Project Report

Natural Language Processing

(CSE 3201)

By Team: console.log

Pratham Mehta (20UCS147)

Shrey Parikh (20UCS185)

Shreyash Sharma (20UCS187)

Vatsal Mehta (20UCS225)

GitHub Link: https://github.com/Vatsal32/NLP_Project_Round_1

GitHub Link: https://github.com/Vatsal32/NLP_Project_Round_2

Data Description

The book we have used is:

Software Engineering
Eight Edition By
Ian Sommerville

Number Of pages: 865
Number of words: 13,97,281

Common processing for Part 1 and Part 2

Processing and plotting are done using libraries of Python programming language.
We have used:

- **NLTK** library that is Natural Language Toolkit for text processing.
- **Matplotlib, pyplot and wordcloud** for data visualization.
- **Spacy** for NER Labelling.

Data Preparation

Text Pre-Processing Steps

Text Pre-processing is the first step in the pipeline of Natural Language Processing (NLP). It brings the text into a form that is predictable and analysable for a machine learning algorithm.

The Pre-processing tasks performed in our code are:

1) Manually removed Contents, Glossary, Index, Authors, and References.

2) Removing the footer

a) Footer has 2 lines:

i) It denotes the end of page using 4 dots as '•• ••', we find it using regex and delete it

```
T1 = re.sub ('[ ]*••[ ]*••\n', '', T1)
```

ii) 2nd has a date '4/4/06' mentioned in it, so we find it using regex and delete that line

```
T1 = re.sub ('.* 4/4/06 .*\n', '', T1)
```

3) Removing the header

a) Header is of three types

i) Left page header has the form '<Page No.> Chapter <Chapter No.> ■ <Chapter Name>', we find and delete it using regex as,

```
T1 = re.sub('[0-9]+[ ]+Chapter.*\n', '', T1)
```

ii) Right page header has the form '<section No.> ■ <Section Name> <Page No.>', we use regex to remove it

```
T1 = re.sub('[0-9]+.[0-9]+ ■.*[0-9]+\n', '', T1)
```

iii) 'Key Points' and 'Exercise' sections of the chapters on the right page have the form 'Chapter <Chapter No.> ■ <Key Points>|<Exercises> <Page No.>', we again use regex to remove it

```
T1 = re.sub('Chapter [0-9]+ ■.*[0-9]+\n', '', T1)
```

4) Removing Punctuations and Normalization:

a) For punctuations and other irrelevant symbols, we create a string that contains all such elements. Then, we filter the original text by removing any elements that match anything in our created string.

b) All the heading of sections and sub-sections in a chapter are of no use to us as they do not follow grammatical structure. They have the form '<section No.> <Section Name>' removed using regex as follows

```
T1 = re.sub(r'\d+(\.\d+)*\.\d+.*\n', '', T1)
```

c) Chapter titles can be removed as well. They are of the form

'<Chapter No.>\n<Chapter Name>\n', using regex as follows

```
T1 = re.sub('[0-9]+\n[a-zA-Z ]+\n', '', T1)
```

d) When a word crosses the limit of characters possible in one line the rest is written on the next line denoted by a hyphen on the current line, in the form 'soft-\nware' we join these words by removing '-\n' using regex

```
T1 = re.sub(r'-\n\s*', '', T1)
```

e) For tables we first divide whole text according to the lines, then we divide each line into number of columns based on sentences divided by a group of spaces using regex as

```
temp = re.split(r'\s{2,}', arr[i])
```

