

NLP project Round1 report

Submitted by

Members of Team Language Revolution

Jatin Dahiya 19ucs033

Rishabh Sahu 19ucs129

Naman Dhanotia 19ucs128

Link to GitHub code repository

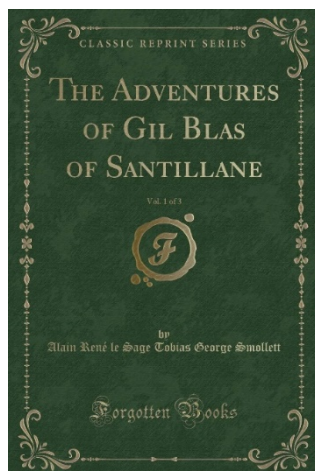
[Rishabhsahu325/NLP_Project_Round1: NLP project \(github.com\)](https://github.com/Rishabhsahu325/NLP_Project_Round1)

Problem Description

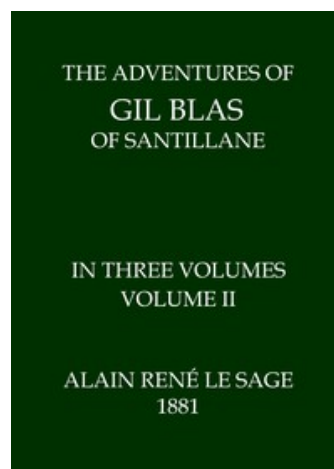
To take two books from <http://gutenberg.org> in .txt format and perform the following Natural Language processing operations on them

- Apply Data pre-processing on the text
- Generating frequency distributions of the words
- Creating word clouds from the text before and after removing stop words
- Evaluating relationship between word length and frequency
- Parts of Speech tagging for the words in the text

Books chosen for applying the processing



T1: The Adventures of Gil Blas of Santillane, Volume I (of 3)



T2: The Adventures of Gil Blas of Santillane, Volume II (of 3)

Python Libraries/Modules used

Matplotlib	: for drawing plots
Python re library (regular expressions library)	: For regular expressions
NumPy	:for parameters of axes while plotting graphs
Nltk:	:Used for tokenizing, removing stop words
Math	:For calculating floor and ceil function values while plotting values
WordCloud	:For creating word cloud
Collections	:For getting the frequency mappings of the POS tags

Inferences after examining raw data

This raw text contains copyright related information, chapter headings, random blank lines and unprocessed text which cannot be directly processed.

Data Pre-processing and Preparation steps

We performed the following data pre-processing steps

1. Removing chapter number and chapter Headings
2. Removing all punctuation marks
3. Changing all text to lowercase
4. Converting short forms like can't to actual representations
5. Tokenising the text into a list of words
6. Removing chapter headings and unrelated data
7. Removing hyperlinks

Preprocessing

```
In [5]: # remove useless text from the data
def remove_useless_text(text):
    text = re.sub(r'\n CHAPTER *[A-Z]*[A-Z]._| \n CHAPTER *[A-Z]*[A-Z].| \n ([\s\S]*) _| \n ([\s\S]*) \)*\*\*\* START OF THE PROJECT ([\s\S]*)\)*\*\*\*', ' ', text)
    text = re.sub(r'\n\*\*\* END OF THE PROJECT ([\s\S]*)', ' ', text)
    return text
```

```
In [6]: # reading text from the book and told
with open('pg66677.txt', 'r', encoding='utf-8') as f:
    text_book1 = ''.join(f.readlines())

with open('pg66678.txt', 'r', encoding='utf-8') as f:
    text_book2 = ''.join(f.readlines())
```

```
In [7]: text_book1 = remove_useless_text(text_book1)
text_book2 = remove_useless_text(text_book2)
```

```
In [8]: # converting all text to lower case and removing any link
def to_lower(text):
    text = text.lower()
    re.sub(r"http\S+", "", text)
    return text
```

```
In [9]: text_book1 = to_lower(text_book1)
text_book2 = to_lower(text_book2)
```

```
In [10]: # converting short forms to full forms
def conversion(text):
    text = re.sub(r"won't", "will not", text)
    text = re.sub(r"can't", "can not", text)
    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
```

```
In [11]: text_book1 = conversion(text_book1)
text_book2 = conversion(text_book2)
```

```
In [12]: # removing all the punctuations from the text
def remove_punctuations(text):
    text = re.sub(r"[^a-zA-Z0-9]", " ", text)
    return text
```

```
In [13]: text_book1 = remove_punctuations(text_book1)
text_book2 = remove_punctuations(text_book2)
```

Tokenization

```
In [14]: from nltk.tokenize import word_tokenize
nltk.download('punkt')

# splitting text into words
def tokenize_word(text):
    words = word_tokenize(text)
    return words

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Unzipping tokenizers/punkt.zip.
```

```
In [15]: words_book1 = tokenize_word(text_book1)
words_book2 = tokenize_word(text_book2)
```

Frequency analysis

```
In [16]: # analyzing the frequency of words
data_analysis_book1 = nltk.FreqDist(words_book1)
data_analysis_book1.plot(25, cumulative=False)
data_analysis_book2 = nltk.FreqDist(words_book2)
data_analysis_book2.plot(25, cumulative=False)
```

Word Cloud

```
In [17]: def listToString(s):
          str1 = " "
          return (str1.join(s))

          # converting list to string
          def list_to_string(words):
              text = listToString(words)
              return text

In [18]: text_book1 = list_to_string(words_book1)
          text_book2 = list_to_string(words_book2)

In [19]: # creating word cloud for book 1
          wc_1 = WordCloud(background_color="white", width=1000, height=1000, random_state=1, stopwords= [], collocations=False).generate(text_book1)
          plt.imshow(wc_1)
```

