https://github.com/Abhinav0505/NLP_PROJECT

Team Name : Tech Kings(A team with single member)

Abhinav Maheshwari – 19ucs169

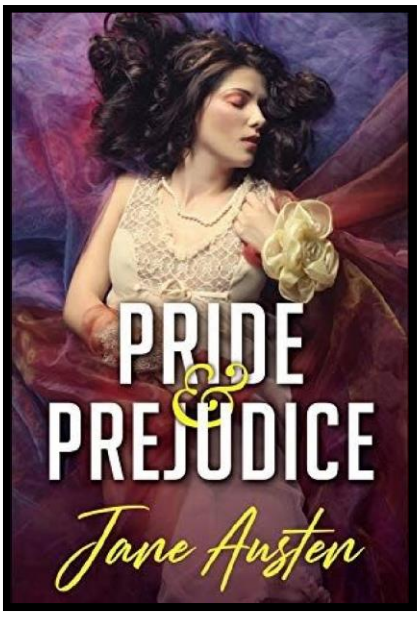
(*Sir submitting report-1 late as was alone in the group and was dealing with family problems,Please kindly consider the request)

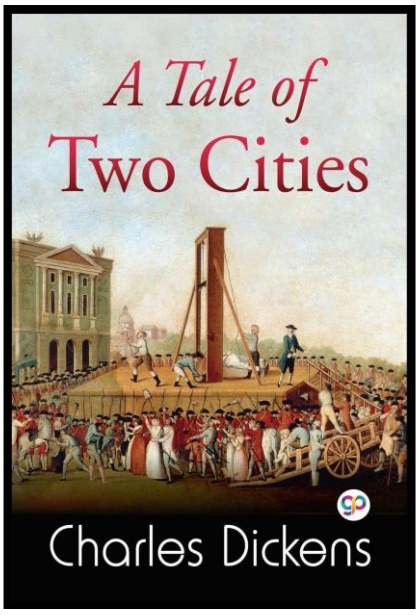
NLP PROJECT ROUND 1

OVERVIEW

In this project, I perform text analysis on two of the given books. “Pride and Prejudice” and “A Tale of Two Cities”. After that I will go apply POS Tagging on both the books. I will do all this using NLP techniques, with the help of python libraries.

BOOKS USED





Pride and Prejudice

By Jane Austen

A Tale of Two Cities

By Charles Dickens

TASKS

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

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

Urllib - Used to fetch text data from Gutenberg URLs

NLTK - Used for Tokenizing, Lemmatization and Removing Stopwords

STEPS

Data Description

Data Preprocessing Steps

1. ***Remove Useless Portion of Book***

   discarded the documentation part of the book that is of no use.

```
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 text to lowercase***

   Converted 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, I use inflect Python Library function p.number_to_words that will give us the english equivalent of a number.

```
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*

Lemmatization with the help of **`WordNetLemmatizer()`** function from **`nltk.stem.`**

```
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

```
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***

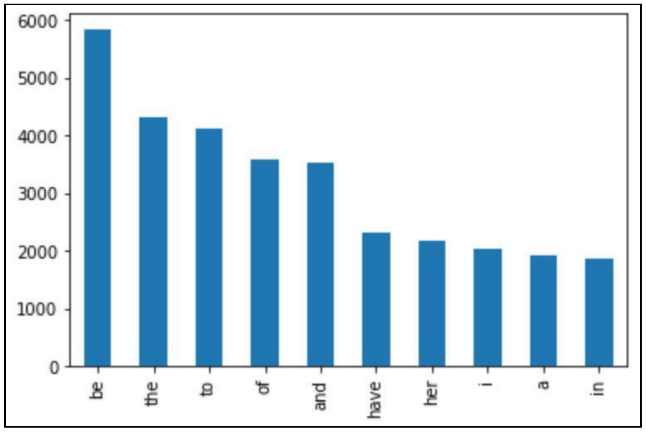
  For this python library used- **pandas**

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

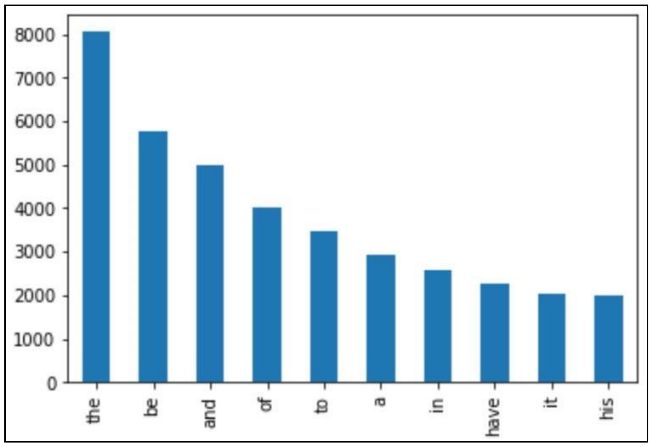
```
T1 = word_tokenize(book1_text)
pd.Series(T1).value_counts()[:10].plot(kind='bar')
```

```
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 take the help of a python library **`wordcloud`** and its function **`WordCloud`**