f) Then for the lines with more than two columns are ignored as they make up a row in the table. Two columns because the book indentation makes start of each line as a group of spaces.

```
[ ] finalText

'  objectives the objectives of this chapter are to introduce software engineering and to provide a framework for understanding the rest of the book when you have read this chapter
you will understand what software engineering is and why it is important know the answers to key questions that provide an introduction to software engineering understand some
ethical and professional issues that are important for software engineers contents   virtually all countries now depend on complex computerbased systems national infrastructures and
utilities rely on computerbased systems and most electrical prod ucts include a computer and controlling software industrial manufacturing and dis tribution is completely
computerised as is the financial system therefore producing and maintaining software costeffectively is essential for the functioning of national and international economies
software engineering is an engineering discipline whose focus is the cost effective development of highquality software systems software is abstract and intangible it is not
constrained by materials or governed by physical laws or by manufacturing processes in some ways this simplifies software engineering as there are no physical limitations on the
potential of software however this lack of nat ural constraints means that software can easily become extremely complex and hence very difficult to understand the notion of software
engineering was first proposed in 1968 at a conference held to discuss what was then called the software crisis this software crisis resulted directly from the introduction of new
computer hardware based on integrated cir cuits their power made hitherto unrealisable computer applications a feasible proposition the resulting software was orders of magnitude
larger and more com plex than previous software systems early experience in building these systems showed that informal software devel opment was not good enough major projects were
sometimes years late the soft ware cost much more than predicted was unreliable was difficult to maintain and performed poorly software development was in crisis hardware costs were
tum bling whilst software costs were rising rapidly new techniques and methods were needed to control the complexity inherent in large software systems these techniques have become
part of software engineering and are now widely used however as our ability to produce software has increased so too has the com plexity of the software systems that we need new
technologies resulting from the convergence of computers and communication systems and complex graphical user interfaces place new demands on software engineers as many companies
still do not apply software engineering techniques effectively too many projects still pro duce software that is unreliable delivered late and over budget i think that we have made
tremendous progress since 1968 and that the devel opment of software engineering has markedly improved our software we have a much better understanding of the activities involved in
software development we have developed effective methods of software specification design and implemen tation new notations and tools reduce the effort required to produce large and
com plex systems we know now that there is no single ideal approach to software engineering the wide diversity of different types of systems and organisations that use these systems
means that we need a diversity of approaches to software development however fundamental notions of process and system organisation underlie all of these techniques and these are
the essence of software engineering  software engineers can be rightly proud of their achievements without com plex software we would not have explored space would not have the
internet and modern telecommunications and all forms of travel would be more dangerous and expensive software engineering has contributed a great deal and i am convinced that as the
discipline matures its contributions in the 21st century will be even greater  this section is designed to answer some fundamental questions about software engi neering and to give
you some impression of my views of the discipline the for mat that i have used here is the faq frequently asked questions list this approach is commonly used in internet newsgroups
to provide newcomers with answers to frequently asked questions i think that it is a very effective way to give a succinct introduction to the subject of software engineering figure
many people equate the term software with computer programs however i prefer a broader definition where software is not just the programs but also all associated doc umentation and
configuration data that is needed to make these programs operate cor rectly a software system usually consists of a number of separate programs configuration files which are used to
set up these programs system documentation which describes the structure of the system and user documentation which explains how to use the system and web sites for users to
download recent product information software engineers are concerned with developing software products ie soft ware which can be sold to a customer there are two fundamental types
of software product opment organisation and sold on the open market to any customer who is able to buy them examples of this type of product include software for pcs such as
databases word processors drawing packages and project management tools by a particular customer a software contractor develops the software especially for that customer examples of
this type of software include control systems for electronic devices systems written to support a particular business process and air traffic control systems  may be developed for a
particular customer or may be developed for a general market with all aspects of software production software support about software for custom products the specification is usually
```

Output for book: Final Text after Pre-Processing

Tokenization

For tokenizing the text, we use the word_tokenize function of nltk library by passing our finalText into it. Tokenizers divide strings into lists of substrings.

```
[ ]  #Tokenizing the text
     tokens = nltk.word_tokenize(finalText)
     tokens

     ['objectives',
      'the',
      'objectives',
      'of',
      'this',
      'chapter',
      'are',
      'to',
      'introduce',
      'software',
      'engineering',
      'and',
      'to',
      'provide',
      'a',
      'framework',
      'for',
      'understanding',
      'the',
      'rest',
      'of',
      'the',
      'book',
      'when',
      'you',
      'have',
      'read',
      'this',
      'chapter',
      'you',
      'will',
      'understand',
      'what',
      'software',
      'engineering',
```

List of tokens

Project Round 1

Problem Statement Round 1

1. Download any Computer Science book (in PDF Format) that you had used in one of the course that you have studied till now at LNMIIT but that contains at least 200 pages in it.

2. Convert the book from PDF to text format (you can use online web-based tools for this). Using Python do the following:

3. Import the text, lets call it as T1 (book that you have downloaded)

4. Perform simple text pre-processing steps — you may have to do the removal of running section / chapter names / remove the pictures / tables and so on. Explore the txt file you will understand.

5. Tokenise the text T1

6. Analyze the frequency distribution of tokens in T1.

7. Create a Word Cloud of T1 using the token that you have got

8. Remove the stop words from T1 and then again create a word cloud - what's the difference it gives when you compare with word cloud got before the removal of stop words?