Removing stop words

StopWords

```
In [21]: from nltk.corpus import stopwords
          nltk.download('stopwords')

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Unzipping corpora/stopwords.zip.

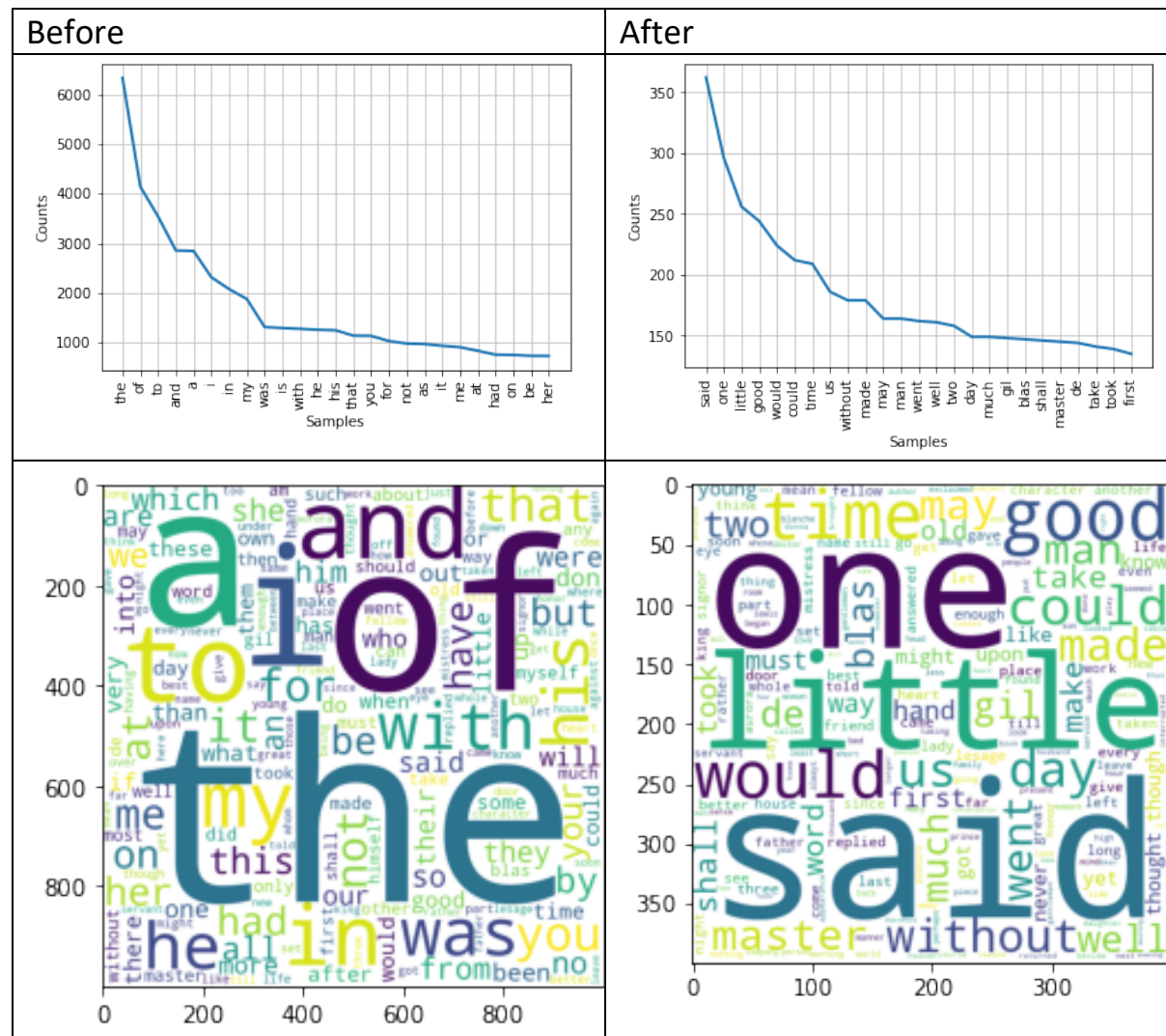
Out[21]: True

In [22]: # removing the stopwords
          def remove_stopwords(words):
              words = [w for w in words if w not in stopwords.words("english")]
              return words

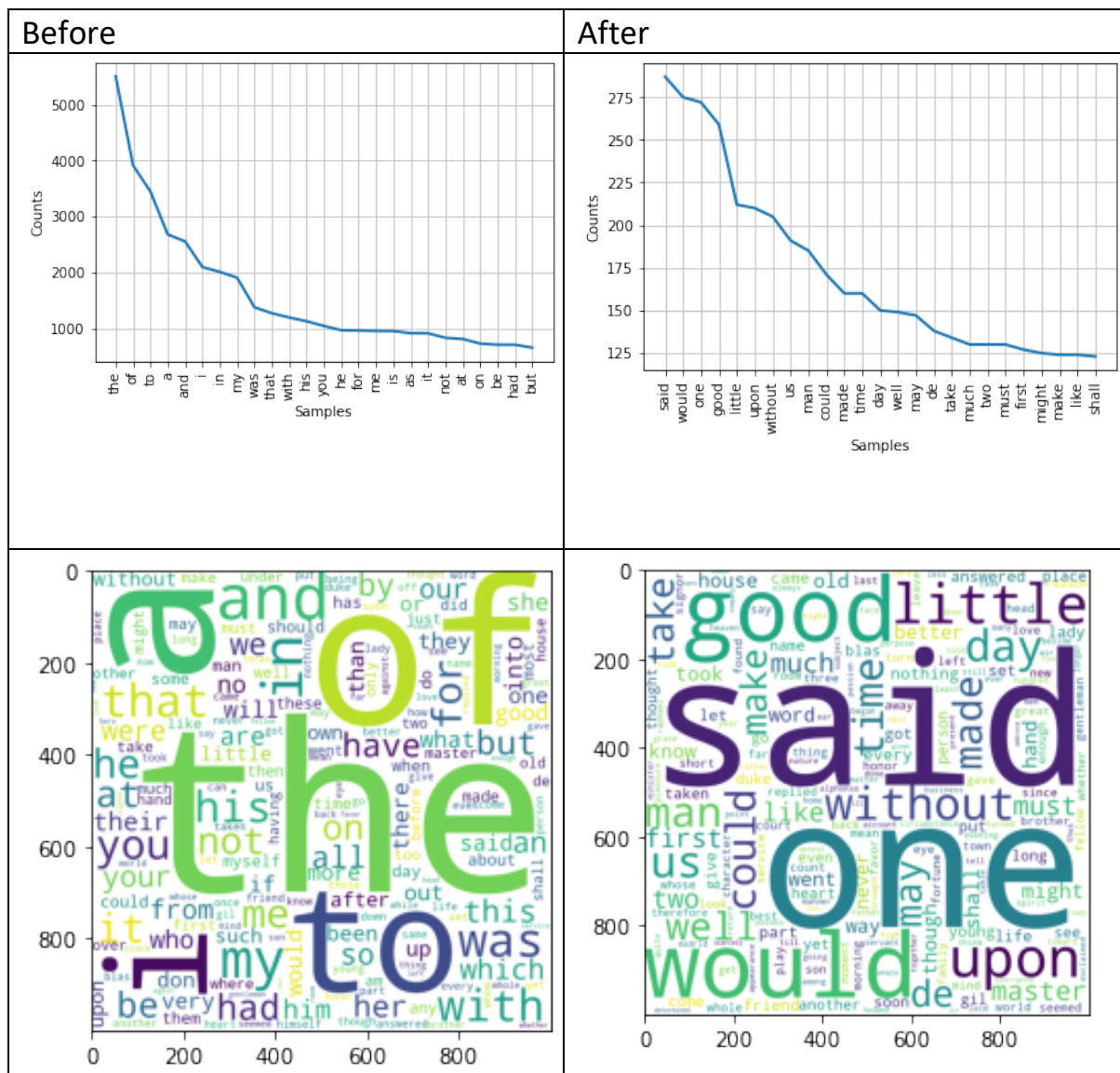
In [23]: words_book1 = remove_stopwords(words_book1)
          words_book2 = remove_stopwords(words_book2)
```

