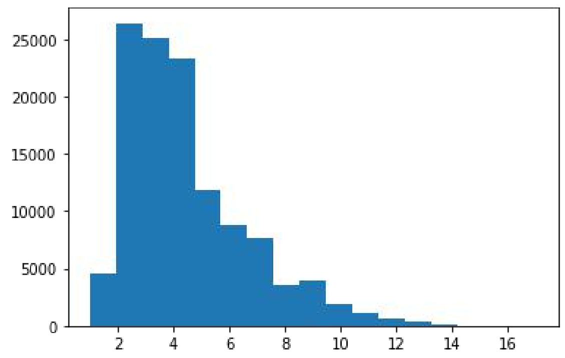
Inferences

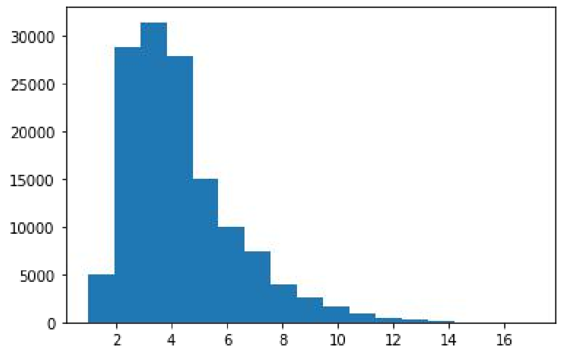
We infer from the above visualizations that

- Now most of the words that were of no meaning to us have been removed.
- New words like ‘elizabeth’, ‘darcy’ and ‘bennet’ now dominate in frequency in T1 which sort of reveals the name of characters of the book around whom the story will revolve.
- Words like ‘say’, ‘look’, ‘one’ and ‘go’ are the new frequently used words in T2, though it is tough to make inferences based on this information but we are able to roughly guess that there is some ‘doctor’ and ‘Lorry’ in the story.
- We have gotten rid of ‘stopwords’ and are now able to draw meaningful conclusions from the data.

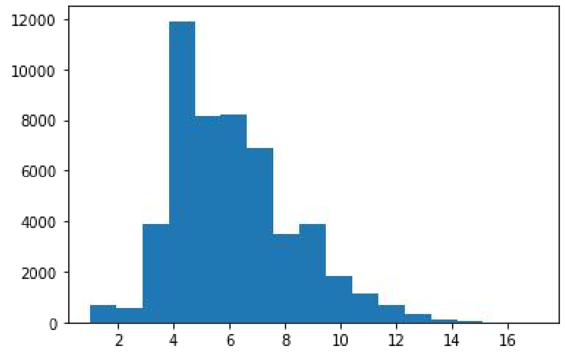
- **As a part of the project, we would also like to evaluate the relationship between the word length and frequency for both T1 and T2 both before and after the removal of stopwords**

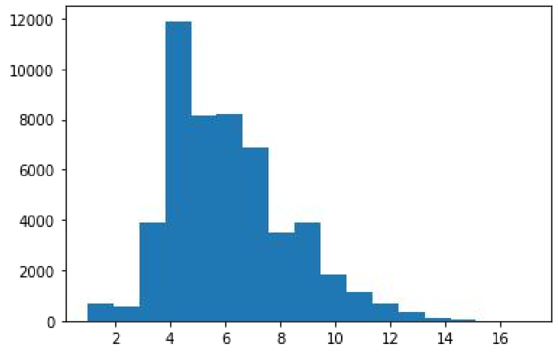
Before





After





Inferences

We infer from the above visualizations that

- The number of words of length 2 and 3 have significantly decreased after the removal of stopwords.
- We can clearly infer that this is due to the removal of stopwords like 'be', 'the', 'of' and 'and' which were the highest occurring words before removal.
- Apart from that there is a general trend that the highest number of words lie in length range 3-6, and there is a significant decrease in frequency of words with either a length lesser than this or more than this.