9. Evaluate the relationship between the word length and frequency for T1 — what's your result?

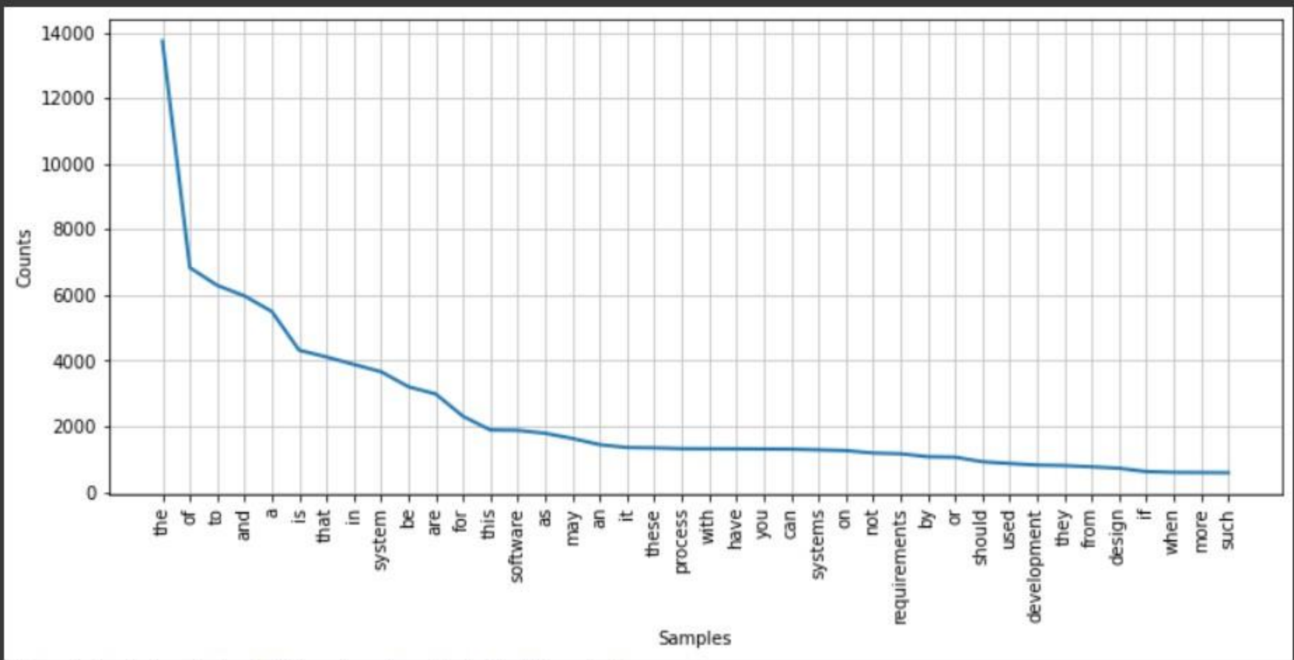
10. Do PoS Tagging for T1 using anyone of the four tag sets studied in the class and get the distribution of various tags (preferably use Treebank tagset)

Data Visualization

Before Removing Stop Words

• The resulting list of tokens is analyzed by plotting its frequency distribution. We get the distribution by using the FreqDist function in nltk library.

```
#Plotting Frequency Distribution before removing stopwords
tokens = nltk.word_tokenize(finalText)
freq = nltk.FreqDist(tokens)
plt.figure(figsize=(12,5))
freq.plot(40, cumulative=False)
```

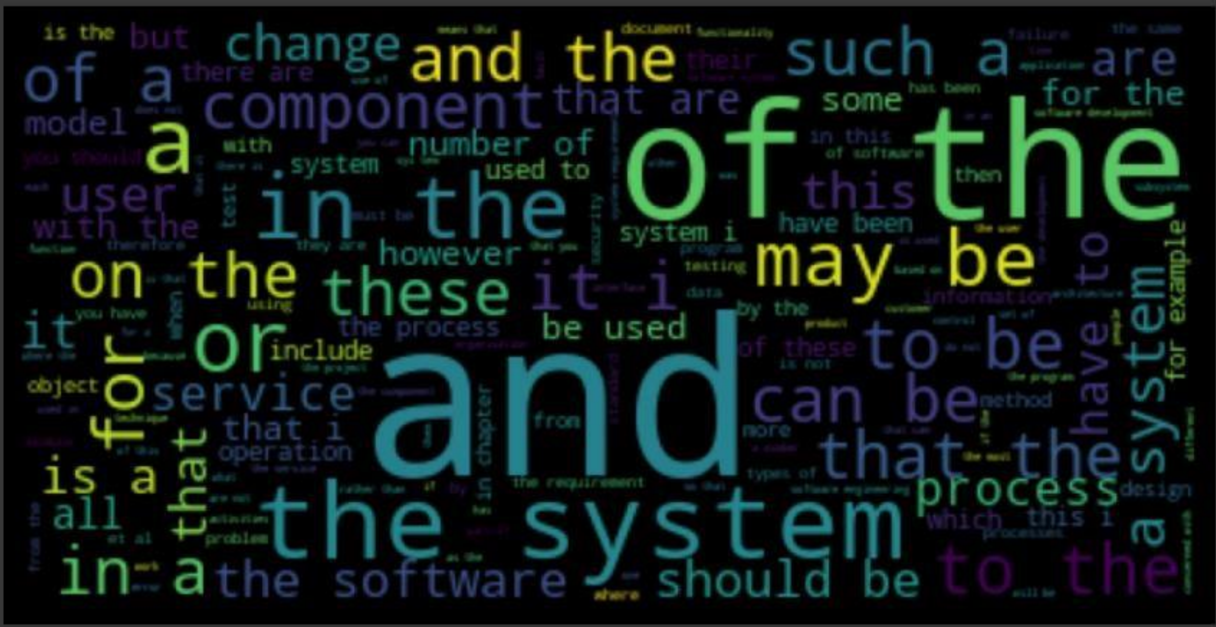



```
<AxesSubplot:xlabel='Samples', ylabel='Counts'>
```