Illustrations (Word clouds and word wise frequency plots)

T1



T2



Inference from word Clouds

- The word clouds before and after removing stop words are quite different due to the high frequency of many of these stop words. One of the reasons may be that stop words can be used in a variety of contexts whereas nouns and verbs are more restricted to the situations to which they relate to.
- After removing stop words, we are able to find the set of words which provide us meaning and context about the document.

Word length – frequency

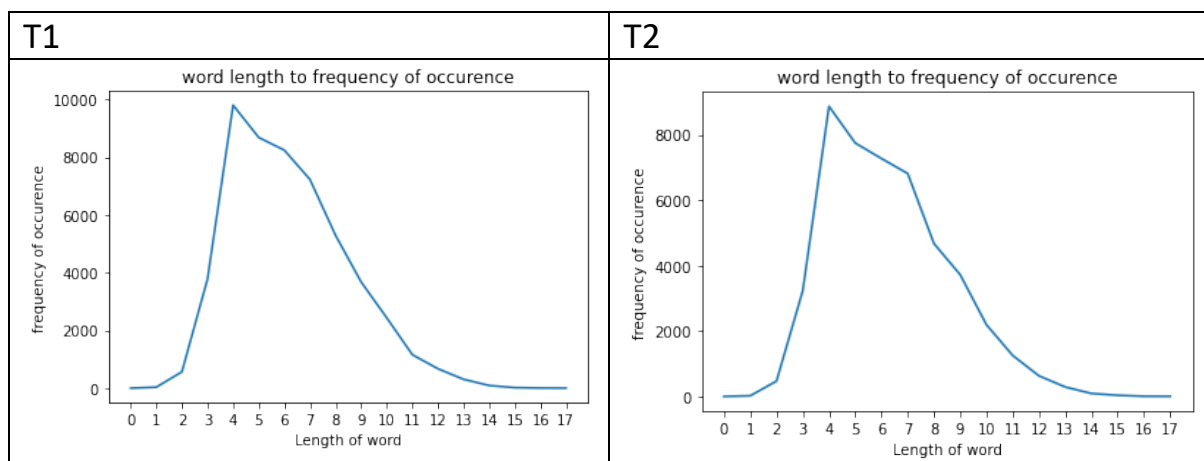
Here we are calculating the word length and their frequency of occurrence.

wordlength to frequency calculation

```
In [28]: def wordlength_to_frequency(words):
frequencyarr = []
lengtharr=[]
frequencyarr.clear()
lengtharr.clear()
y=0
x=0
# finding the largest word in list
res = max(words, key = len)
#print(res)
i=0
fre=0
#print(len(res))
# calculating the frequency of words with different length
while(i<=len(res)):
    for word in words:
        if(len(word)==i):
            fre= fre+1
        print(i,"-",fre)
        frequencyarr.append(fre)
        lengtharr.append(i)
        fre=0
        i=i+1

# plotting the line graph using the values calculated
y = np.array(frequencyarr)
x = np.array(lengtharr)
plt.plot(x, y)
new_list = range(math.floor(min(x)), math.ceil(max(x))+1)
plt.xticks(new_list)
plt.xlabel("Length of word")
plt.ylabel("frequency of occurrence")
plt.title("word length to frequency of occurrence")
plt.show()
```

Illustration: Word length – frequency plots



Inferences from word length- frequency plot

- For both the books Words having length between 3 to 5 are the most frequently occurring words in these books. After those words with larger lengths (up to a certain length) are frequent followed by words of length

1 to 2. Very long words appear very rarely. Overall implying that most of the words lie in the length range of 3 to 5.