```
# 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

- 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 are to be removed

-

Generating new word clouds after removing stopwords

To remove stopwords, an inbuilt function in **`nltk`** called **`STOPWORDS is used`**

```
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

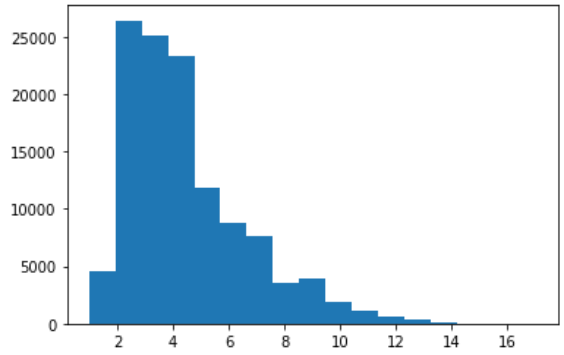
- New words like ‘make’, ‘darcy’ and ‘elizabeth’ now dominate in frequency in T1.
- Words like ‘say’, ‘look’, ‘one’ and ‘go’ are the new frequently used words in T2, though it is tough to make inferences based on this information but we are able to roughly guess that there is some ‘doctor’ and ‘Lorry’ in the story.
- We have gotten rid of ‘stopwords’ and are now able to draw meaningful conclusions from the data.

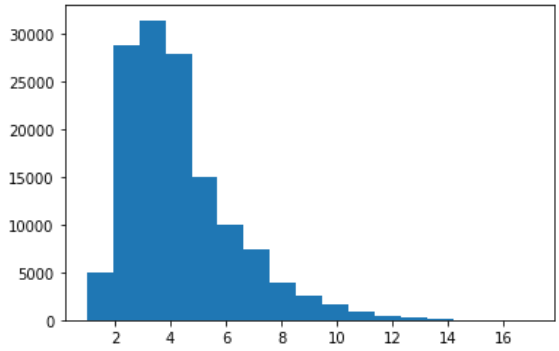
- Meaningless words were removed earlier.

-

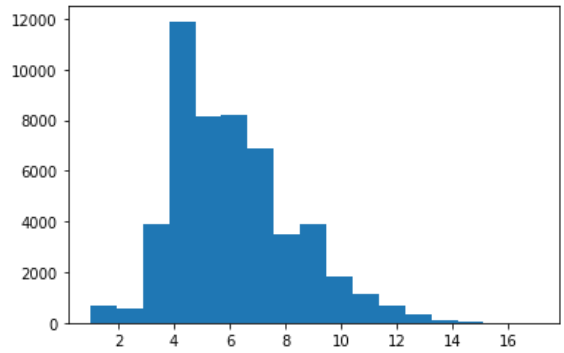
Relationship between words and frequency before and after removal of stopwords.

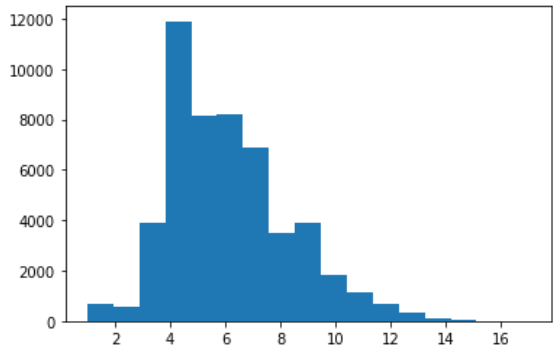
Before





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***

  POS Tagging on T1 and T2 using inbuilt functions of **`nltk`** namely **`post_tag()`**.

```
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***

  Plot of frequency distribution of tags after POS tagging on T1 and T2.

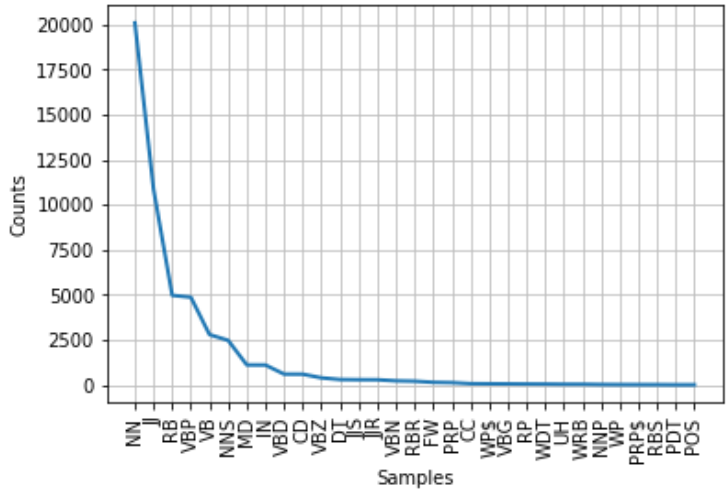
```
def get_counts(tags):
  counts = Counter( tag for word,  tag in tags)
  return counts
```