Frequency Distribution of Tokens in finalText beforer removing stop words

- Without removing Stop Words, we create a Word Cloud using the wordcloud library and passing our text into its generate method. The word cloud generated is then plotted using matplotlib.
- In a word cloud, size of each word is proportional to its frequency or importance in the text.

```
[ ] #Creating a wordcloud before removing StopWords
    wordcloud = WordCloud(stopwords={}, background_color='black').generate(finalText)

    #Plotting the wordcloud
    plt.figure(figsize = (10,10), facecolor = 'black')
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.tight_layout(pad = 3)
    plt.show()
```



Output for Word Cloud (with Stopwords)

Inference

- “The, of, to, and, a, is, that, in” are the largest words seen in the word cloud.
- And as inferred from the frequency distribution as well these stop words occupy an unnecessary high frequency count which is meaningless in the processing of the text.
- These hide the essence of the corpus.

Removing Stop Words

Stopwords refers to the extra common words used in natural language such as articles (the, an, a). In NLP and text mining applications, stop words are used to eliminate unimportant words, allowing applications to focus on the important words instead. So, we need to eliminatethe Stop words.

a. We include STOPWORDS and utilize it to create a set of english stop words.

b. We then filter our tokens by eliminating all the words that are common in the stopword set and our tokens.

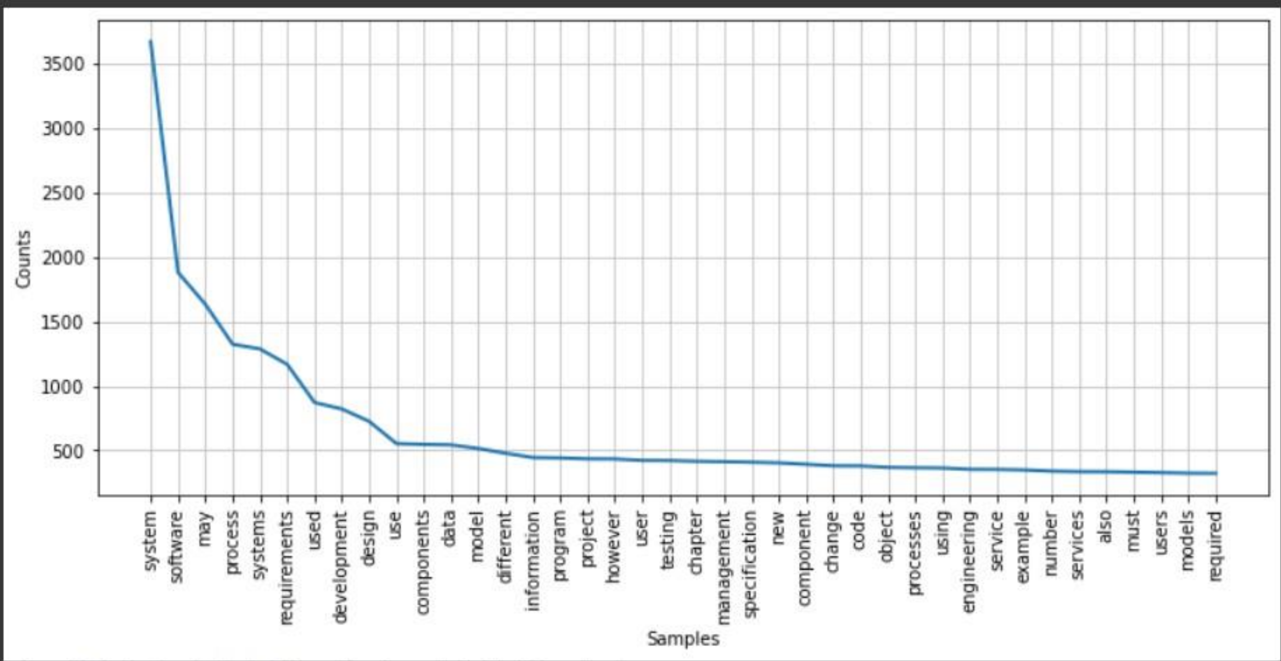
```
[ ] #Removing Stopwords
    stop_words =set(stopwords.words('english'))
    filtered_tokens =[w for w in tokens if w not in stop_words]
    tokens = filtered_tokens
    finalText = " "
    finalText = finalText.join(tokens)
    finalText
```

```
'objectives objectives chapter introduce software engineering provide framework understanding rest book read chapter understand software engineering important know answers key
questions provide introduction software engineering understand ethical professional issues important software engineers contents virtually countries depend complex computerbased
systems national infrastructures utilities rely computerbased systems electrical prod ucts include computer controlling software industrial manufacturing dis tribution completely
computerised financial system therefore producing maintaining software costeffectively essential functioning national international economies software engineering engineering
discipline whose focus cost effective development highquality software systems software abstract intangible constrained materials governed physical laws manufacturing processes ways
simplifies software engineering physical limitations potential software however lack nat ural constraints means software easily become extremely complex hence difficult understand
notion software engineering first proposed 1968 conference held discuss called software crisis software crisis resulted directly introduction new computer hardware based integrated
cir cuits power made hitherto unrealisable computer applications feasible proposition resulting software orders magnitude larger com plex previous software systems early experience
building systems showed informal software devel opment good enough major projects sometimes years late soft ware cost much predicted unreliable difficult maintain performed poorly
software development crisis hardware costs tum bling whilst software costs rising rapidly new techniques methods needed control complexity inherent large software systems techniques
become part software engineering widely used however ability produce software increased com plexity software systems need new technologies resulting convergence computers
communication systems complex graphical user interfaces place new demands software engineers many companies still apply software engineering techniques effectively many projects