POS tagging

Here we are finding the tag associated with each word that was pre-processed.

```
In [32]: # POS tagging the words
result1=nlk.pos_tag(words_book1)

In [33]: result2=nlk.pos_tag(words_book2)

In [34]: from collections import Counter
def get_counts(tags):
    counts = Counter( tag for word, tag in tags)
    return counts

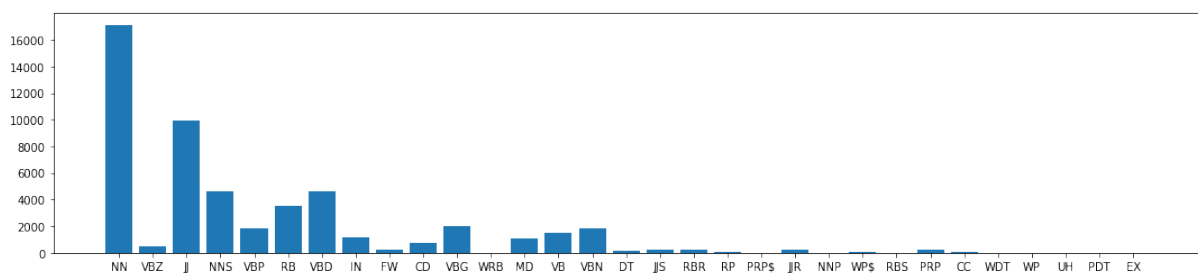
In [69]: def FrequencyPlot(distribution):
    #plt.rcParams["figure.autolayout"] = True
    plt.rcParams["figure.figsize"] = [15, 3.50]
    plt.bar(distribution.keys(),distribution.values())
    plt.show()

In [71]: distribution1=get_counts(result1)
print("Number of tags used in T1=",len(distribution1))
distribution1
FrequencyPlot(distribution1)

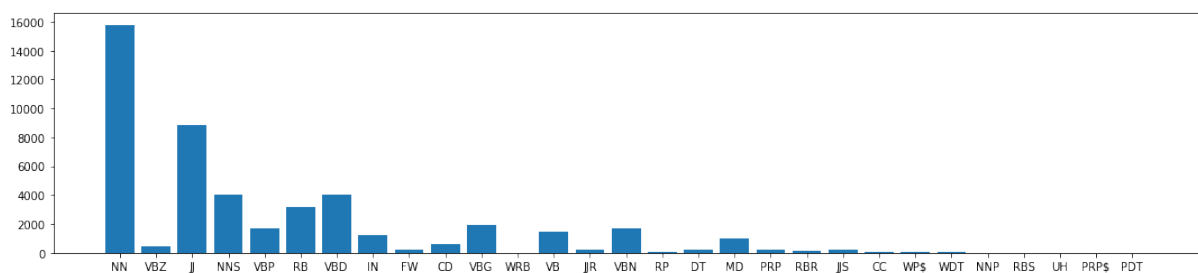
In [72]: distribution2=get_counts(result2)
print("Number of tags used in T2=",len(distribution2))
distribution2
FrequencyPlot(distribution2)
```

Illustrations POS tagging

Book T1



Book T2



Inferences POS_tagging

We applied pos_tagging on the two books using pos_tag function. The pos_tag(words) function uses the Penn treebank as the default tag set as per official documentation.

In T1 the most frequently occurring POS Tag is 'NN' with count 17139 followed by 'JJ' having count 9972.

In T2 the most frequently occurring POS Tag is 'NN' with count 15804 followed by 'JJ ' having count 8871.

Conclusions

In this Round 1 of our project, we performed the tasks of word pre-processing, word tokenisation, Word Cloud generation, POS tagging and also deduced many inferences from them about the books while also learning in the process.

NLP Project Round2 Report

Team Name : Language Revolution

Jatin Dahiya 19ucs033

Rishabh Sahu 19ucs129

Naman Dhanotia 19ucs128

Problem Description

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 novel.

Second Part:

1. Recognise all Persons, Locations, organisations (Types given in Fig 22.1) in the book. For this, you must 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 number of random passages from the Novel, do manual labelling, and then compare your result with it. Present the accuracy with F1 score here.

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.

SPECIFICATIONS:

Python Libraries used in this project:

Urllib - Used to fetch text data from Gutenberg URLs

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

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

Matplotlib - Used to Visualize our text data

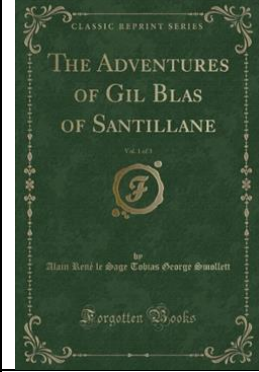
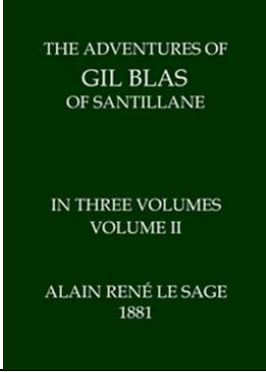
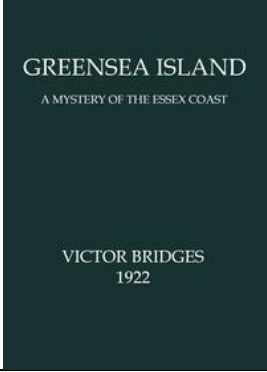
Spacy- To perform Entity recognition in text

NumPy- To get frequency distributions of nouns and verbs
Typing- To perform evaluation of the Algorithm in Entity Recognition.