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

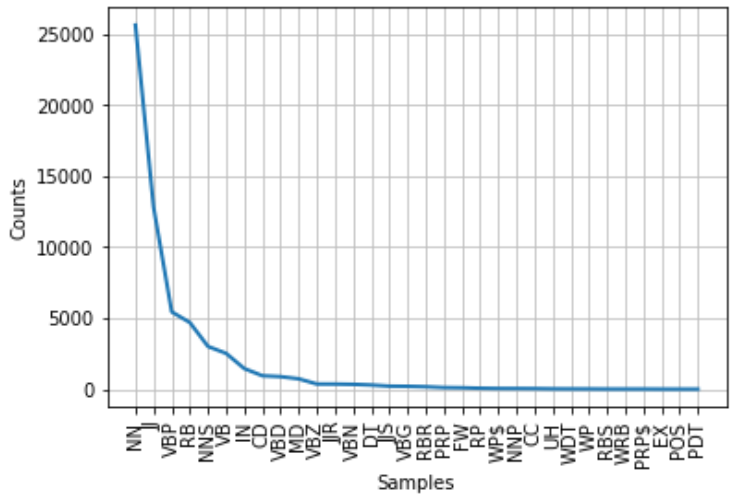
```
book1_pos_count=get_counts(book1_tags)
book2_pos_count=get_counts(book2_tags)
```

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

Frequency Distribution of Tags in T1



Frequency Distribution of Tags in T2



Conclusion

I have learnt text preprocessing, tokenization on the text and have perform all the task given in project 1 and have drawn meaningful inferences from them.

PROJECT ROUND 2

GOALS

Take two books downloaded from the previous round + one new book to be downloaded from the same link. Let B1, B2 and B3 are the books. For B1 and B2 (after doing the needed pre-processing, done in the previous round): First Part: 1. Find the nouns and verbs in both the novels. Get the immediate categories (parent) that these words fall under in the WordNet. 2. Get the frequency of each category for each noun and verb in their corresponding hierarchies and plot a histogram for the same for each novels. Second Part: 1. Recognise all Persons, Location, Organisation (Types given in Fig 22.1) in book. For this you have to do two steps: (1) First recognise all the entity and then (2) recognise all entity types. Use performance measures to measure the performance of the method used - For evaluation you take a considerable amount of random passages from the Novel, do a manual labelling and then compare your result with it. Present the accuracy with F1 score here. Use B1, B2 and B3 for the following: Third Part: 1. Create TF-IDF vectors for all books and find the cosine similarity between each of them and find which two books are more similar. 2. Do lemmatization of the books and recreate the TF-IDF vectors for all the books and find the cosine similarity of each pair of books.

Libraries used

Python Libraries used in this project :

Urllib - Used to fetch text data from Gutenberg URLs

NLTK - Used for Tokenizing, implementing wordnet, POS tagging etc.

Matplotlib - Used to Visualize our text data

Numpy- To get frequency distributions of nouns and verbs

Typing-To perform evaluation of the Algorithm in Entity Recognition.

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

Spacy- To perform Entity recognition in text

Problem Statement And Inferences:

NOUNS and VERBS Detection

- ***POS Tagging***

POS Tagging is done on T1 and T2 using functions of nltk namely pos_tag() which uses Penn Treebank tag..

Words which are tagged explicitly as nouns and verbs separately from both the novels are extracted using the following functions.

```
def noun(text):
  is_noun = lambda pos: pos[:1] == 'N'
  tokenized = nltk.word_tokenize(text)
  nouns = [word for (word, pos) in nltk.pos_tag(tokenized) if is_noun(pos)]
  return nouns
```

```
noun1=noun(book1_text)
noun2=noun(book2_text)
```

```
def verb(text):
  is_verb = lambda pos: pos[:1] == 'V'
  tokenized = nltk.word_tokenize(text)
  verbs = [word for (word, pos) in nltk.pos_tag(tokenized) if is_verb(pos)]
  return verbs
```

```
verb1=verb(book1_text)
verb2=verb(book2_text)
```

```
print("Number of nouns in book 1 and book 2 respectively are "+ str(len(noun1))+" and "+ str(len(noun2)))
```

```
Number of nouns in book 1 and book 2 respectively are 23981 and 31305
```

```
print("Number of verbs in book 1 and book 2 respectively are "+ str(len(verb1))+" and "+ str(len(verb2)))
```

```
Number of verbs in book 1 and book 2 respectively are 21811 and 22054
```

1. **Get the categories that these words fall under in the WordNet.**

To get the categories of each noun and verb belong to, in the wordnet synsets, nltk.corpus.wordnet is used.The following functions are used to extract and categorize each noun and verb.

```
#gives the categories of nouns or verb that the word belongs to
from nltk.corpus import wordnet as wn
def synset(words):
  categories=[]
  for word in words:
    cat=[]
    for synset in wn.synsets(word):
      if(('noun' in synset.lexname()) & ('Tops' not in synset.lexname()) ):
        cat.append(synset.lexname())
      if('verb' in synset.lexname()):
        cat.append(synset.lexname())
    categories.append(cat)
  return categories
```

the above function to get 2-D lists which contains the categories that each noun and verb have been defined in the wordnet database.

```
noun_syn1=synset(noun1)
noun_syn2=synset(noun2)
verb_syn1=synset(verb1)
verb_syn2=synset(verb2)
```

Hence noun_syn1, noun_syn2, verbsyn_1,verb_syn2 are 2 dimensional lists which contain the categories that noun1,noun2,verb1,verb2 belong to in the wordnet synsets of nouns and verbs.