still pro duce software unreliable delivered late budget think made tremendous progress since 1968 devel opment software engineering markedly improved software much better
understanding activities involved software development developed effective methods software specification design implemen tation new notations tools reduce effort required produce
large com plex systems know single ideal approach software engineering wide diversity different types systems organisations use systems means need diversity approaches software
development however fundamental notions process system organisation underlie techniques essence software engineering software engineers rightly proud achievements without com plex
software would explored space would internet modern telecommunications forms travel would dangerous expensive software engineering contributed great deal convinced discipline
matures contributions 21st century even greater section designed answer fundamental questions software engi neering give impression views discipline mat used faq frequently asked
questions list approach commonly used internet newsgroups provide newcomers answers frequently asked questions think effective way give succinct introduction subject software
engineering figure many people equate term software computer programs however prefer broader definition software programs also associated doc umentation configuration data needed
make programs operate cor rectly software system usually consists number separate programs configuration files used set programs system documentation describes structure system user
documentation explains use system web sites users download recent product information software engineers concerned developing software products ie soft ware sold customer two
fundamental types software product opment organisation sold open market customer able buy examples type product include software pcs databases word processors drawing packages
project management tools particular customer software contractor develops software especially customer examples type software include control systems electronic devices systems
written support particular business process air traffic control systems may developed particular customer may developed general market aspects software production software support
software custom products specification usually developed controlled engineering organisation buying software software developers must work specification however line types products
becoming increasingly blurred software companies starting generic system customising needs particular customer enterprise resource planning erp sys tems sap system best examples
approach large complex system adapted company incorporating information business rules processes reports required software engineering engineering discipline concerned aspects
software production early stages system specification maintaining system gone use definition two key phrases ods tools appropriate use selectively always try discover solutions
problems even applicable theories methods engineers also recognise must work organisational financial constraints look solutions within constraints technical processes software
development also activities software project management development tools meth ods theories support software production general software engineers adopt systematic organised
approach work often effective way produce highquality software however engineering selecting appropriate method set circum stances creative less formal approach development may
```

Final Text after removing Stopwords

We again create the frequency graphs of the remaining words.

```
#Plotting Frequency Distribution after removing stopwords
tokens = nltk.word_tokenize(finalText)
freq = nltk.FreqDist(tokens)
plt.figure(figsize=(12,5))
freq.plot(40, cumulative=False)
```