Problem Statement and Inferences:

NOUNS and VERBS Detection

Books chosen for applying the processing

		
T1: The Adventures of Gil Blas of Santillane, Volume I (of 3)	T2: The Adventures of Gil Blas of Santillane, Volume II (of 3)	T3: Greensea Island A Mystery of the Essex Coast

Python Libraries/Modules used

Matplotlib: for drawing plots

Python re library (regular expressions library): For regular expressions

NumPy: for parameters of axes while plotting graphs

Nltk: Used for tokenizing, removing stop words

Math: For calculating floor and ceil function values while plotting values

WordCloud: For creating word cloud

Collections: For getting the frequency mappings of the POS tags

spacy: For applying named entity recognition

Problem Statement And Inferences:

Code snippets

Importing libraries

importing libraries

```
[ ] import re
import nltk
import math
import numpy as np

import matplotlib.pyplot as plt

from textblob import TextBlob
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from urllib.request import urlopen

import inflect
import pandas as pd

from wordcloud import WordCloud, STOPWORDS
from nltk import FreqDist
from collections import Counter
nltk.download('brown')
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

Preprocessing the text in the books

Preprocessing

```
[ ] # remove useless text from the data
def remove_useless_text(text):
    text = re.sub(r'\n CHAPTER *[A-Z]*[A-Z]._| \nCHAPTER *[A-Z]*[A-Z].| \n_([s\S]*)?_| \n([s\S]*)? \*\* START OF THE PROJECT ([s\S]*)? \*\*', ' ', text)
    text = re.sub(r'\n\*\*\* END OF THE PROJECT ([s\S]*)', ' ', text)
    return text

[ ] # reading text from the book and told
with open('pg66677.txt', 'r', encoding='utf-8') as f:
    text_book1 = ''.join(f.readlines())

with open('pg66678.txt', 'r', encoding='utf-8') as f:
    text_book2 = ''.join(f.readlines())

with open('pg66969.txt', 'r', encoding='utf-8') as f:
    text_book3 = ''.join(f.readlines())

[ ] text_book1 = remove_useless_text(text_book1)
    text_book2= remove_useless_text(text_book2)
    text_book3= remove_useless_text(text_book3)

[ ] # converting all text to lower case and removing any link
def to_lower(text):
    text = text.lower()
    re.sub(r"http\S+", "", text)
    return text

[ ] text_book1 = to_lower(text_book1)
    text_book2=to_lower(text_book2)
    text_book3=to_lower(text_book3)

[ ] #converting short forms to full forms
def conversion(text):
    text = re.sub(r"won't", "will not", text)
    text = re.sub(r"can't", "can not", text)
    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

[ ] text_book1 = conversion(text_book1)
    text_book2= conversion(text_book2)
    text_book3= conversion(text_book3)

[ ] # removing all the punctuations from the text
def remove_punctuations(text):
    text = re.sub(r"[^a-zA-Z0-9]", " ", text)
    return text

[ ] text_book1= remove_punctuations(text_book1)
    text_book2=remove_punctuations(text_book2)
    text_book3=remove_punctuations(text_book3)
```

Part 1

NOUNS and VERBS Detection

● Performing POS Tagging

We will now perform the POS Tagging on T1 and T2 using inbuilt functions of nltk

namely `pos_tag()` which uses Penn Treebank tag set to perform POS tagging.

We will extract the words which are tagged explicitly as nouns and verbs

separately from both the novels using the following functions.

Tokenization

```
[ ] from nltk.tokenize import word_tokenize
    nltk.download('punkt')

    # splitting text into words
    def tokenize_word(text):
        words = word_tokenize(text)
        return words

[ ] words_book1 = tokenize_word(text_book1)
    words_book2 = tokenize_word(text_book2)
```

FINDING NOUNS AND VERBS IN BOTH NOVELS AND THEIR RESPECTIVE WORDNET CATEGORIES

```
▶ nltk.download('averaged_perceptron_tagger')

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

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

noun1=noun(words_book1)
noun2=noun(words_book2)
verb1=verb(words_book1)
verb2=verb(words_book2)

[ ] print("Number of nouns in book 1 and book 2 respectively are "+ str(len(noun1))+ " and "+ str(len(noun2)))
    print("Number of verbs in book 1 and book 2 respectively are "+ str(len(verb1))+ " and "+ str(len(verb2)))
```

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

To retrieve the categories that each noun and verb belong to, in the wordnet synsets, we have used `nltk.corpus.wordnet` as it has all the tools required for this task. We have used the following function to extract categories each noun and verb belongs to. Since a noun

also has synsets interpretations as verbs and vice versa hence we have included them as lists corresponding to its index in the noun and verb lists respectively.

```
[ ] #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

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

[ ] print(noun1[88])

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

Count the total number of noun lexnames and verb lexnames

```
[ ] #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)

[ ] len(noun_superset1)
```

2. Get the frequency of each category for each noun and verb in their corresponding

and plot histogram/bar plots for each corresponding categories.

To get the frequency of all the categories of nouns and verbs in the novels, we have created a set for each novel which contains all the types of nouns and verbs occurring and then plotting the frequency distribution for the data.

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

```
[ ] 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)))
```

```
[ ] print(labels)
```

```
▶ labels, counts = np.unique(verb_superset1,return_counts=True)
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(verb_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)))
```

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

```
[ ] 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)))
```

```
[ ] print(labels)
```

```
[ ] labels, counts = np.unique(verb_superset1,return_counts=True)
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(verb_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)))
```

```
[ ] print(labels)
```

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.

```
[ ] !pip install -U spacy
    !python -m spacy download en_core_web_lg

import spacy
from spacy import displacy
from collections import Counter
import en_core_web_lg

nlp = en_core_web_lg.load()
doc1 = nlp(text_book1)
doc2 = nlp(text_book2)
print("there are total "+str(len(doc1.ents))+ " entities in book 1 and "+str(len(doc2.ents))+ " in book 2")

[ ] print(doc1.similarity(doc2))

[ ] print(doc2.similarity(doc1))

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

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

Part 2

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

We have used the following function on the entities returned by spacy to collect the Person, Organization and Location entities in each of the novels, respectively. The ent_type function returns the type of entity annotated by Spacy and hence return lists containing the above types of entities.