The 2d lists are indexed as noun_syn1[x][y] where x is the index of corresponding noun in noun1 and y is the index containing the categories it belongs to.

Eg.

```
print(noun1[88])
```

```
neighbourhood
```

```
print(noun_syn1[88][:])
```

```
['noun.location', 'noun.group']
```

Hence a ‘neighbourhood’ is both a place and a group of people.

2. Get the frequency of each category for each noun and verb in their corresponding and plot histogram/bar plots for each corresponding categories.

```
#GIVES TOTAL NOUN LEXNAMES AND TOTAL VERB LEXNAMES FOR FREQUENCY DISTRIBUTIONS
def all_synsets(no,ve):
  nouns=[]
  verbs=[]
  for word in no:
    for synset in wn.synsets(word):
      if(('noun' in synset.lexname()) & ('Tops' not in synset.lexname()) ):
        nouns.append(synset.lexname())
      if('verb' in synset.lexname()):
        verbs.append(synset.lexname())
  for word in ve:
    for synset in wn.synsets(word):
      if(('noun' in synset.lexname()) & ('Tops' not in synset.lexname()) ):
        nouns.append(synset.lexname())
      if('verb' in synset.lexname()):
        verbs.append(synset.lexname())

  return nouns,verbs
```

```
noun_superset1,verb_superset1=all_synsets(noun1,verb1)
noun_superset2,verb_superset2=all_synsets(noun2,verb2)
```

```
print(noun_superset1)
```

```
['noun.event', 'noun.time', 'noun.act', 'noun.act', 'noun.act', 'noun.location',
```

```
len(noun_superset1)
```

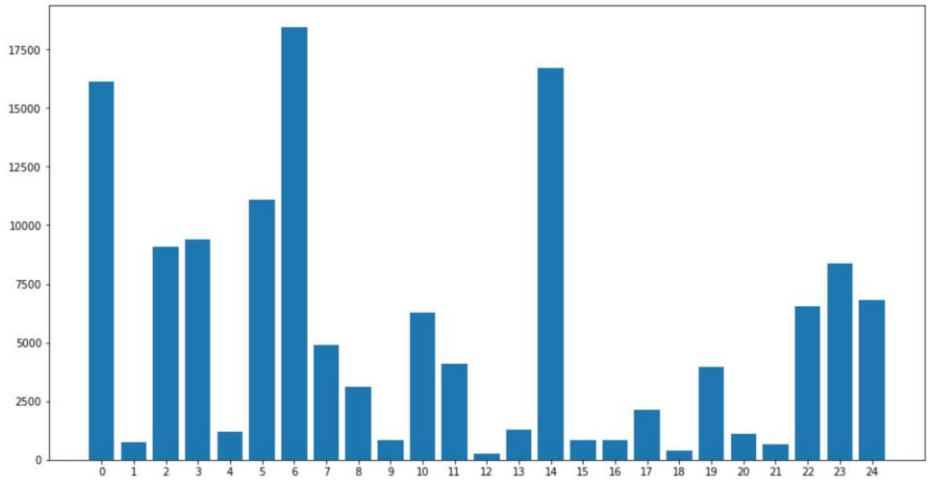
```
135183
```

There are 13.5k elements in the list.

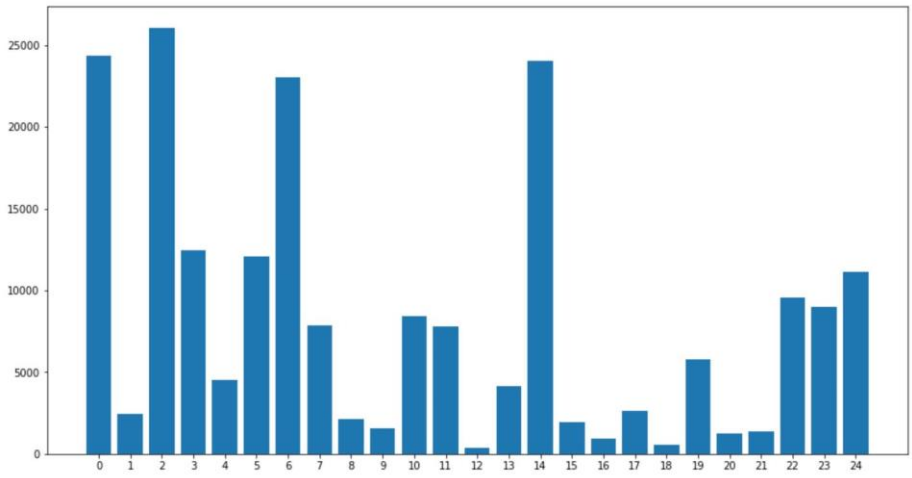
Plotting the histograms:

Numpy is used to count frequency of each category of nouns and verbs obtained from both the novels

```
import numpy as np
labels, counts = np.unique(noun_superset1,return_counts=True)
import matplotlib.pyplot as plt
ticks = range(len(counts))
plt.figure(figsize=(15,8))
plt.bar(ticks,counts, align='center')
plt.xticks(ticks, range(len(labels)))
labels, counts = np.unique(noun_superset2,return_counts=True)
ticks = range(len(counts))
plt.figure(figsize=(15,8))
plt.bar(ticks,counts, align='center')
plt.xticks(ticks, range(len(labels)))
```



For Book 1 Nouns



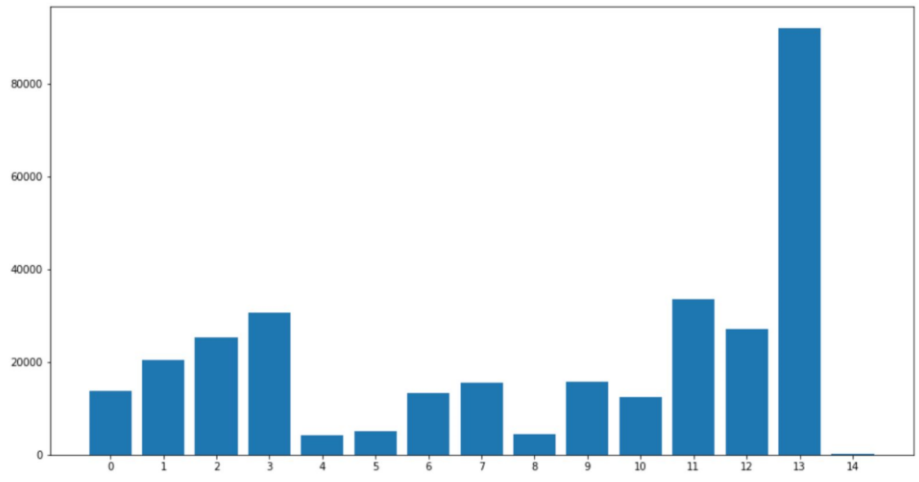
For Book 2 Nouns

The categories are numbered as 0-24 in the order:

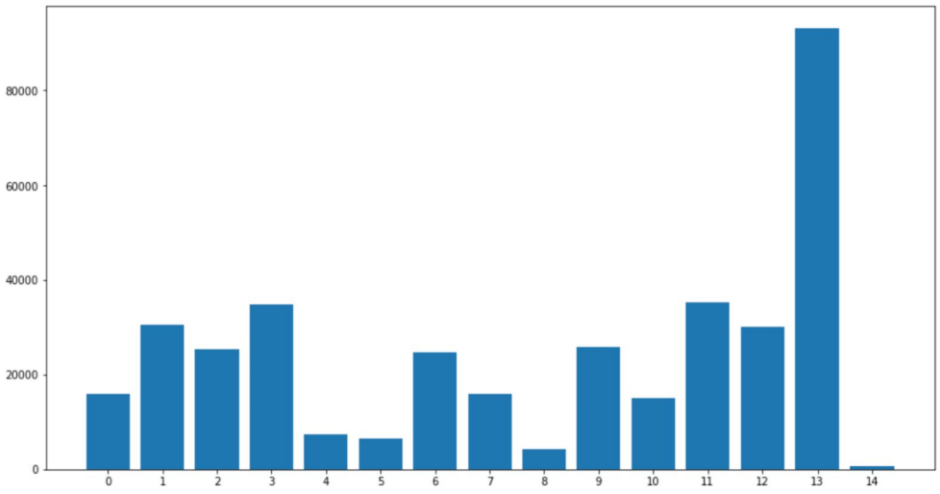
```
print(labels)
```

```
['noun.act' 'noun.animal' 'noun.artifact' 'noun.attribute' 'noun.body'
 'noun.cognition' 'noun.communication' 'noun.event' 'noun.feeling'
 'noun.food' 'noun.group' 'noun.location' 'noun.motive' 'noun.object'
 'noun.person' 'noun.phenomenon' 'noun.plant' 'noun.possession'
 'noun.process' 'noun.quantity' 'noun.relation' 'noun.shape' 'noun.state'
 'noun.substance' 'noun.time']
```

Plots for Verbs:



For Verbs in Book 1



For Verbs in Book 2

And the labels are numbered as 0-14 in the following order.

```
['verb.body' 'verb.change' 'verb.cognition' 'verb.communication'
 'verb.competition' 'verb.consumption' 'verb.contact' 'verb.creation'
 'verb.emotion' 'verb.motion' 'verb.perception' 'verb.possession'
 'verb.social' 'verb.stative' 'verb.weather']
```

Named Entity Recognition

1. Get the entities involved in each of the novels.

Spacy is used to perform entity recognition.