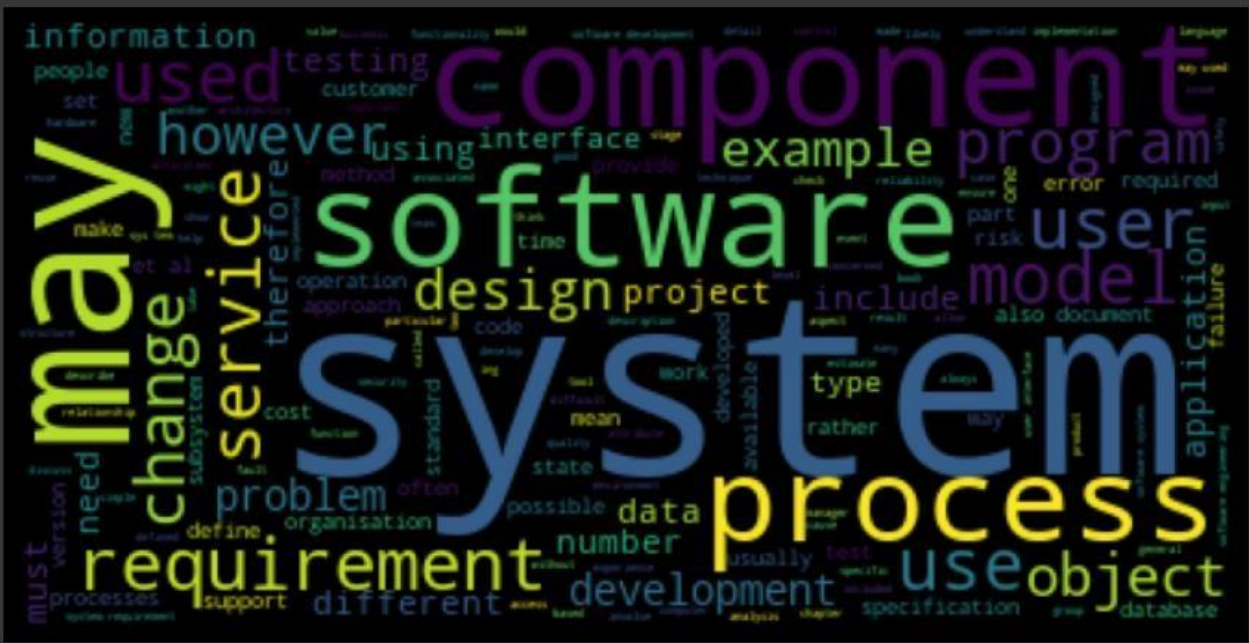



```
<AxesSubplot:xlabel='Samples', ylabel='Counts'>
```

Frequency Distribution of Tokens in finalText after removing stop words

• After cleaning the data of stopwords, we again create the word cloud using the wordcloud library

```
#Generating wordcloud after removing stopwords
wordcloud = WordCloud(stopwords={}, background_color='black').generate(finalText)

plt.figure(figsize = (10,10), facecolor = 'black')
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 3)
plt.show()
```



Output for Word Cloud (without Stopwords)

Inference

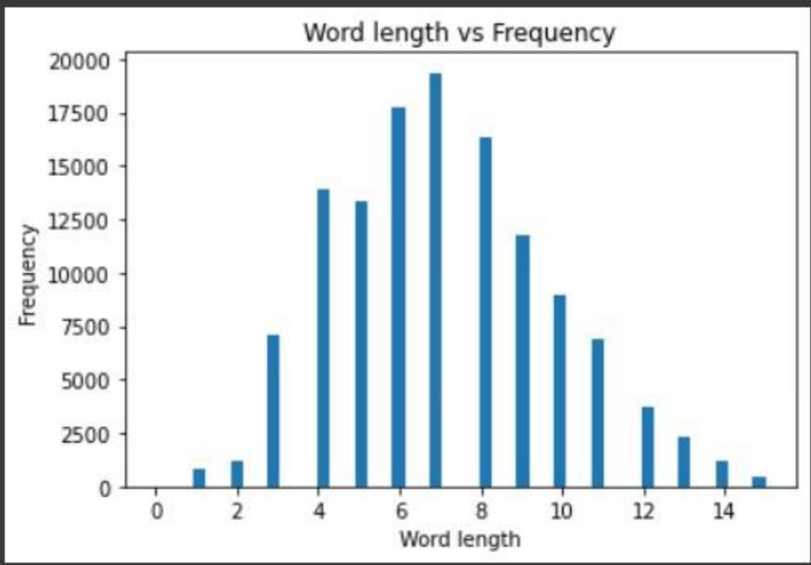
- The prominent words now are “system, software, process, requirements, used, development” etc instead of stop words.
- Frequency distribution also shows more subject specific words occupying higher frequencies.
- Now true essence of the text is delivered.