```
[ ] 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

[ ] person1,org1,location1=entity_recognition(text_book1)
    person2,org2,location2=entity_recognition(text_book2)
    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)))

[ ] print(org1)

[ ] def freq(str_list):
    unique_words = set(str_list)
    counts = {}
    for words in unique_words :
        counts[words] = str_list.count(words)
    return counts

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

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

Performance of The Entity Recognition model on the dataset:

To measure the performance of the model on the dataset, we collected 10 random samples/passages

from the novel and labelled them manually for these entities. After Labelling we compared them with the

Labels assigned to them by Spacy's Entity Recognition Algorithm

```
[ ] #Performance Measures

#book1
B1_text1=' Do not you recollect Fabricio,your townsman and schoolfellow?'
B1_text2='I would give all the wine in Valladolid for a pint of water.'

#book2
B2_text1='I gave a left-handed blessing to Euphrasia, and wept over the weakness of Don Gonzales, to be so foolishly infatuated by her.
B2_text2='We were interrupted by the arrival of a courier, charged with a letter for Seraphina from the Count de Polan. '

hand_label_1_1=['O','O','O','O','PER','O','O','O','O']
hand_label_1_2=['O','O','O','O','O','O','O','LOC','O','O','O','O','O']
hand_label_2_1=['O','O','O','O','O','O','PER','O','O','O','O','O','O','PER','O','O','O','O','O','O','O']
hand_label_2_2=['O','O','O','O','O','O','O','O','O','O','O','O','O','O','O','O','PER','O','O','O','O','PER']

#labels generated

labelB1T1=([(word.text) for word in nlp(B1_text1).ents])
labelB1T2=([(word.text) for word in nlp(B1_text2).ents])

labelB2T1=([(word.text) for word in nlp(B2_text1).ents])
labelB2T2=([(word.text) for word in nlp(B2_text2).ents])

from typing import List, Dict, Sequence

class Matrices:
    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}
```

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

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

```

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']:
                    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 and 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

def print_confusion_matrices(self):
    for entity_type, matrix in self.confusion_matrices.items():
        print(f"{entity_type}: {matrix}")

def print_scores(self):
    for entity_type, score in self.scores.items():
        print(f"{entity_type}: f1 {score['f1']:.4f}, precision {score['p']:.4f}, recall {score['r']:.4f}")

```

Using the above code snippet, we will be able to calculate the scores straight from the predicted entities and true entities.

To ensure that we are only computing the PER, ORG, LOC, GPE (Person, organization, and location) entities, we mark other entities as O and proceed with the performance evaluation.


```
[ ] for x in range(len(labelB1T1)):
    if (labelB1T1[x]=='') or (labelB1T1[x] not in ['ORG', 'LOC', 'GPE', 'PER']):
        labelB1T1[x] = 'O'
for x in range(len(labelB1T2)):
    if (labelB1T2[x]=='') or (labelB1T2[x] not in ['ORG', 'LOC', 'GPE', 'PER']):
        labelB1T2[x] = 'O'
for x in range(len(labelB2T1)):
    if (labelB2T1[x]=='') or (labelB2T1[x] not in ['ORG', 'LOC', 'GPE', 'PER']):
        labelB2T1[x] = 'O'
for x in range(len(labelB2T2)):
    if (labelB2T2[x]=='') or (labelB2T2[x] not in ['ORG', 'LOC', 'GPE', 'PER']):
        labelB2T2[x] = 'O'
print(type(hand_label_1_1))
print(type(labelB1T1))
```

```
[ ] matrices1 = Matrices(hand_label_1_1,labelB1T1)
matrices1.cal_confusion_matrices()
matrices1.print_confusion_matrices()
matrices1.cal_scores()
matrices1.print_scores()
```

```
[ ] import spacy
from spacy.gold import GoldParse
from spacy.scorer import Scorer

def evaluate(ner_model, examples):
    scorer = Scorer()
    for input_, annot in examples:
        doc_gold_text = ner_model.make_doc(input_)
        gold = GoldParse(doc_gold_text, entities=annot)
        pred_value = ner_model(input_)
        scorer.score(pred_value, gold)
    return scorer.scores

# example run

#book1
B1_text1='Do not you recollect Fabricio,your townsman and schoolfellow?'
B1_text2='I would give all the wine in Valladolid for a pint of water.'

#book2
B2_text1='I gave a left-handed blessing to Euphrasia, and wept over the weakness of Don Gonzales, to be so foolishly infatuated by her.'
B2_text2='We were interrupted by the arrival of a courier, charged with a letter for Seraphina from the Count de Polan. '

hand_label_1_1=['O','O','O','O','PER','O','O','O','O']
hand_label_1_2=['O','O','O','O','O','O','O','LOC','O','O','O','O','O']
hand_label_2_1=['O','O','O','O','O','O','PER','O','O','O','O','O','O','O','PER','O','O','O','O','O','O','O']
hand_label_2_2=['O','O','O','O','O','O','O','O','O','O','O','O','O','O','O','O','O','PER','O','O','O','PER']
```

Part 3:

Calculating cosine similarity

First created TF-IDF vector using the sklearn library and then calculated cosine similarity between docs based on the vectors we received.