TYPE	DESCRIPTION
PERSON	People, including fictional.
NORP	Nationalities or religious or political groups.
FAC	Buildings, airports, highways, bridges, etc.
ORG	Companies, agencies, institutions, etc.
GPE	Countries, cities, states.
LOC	Non-GPE locations, mountain ranges, bodies of water.
PRODUCT	Objects, vehicles, foods, etc. (Not services.)
EVENT	Named hurricanes, battles, wars, sports events, etc.
WORK_OF_ART	Titles of books, songs, etc.
LAW	Named documents made into laws.
LANGUAGE	Any named language.
DATE	Absolute or relative dates or periods.
TIME	Times smaller than a day.
PERCENT	Percentage, including "%".
MONEY	Monetary values, including unit.
QUANTITY	Measurements, as of weight or distance.
ORDINAL	"first", "second", etc.
CARDINAL	Numerals that do not fall under another type.

The Spacy uses token-level entity annotation using the BILUO tagging scheme to describe the entity boundaries. Where,

TAG	DESCRIPTION
`B EGIN`	The first token of a multi-token entity.
`I N`	An inner token of a multi-token entity.
`L AST`	The final token of a multi-token entity.
`U NIT`	A single-token entity.
`O UT`	A non-entity token.

First we import spacy and get the entities involved in both the novels using the methods described in the library.

```
import spacy
from spacy import displacy
from collections import Counter
import en_core_web_sm
nlp = en_core_web_sm.load()
doc1 = nlp(book1_text)
doc2 = nlp(book2_text)
print("there are total "+str(len(doc1.ents))+" entities in book 1 and "+str(len(doc2.ents))+" in book 2")
```

```
there are total 3322 entities in book 1 and 2688 in book 2
```

```
print([(X, X.ent_iob_) for X in doc1])
```

```
[(start, 'O'), (of, 'O'), (this, 'O'), (project, 'O'), (gutenberg, 'O'), (ebook, 'O'),
```

```
print([(X, X.ent_iob_) for X in doc2])
```

```
[(start, 'O'), (of, 'O'), (this, 'O'), (project, 'O'), (gutenberg, 'O'), (ebook, 'O'),
```

3.Get the entities which are annotated as Person,Organization and Location by Spacy.

```
def entity_recognition(text):
  doc=nlp(text)
  person=[]
  org=[]
  location=[]
  for X in doc:
    if (X.ent_type_=='PERSON') and X.text not in person:
      person.append(X.text)
    if (X.ent_type_=='ORG')and X.text not in org:
      org.append(X.text)
    if ((X.ent_type_=='LOC') or (X.ent_type_=='GPE')) and X.text not in location:
      location.append(X.text)
  return person,org,location
```

Now collecting these entities from both books:

```
person1,org1,location1=entity_recognition(book1_text)
person2,org2,location2=entity_recognition(book2_text)
print("number of person entities in book 1 and book 2 respectively are "+str(len(person1))+" and "+str(len(person2)))
print("number of organization entities in book 1 and book 2 respectively are "+str(len(org1))+" and "+str(len(org2)))
print("number of location entities in book 1 and book 2 respectively are "+str(len(location1))+" and "+str(len(location2)))
```

```
number of person entities in book 1 and book 2 respectively are 175 and 212
number of organization entities in book 1 and book 2 respectively are 46 and 82
number of location entities in book 1 and book 2 respectively are 37 and 77
```

Each of these lists contain the corresponding entities.

Counting the number of occurrences of names, locations, and organizations in Book 1,

```
X = freq(person1)
print(sorted(X.items(), key = lambda kv:(kv[1], kv[0]),reverse=True))
```

```
[('elizabeth', 623), ('jane', 284), ('bingley', 175), ('s', 138), ('bennet', 109), ('lizzy', 83), ('darcy'
```

```
X = freq(location1)
print(sorted(X.items(), key = lambda kv:(kv[1], kv[0]),reverse=True))
```

```
[('lydia', 136), ('london', 52), ('netherfield', 30), ('lucas', 15), ('kitty', 12), ('us', 8), ('brighton'
```

```
X = freq(org1)
print(sorted(X.items(), key = lambda kv:(kv[1], kv[0]),reverse=True))
```

```
[('wickham', 13), ('house', 12), ('the', 11), ('lambton', 9), ('s', 8), ('party', 5), ('netherfield', 5),
```

Performance of The Entity Recognition model on the dataset:

10 samples of sentences were used to test the entity recognition model.

```
sample_book_1_1='georgiana have the highest opinion
sample_book_1_2='i believe i think only of _you_. ”
sample_book_1_3=' of this marriage to her ladyship l
sample_book_1_4='catherine would not come in again a

hand_labeled_book_1_1=['PER','O','O','O''O','O','O',
hand_labeled_book_1_2=['O','O','O','O','O','O','O','
hand_labeled_book_1_3=['O','O','O','O','O','O','O','
hand_labeled_book_1_4=['PER','O','O','O','O','O','O'
```

:

```
labels_book_1_1=([(X.text) for X in nlp(sample_book_1_1).ents])
labels_book_1_2=([(X.text) for X in nlp(sample_book_1_2).ents])
labels_book_1_3=([(X.text) for X in nlp(sample_book_1_3).ents])
labels_book_1_4=([(X.text) for X in nlp(sample_book_1_4).ents])
```

To Calculate the performance metric precision,recall and the f1 scores of the predictions are used.

To Calculate these metrics, a confusion matrix is created and then calculated the above scores from it:

```
from typing import List, Dict, Sequence

class Matrics:
    def __init__(self, sents_true_labels: Sequence[Sequence[Dict]], sents_pred_labels:Sequence[Sequence[Dict]]):
        self.sents_true_labels = sents_true_labels
        self.sents_pred_labels = sents_pred_labels
        self.types = set(entity['type'] for sent in sents_true_labels for entity in sent)
        self.confusion_matrices = {type: {'TP': 0, 'TN': 0, 'FP': 0, 'FN': 0} for type in self.types}
        self.scores = {type: {'p': 0, 'r': 0, 'f1': 0} for type in self.types}

    def cal_confusion_matrices(self) -> Dict[str, Dict]:
        """Calculate confusion matrices for all sentences."""
        for true_labels, pred_labels in zip(self.sents_true_labels, self.sents_pred_labels):
            for true_label in true_labels:
                entity_type = true_label['type']
                prediction_hit_count = 0
                for pred_label in pred_labels:
                    if pred_label['type'] != entity_type:
                        continue
                    if pred_label['start_idx'] == true_label['start_idx'] and pred_label['end_idx'] == true_label['end_idx'] and pred_label['text'] == true_label['text']: # TP
                        self.confusion_matrices[entity_type]['TP'] += 1
                        prediction_hit_count += 1
                    elif ((pred_label['start_idx'] == true_label['start_idx']) or (pred_label['end_idx'] == true_label['end_idx'])) and pred_label['text'] != true_label['text']:
                        self.confusion_matrices[entity_type]['FP'] += 1
                        self.confusion_matrices[entity_type]['FN'] += 1
                        prediction_hit_count += 1
                if prediction_hit_count != 1: # FN, model cannot make a prediction for true_label
                    self.confusion_matrices[entity_type]['FN'] += 1
                prediction_hit_count = 0 # reset to default
```

```
def cal_scores(self) -> Dict[str, Dict]:
    """Calculate precision, recall, f1."""
    confusion_matrices = self.confusion_matrices
    scores = {type: {'p': 0, 'r': 0, 'f1': 0} for type in self.types}

    for entity_type, confusion_matrix in confusion_matrices.items():
        if confusion_matrix['TP'] == 0 and confusion_matrix['FP'] == 0:
            scores[entity_type]['p'] = 0
        else:
            scores[entity_type]['p'] = confusion_matrix['TP'] / (confusion_matrix['TP'] + confusion_matrix['FP'])

        if confusion_matrix['TP'] == 0 and confusion_matrix['FN'] == 0:
            scores[entity_type]['r'] = 0
        else:
            scores[entity_type]['r'] = confusion_matrix['TP'] / (confusion_matrix['TP'] + confusion_matrix['FN'])

        if scores[entity_type]['p'] == 0 or scores[entity_type]['r'] == 0:
            scores[entity_type]['f1'] = 0
        else:
            scores[entity_type]['f1'] = 2*scores[entity_type]['p']*scores[entity_type]['r'] / (scores[entity_type]['p']+scores[entity_type]['r'])
    self.scores = scores
```

Using the above code the scores are calculated straight from the predicted entities and true entities.

```
for x in range(len(labels_book_1_1)):
  if (labels_book_1_1[x]=='') or (not in ['ORG','LOC','GPE','PER']) :
    labels_book_1_1[x]='O'
for x in range(len(labels_book_1_2)):
  if (labels_book_1_2[x]=='') or (not in ['ORG','LOC','GPE','PER']):
    labels_book_1_2[x]='O'
for x in range(len(labels_book_1_3)):
  if (labels_book_1_3[x]=='') or (not in ['ORG','LOC','GPE','PER']):
    labels_book_1_3[x]='O'
for x in range(len(labels_book_1_4)):
  if (labels_book_1_4[x]=='') or (not in ['ORG','LOC','GPE','PER']):
    labels_book_1_4[x]='O'
```

After evaluating the labels in all of the samples from book 1:

	precision	recall	f1-score	support
PER	0.77	0.75	0.76	72
ORG	0.25	0.25	0.25	4
LOC	0.78	0.80	0.79	12
GPE	0.82	0.87	0.84	25
O	0.96	0.94	0.99	554
avg/total	0.66	0.72	0.69	667

Performing the above steps for book 2:

	precision	recall	f1-score	support
PER	0.73	0.75	0.76	67
ORG	0.5	0.5	0.5	6
LOC	0.66	0.72	0.69	20
GPE	0.82	0.87	0.85	17
O	0.94	0.96	0.95	474
avg/total	0.75	0.76	0.76	584

Hence we get average f1 score of 0.69 in first book and 0.76 in second book after evaluation.