Relation Between the Word Length and Frequency

Here, we tend to analyse a relationship between the word length and how many words with such word length occurs.

- We first associate a bin for the bar graph using "numpy" library
- Then using len( ) function we calculate the length of each token
- Then we plot a graph for frequency of such word lengths using matplotlib.pyplot

```
[ ]  #Plotting relationship between Word Length and Frequency
     wordLen = [len(w) for w in tokens]
     plt.hist(wordLen, bins=np.linspace(0,15))

     plt.xlabel('Word length')
     plt.ylabel('Frequency')
     plt.title('Word length vs Frequency')
     plt.show()
```



Inference

- It is observed that most of the common words are of medium length between 4 and 10.
- As we go away from the peak, generally the count decreases.
- Distribution appears to be roughly normal

PoS Tagging

- Finally, after complete processing, now we assign appropriate Part-of-Speech Tags to the words.
- We utilize the pos_tag feature of nltk library and pass our list of tokens into it to get the POS tagged list of tokens.
- The pos_tag() function uses the Penn Treebank Tag Set , which has 36 tags to assign from to the words.
- A tuple consisting of the tokens and tags is returned.

```
#POS Tagging
postTag = nltk.pos_tag(tokens)
postTag
```

```
[('objectives', 'NNS'),
 ('objectives', 'VBZ'),
 ('chapter', 'NN'),
 ('introduce', 'NN'),
 ('software', 'NN'),
 ('engineering', 'NN'),
 ('provide', 'NN'),
 ('framework', 'NN'),
 ('understanding', 'VBG'),
 ('rest', 'JJ'),
 ('book', 'NN'),
 ('read', 'VBP'),
 ('chapter', 'NN'),
 ('understand', 'NN'),
 ('software', 'NN'),
 ('engineering', 'NN'),
 ('important', 'JJ'),
 ('know', 'VBP'),
 ('answers', 'NNS'),
 ('key', 'JJ'),
 ('questions', 'NNS'),
 ('provide', 'VBP'),
 ('introduction', 'NN'),
 ('software', 'NN'),
 ('engineering', 'NN'),
 ('understand', 'JJ'),
 ('ethical', 'JJ'),
 ('professional', 'NN'),
 ('issues', 'NNS'),
 ('important', 'JJ'),
 ('software', 'NN'),
 ('engineers', 'NNS'),
 ('contents', 'NNS'),
 ('virtually', 'RB'),
 ('countries', 'NNS'),
 ('depend', 'VBP'),
```