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.

before lemmatization

```
[ ] data = [text_book1,text_book2]
```

```
[ ] import pandas as pd

def create_dataframe(matrix, tokens):

    doc_names = [f'text_book{i+1}' for i, _ in enumerate(matrix)]
    df = pd.DataFrame(data=matrix, index=doc_names, columns=tokens)
    return(df)
```

bold text# cosine

```
[ ] from sklearn.feature_extraction.text import TfidfVectorizer

Tfidf_vect = TfidfVectorizer()
vector_matrix = Tfidf_vect.fit_transform(data)

tokens = Tfidf_vect.get_feature_names()
create_dataframe(vector_matrix.toarray(),tokens)

[ ] from sklearn.metrics.pairwise import cosine_similarity
cosine_similarity_matrix = cosine_similarity(vector_matrix)
create_dataframe(cosine_similarity_matrix,['text_book1','text_book2'])
```

```
[ ] data = [text_book1,text_book3]
```

```
[ ] from sklearn.feature_extraction.text import TfidfVectorizer

Tfidf_vect = TfidfVectorizer()
vector_matrix = Tfidf_vect.fit_transform(data)

tokens = Tfidf_vect.get_feature_names()
create_dataframe(vector_matrix.toarray(),tokens)

[ ] from sklearn.metrics.pairwise import cosine_similarity
cosine_similarity_matrix = cosine_similarity(vector_matrix)
create_dataframe(cosine_similarity_matrix,['text_book1','text_book3'])
```

```
[ ] data = [text_book2,text_book3]
```

```
[ ] from sklearn.feature_extraction.text import TfidfVectorizer

Tfidf_vect = TfidfVectorizer()
vector_matrix = Tfidf_vect.fit_transform(data)

tokens = Tfidf_vect.get_feature_names()
create_dataframe(vector_matrix.toarray(),tokens)
```

```
[ ] from sklearn.metrics.pairwise import cosine_similarity
cosine_similarity_matrix = cosine_similarity(vector_matrix)
create_dataframe(cosine_similarity_matrix,['text_book2','text_book3'])
```

Lemmatizing the words in the book and then calculating TF-IDF vectors and calculating the cosine similarity between the books.

after lemmatization

```
[ ] from nltk.stem import WordNetLemmatizer
    from nltk.corpus import wordnet
    lemmatizer = WordNetLemmatizer()
    def nltk2wn_tag(nltk_tag):
        if nltk_tag.startswith('J'):
            return wordnet.ADJ
        elif nltk_tag.startswith('V'):
            return wordnet.VERB
        elif nltk_tag.startswith('N'):
            return wordnet.NOUN
        elif nltk_tag.startswith('R'):
            return wordnet.ADV
        else:
            return None
    def lemmatize_sentence(sentence):
        nltk_tagged = nltk.pos_tag(nltk.word_tokenize(sentence))
        wn_tagged = map(lambda x: (x[0], nltk2wn_tag(x[1])), nltk_tagged)
        res_words = []
        for word, tag in wn_tagged:
            if tag is None:
                res_words.append(word)
            else:
                res_words.append(lemmatizer.lemmatize(word, tag))
        return " ".join(res_words)

[ ] text_book1 = lemmatize_sentence(text_book1)

[ ] text_book2 = lemmatize_sentence(text_book2)
```



```
[ ] text_book3 = lemmatize_sentence(text_book3)

[ ] data = [text_book1,text_book2]

from sklearn.feature_extraction.text import TfidfVectorizer

Tfidf_vect = TfidfVectorizer()
vector_matrix = Tfidf_vect.fit_transform(data)

tokens = Tfidf_vect.get_feature_names()
create_dataframe(vector_matrix.toarray(),tokens)

[ ] from sklearn.metrics.pairwise import cosine_similarity
cosine_similarity_matrix = cosine_similarity(vector_matrix)
create_dataframe(cosine_similarity_matrix,['doc_1','doc_2'])

[ ] data = [text_book1,text_book3]

from sklearn.feature_extraction.text import TfidfVectorizer

Tfidf_vect = TfidfVectorizer()
vector_matrix = Tfidf_vect.fit_transform(data)

tokens = Tfidf_vect.get_feature_names()
create_dataframe(vector_matrix.toarray(),tokens)

[ ] from sklearn.metrics.pairwise import cosine_similarity
cosine_similarity_matrix = cosine_similarity(vector_matrix)
create_dataframe(cosine_similarity_matrix,['text_book1','text_book3'])

[ ] data = [text_book2,text_book3]

from sklearn.feature_extraction.text import TfidfVectorizer

Tfidf_vect = TfidfVectorizer()
vector_matrix = Tfidf_vect.fit_transform(data)

tokens = Tfidf_vect.get_feature_names()
create_dataframe(vector_matrix.toarray(),tokens)

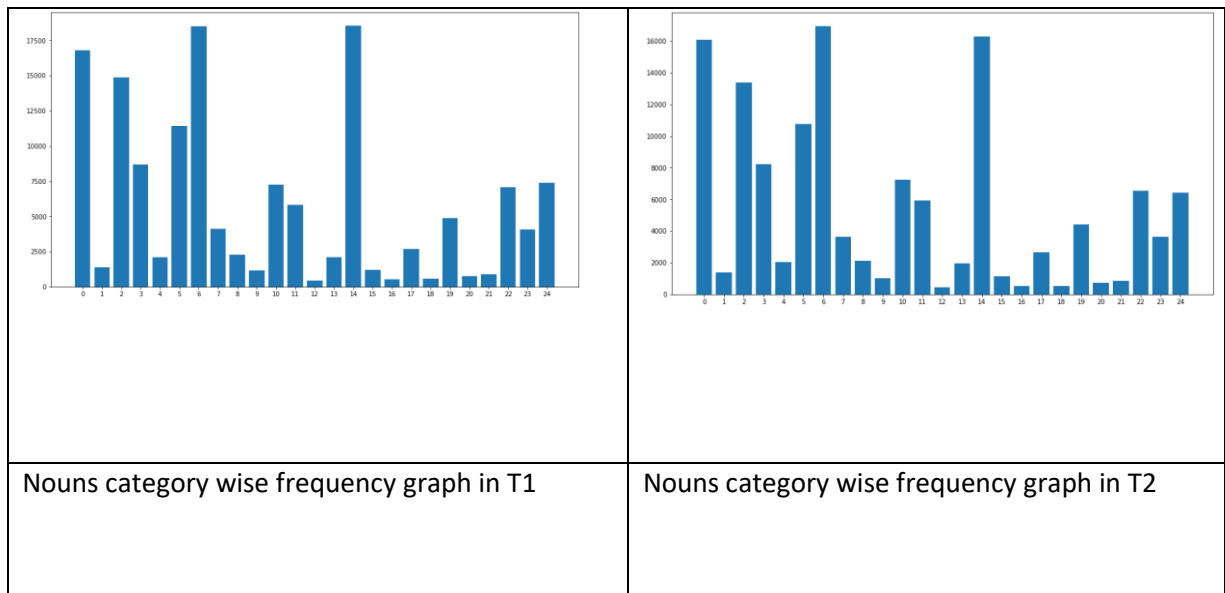
[ ] from sklearn.metrics.pairwise import cosine_similarity
cosine_similarity_matrix = cosine_similarity(vector_matrix)
create_dataframe(cosine_similarity_matrix,['text_book2','text_book3'])
```