- ## Performing POS Tagging

  We will now perform the POS Tagging on T1 and T2 using inbuilt functions of `nltk` namely `post_tag()` which uses Penn Treebank tag set to perform POS tagging.

```
def tag_treebank(tokens):
  tagged=nltk.pos_tag(tokens)
  return tagged
```

```
book1_tags=tag_treebank(T1)
book2_tags=tag_treebank(T2)
print(book1_tags)
```

```
('start', 'NN')
('project', 'NN')
('gutenberg', 'NN')
('ebook', 'NN')
('pride', 'NN')
('prejudice', 'NN')
('produce', 'VBP')
('anonymous', 'JJ')
('volunteer', 'NN')
('david', 'NN')
('widger', 'NN')
('illustrate', 'VBP')
('edition', 'NN')
('title', 'NN')
('may', 'MD')
('view', 'VB')
('ebook', 'NN')
('thousand', 'CD')
('six', 'CD')
('hundred', 'VBD')
```

- ## Frequency Distribution of Tags

  Now, we plot the frequency distribution of tags after POS tagging on T1 and T2. For this we take the help of `Counter()` function from `collections` python library and `FreqDist()` function from nltk python library.

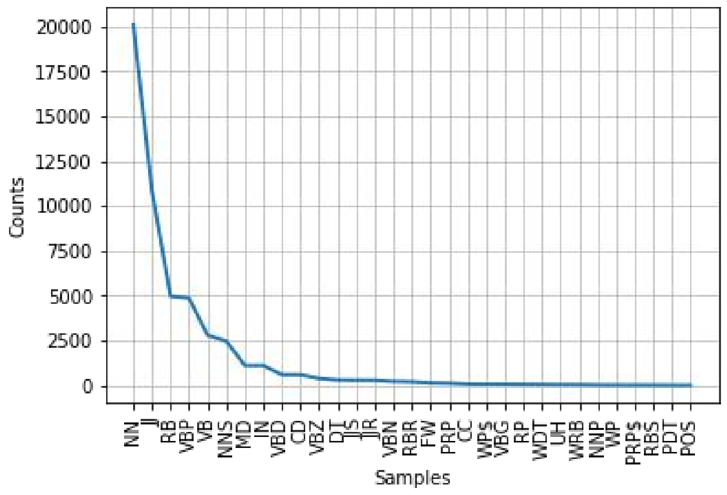
```
def get_counts(tags):
  counts = Counter( tag for word,  tag in tags)
  return counts
```

```
def FrequencyDist(tags):
  wfd=FreqDist(t for (w,t) in tags)
  wfd
  wfd.plot(50)
```

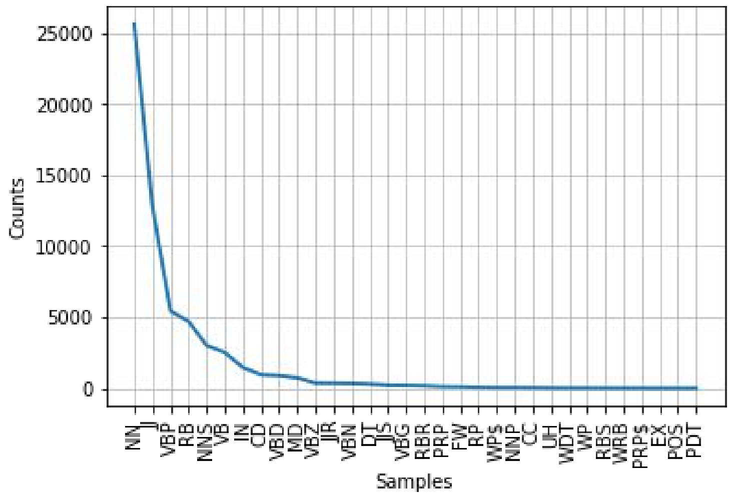
```
book1_pos_count=get_counts(book1_tags)
book2_pos_count=get_counts(book2_tags)
```

```
FrequencyDist(book1_tags)
FrequencyDist(book2_tags)
```

Frequency Distribution of Tags in T1



Frequency Distribution of Tags in T2



Inferences

From the above results we infer that the highest occurring tag is ‘NN’, and ‘Determinant’ Tags are on the lower frequency side. This is largely due to the removal of stopwords before POS Tagging.

Conclusion

We learned how to do Text Preprocessing and Tokenization on Text Data, as well as how to use Plots and Visualizations to answer all of the problems presented to us in Round 1 of the NLP Project, and how to make meaningful inferences from it