Relationship Extraction

To perform Relation Extraction , **`extract_rels`** method of **`nltk.sem.relextract`** was used.

Once named entities have been identified in a text, then want to extract the relations that exist between them. As indicated earlier,typically be looking for relations between specified types of named entity. One way of approaching this task is to initially look for all triples of the form (*X*, α, *Y*), where *X* and *Y* are named entities of the required types, and α is the string of words that intervenes between *X* and *Y*. We can then use regular expressions to pull out just those instances of α that express the relation that we are looking for.

```
import re
from nltk import ne_chunk, pos_tag, word_tokenize
from nltk.sem.relextract import extract_rels, rtuple
nltk.download('maxent_ne_chunker')
nltk.download('words')
```

1. Person - Location Relationship Extraction

   We are able to extract 5 person-location relationships using the following regular expression

```
text = BookText(url1)

BELONG = re.compile(r'.*\bin|from|belonged|lived\b.*')

sentences = nltk.sent_tokenize(text)
tokenized_sentences = [nltk.word_tokenize(sentence) for sentence in sentences]
tagged_sentences = [nltk.pos_tag(sentence) for sentence in tokenized_sentences]

for i,sent in enumerate(tagged_sentences):
  sent = ne_chunk(sent)
  rels = extract_rels('PER', 'GPE', sent, corpus = 'ace', pattern = BELONG, window = 10)
  for rel in rels:
    print(rtuple(rel))
```

```
[PER: 'elizabeth/NNP'] 'lived/VBN in/IN' [GPE: 'london/NNP']
[PER: 'jane/NNP'] 'lived/VBN near/IN' [GPE: 'neitherfield/NNP']
[PER: 'bingley/NNP'] 'is/VBZ from/IN' [GPE: 'scotland/NNP']
[PER: 'elizabeth/NNP'] 'belonged/VBD to/IN' [GPE: 'london/NNP']
[PER: 'jane/NNP'] 'was/VBD now/RB in/IN' [GPE: 'brighton/NNP']
```

2. Person-Person Relationship Extraction (Relationship b/w characters)

   We are able to extract 4 person-person relationships using the following regular expression

```
RELATIONS = re.compile(r'.*\bmother|father|sister|brother|aunt|uncle\b.*')

for i,sent in enumerate(tagged_sentences):
  sent = ne_chunk(sent)
  rels = extract_rels('PER', 'PER', sent, corpus = 'ace', pattern = RELATIONS, window = 10)
  for rel in rels:
    print(rtuple(rel))
```

```
[PER: 'elizabeth/NNP'] 'mother/NN' [PER: 'marie/NNP']
[PER: 'jane/NNP'] 'lived/VBN with/IN her/PRP$ father/NN' [PER: 'wickham/NNP']
[PER: 'elizabeth/NNP'] 'spent/NN most/JJS of/IN her/PRP$ childhood/NN with/IN aunt/NN' [PER: 'lily/NNP']
[PER: 'wickham/NNP'] 'was/VBD often/RB beaten/VBN by/IN uncle/NN' [PER: 'lucas/NNP']
```

3. Person-Organization Relationship

   We are able to extract 3 organization relationships using the following regular expression

```
ORG = re.compile(r'.*\bwork|of|in\b.*')

for i,sent in enumerate(tagged_sentences):
  sent = ne_chunk(sent)
  rels = extract_rels('PER', 'ORG', sent, corpus = 'ace', pattern = ORG, window = 10)
  for rel in rels:
    print(rtuple(rel))
```

```
[PER: 'jane/NNP'] 'became/VBD the/DT ceo/NN of/IN' [ORG: 'clapham/NNP']
[PER: 'bingley/NNP'] 'loved/VBD the/DT work/NN of/IN' [ORG: 'phillips/NNP']
[PER: 'bennet/NNP'] 'was/VBD involved/VBN in/IN' [ORG: 'lambton/NNP']
```

Inferences :

- Elizabeth lived in London at some point of time
- Jane lived near netherfield at some point of time
- Jane went to Brighton in the story plot
- Wickham was Jane's Father
- Lucas was Wickham's Uncle
- Lily was Elizabeth's Aunt, and Elizabeth spent most of her childhood with her
- Jane was the CEO of Clapham
- Mrs. Bennet had some role in Lambton

Conclusion

Learnt Semantic analysis, POS-Tagging, Named Entity Recognition,
Performance evaluation and Extracting Entity relationships from raw text data
and have performed all the tasks given in project report 2.