Illustrations and results

Part1

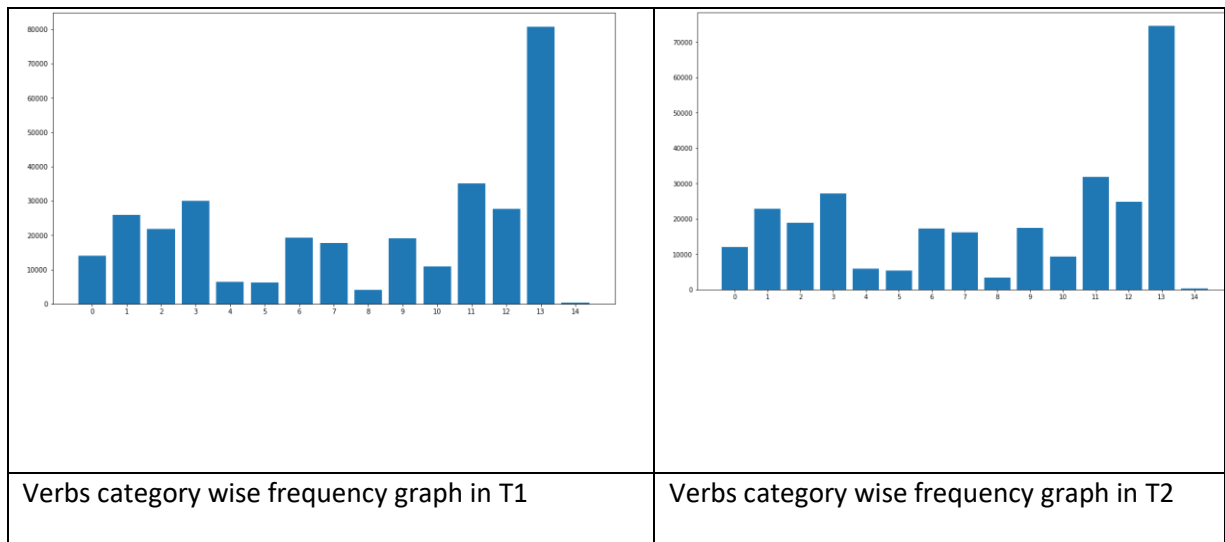
Number of nouns in book 1 and book 2 respectively are 27347 and 25043

Number of verbs in book 1 and book 2 respectively are 21434 and 19430



Noun labels in graph

```
[ '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' ]
```



Verb labels in graph

['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']

Part2

number of person entities in book 1 and book 2 respectively are 272 and 202

number of organization entities in book 1 and book 2 respectively are 117 and 83

number of location entities in book 1 and book 2 respectively are 47 and 55

Some detected ORG entities

['le', 'york', 'castille', 'madrid', 'burgos']

Labels accuracy metrics in T1

	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

Labels and accuracy metrics in T2

	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

Part3

Tf idf values extracted

	000	100	117	14	1500	1668	1686	1692	1694	1695	1707	1708	1710
text_book1	0.000091	0.000000	0.000000	0.000117	0.000091	0.000154	0.000154	0.000154	0.000154	0.000154	0.000154	0.000154	0.000154
text_book2	0.000099	0.000167	0.000000	0.000127	0.000099	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
text_book3	0.000109	0.000000	0.000553	0.000000	0.000109	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

3 rows × 16672 columns

Cosine similarity values

	text_book1	text_book2	text_book3
text_book1	1.000000	0.995679	0.967278
text_book2	0.995679	1.000000	0.964137
text_book3	0.967278	0.964137	1.000000

Thus, books T3 and T1 are more similar as they have higher cosine similarity.

After Lemmatization

Tf-idf vectors T1 AND T2

	117	14	1668	1686	1692	1694	1695	1707	1708	1715	...	youthful	youthful
text_book1	0.000000	0.000122	0.000122	0.000122	0.000122	0.000122	0.000122	0.000122	0.000122	0.000244	...	0.000087	0.000087
text_book2	0.000425	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000101	0.000101

2 rows × 10405 columns

Cosine similarity pair wise T1 AND T2

	text_book1	text_book3
text_book1	1.000000	0.971275
text_book2	0.971275	1.000000

Tf idf vectors T2 AND T3

	000	100	117	14	1500	17th	18	1881	1887	1921	1922	20
text_book1	0.000091	0.000128	0.000000	0.000128	0.000091	0.000000	0.000000	0.000128	0.000091	0.000000	0.000000	0.000091
text_book2	0.000098	0.000000	0.000414	0.000000	0.000098	0.000138	0.000138	0.000000	0.000098	0.000138	0.000138	0.000098

Cosine similarity T2 and T3

	text_book2	text_book3
text_book1	1.000000	0.968446
text_book2	0.968446	1.000000