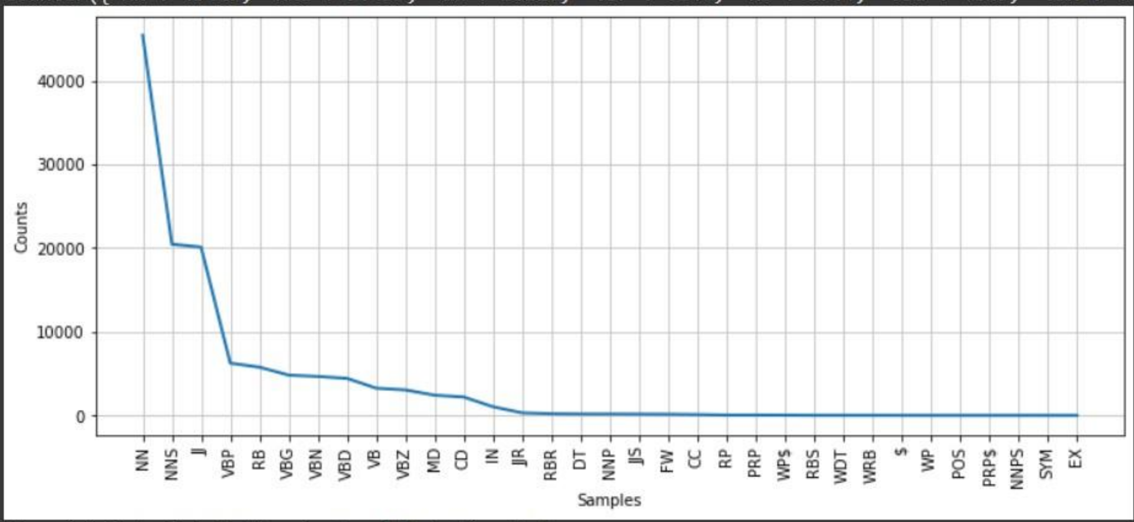
Part-of-Speech Tags

Distribution of Part-of-Speech Tags

- To analyse the obtained list, we plot the frequency of different tags.
- “Counter” library was used which calculates the frequency of each tag used for the final text.

```
#Plotting Tag frequency
tagCount = Counter(tag for word, tag in postTag)
print(tagCount)
tagFreq = nltk.FreqDist(tagCount)
plt.figure(figsize=(12,5))
tagFreq.plot(40, cumulative=False)
```

```
Counter({'NN': 45417, 'NNS': 20459, 'JJ': 20113, 'VBP': 6246, 'RB': 5764, 'VBG': 4818, 'VBN': 4655, 'VBD': 4426,
```




```
<AxesSubplot:xlabel='Samples', ylabel='Counts'>
```

Inference

- From the above plot we see NN is the most common tag out of all.
- Also, from the tuple of pos tags returned we see that same words are assigned many tags. For example, first 2 words ‘objectives’ is assigned ‘NNS’ and ‘VBZ’ respectively.
- There are a smaller number of ambiguous words (i.e., with multiple parts of speech tags) but they are being frequently used (as observed from frequency distribution and tuple of pos tags).

Project Round 2

Problem Statement Round 2

First Part:

1. Find the nouns and verbs in the book. Get the categories that these words fall under in the WordNet. Note that there are 25 categories and 16 categories for Nouns and Verbs respectively.
2. Get the frequency of each category for each noun and verb in their corresponding heirarchies and plot a histogram for the same.

Second Part:

1. Recognise all entities (Types given in Fig 22.1). 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 book, do a manual labelling and then compare your result with it. Present the accuracy here and F1 score.

Third Part:

1. For extracting the relationship between the entities from the book - what are the features necessary for this? Use the ideas given in the book and presented in the class to augment the data of Entities and augment that data by extracting additional features and build the table and present it.

Removing Stop Words

Stopwords refers to the extra common words used in natural language such as articles (the, an, a). In NLP and text mining applications, stop words are used to eliminate unimportant words, allowing applications to focus on the important words instead. So, we need to eliminatethe Stop words.

a. We include STOPWORDS and utilize it to create a set of english stop words.

b. We then filter our tokens by eliminating all the words that are common in the stopword set and our tokens.

Removing stopwords

```
#Removing Stopwords
stop_words =set(stopwords.words('english'))
filtered_tokens =[w for w in tokens if w not in stop_words]
tokens = filtered_tokens
finalText = " "
finalText = finalText.join(tokens)
finalText[:5000]
```

```
'objectives objectives chapter introduce software engineering provide framework understanding rest book read chapter understand software engineering important know answers
key questions provide introduction software engineering understand ethical professional issues important software engineers contents virtually countries depend complex
computerbased systems national infrastructures utilities rely computerbased systems electrical products include computer controlling software industrial manufacturing
distribution completely computerised financial system therefore producing maintaining software costeffectively essential functioning national international economies
software engineering engineering discipline whose focus costeffective development highquality software systems software abstract intangible constrained materials governed
physical laws manufacturing processes ways simplifies software engineering physical limitations potential software however lack natural constraints means software easily
become extremely complex hence difficult understand notion software engineering first proposed 1968 conference held discuss called software crisis software crisis resulted
directly introduction new computer hardware based integrated circuits power made hitherto unrealisable computer applications feasible proposition resulting software orders
magnitude larger complex previous software systems early experience building systems showed informal software development good enough major projects sometimes years late
software cost much predicted unreliable difficult maintain performed poorly software development crisis hardware costs tumbling whilst software costs rising rapidly new
techniques methods needed control complexity inherent large software systems techniques become part software engineering widely used however ability produce software
increased complexity software systems need new technologies resulting convergence computers communication systems complex graphical user interfaces place new demands
software engineers many companies still apply software engineering techniques effectively many projects still produce software unreliable delivered late budget think made
tremendous progress since 1968 development software engineering markedly improved software much better understanding activities involved software development developed
effective methods software specification design implementation new notations tools reduce effort required produce large complex systems know single ideal approach software
engineering wide diversity different types systems organisations use systems means need diversity approaches software development however fundamental notions process
system organisation underlie techniques essence software engineering software engineers rightly proud achievements without complex software would explored space would
internet modern telecommunications forms travel would dangerous expensive software engineering contributed great deal convinced discipline matures contributions 21st
century even greater section designed answer fundamental questions software engineering give impression views discipline format used faq frequently asked questions list
approach commonly used internet newsgroups provide newcomers answers frequently asked questions think effective way give succinct introduction subject software engineering
figure many people equate term software computer programs however prefer broader definition software programs also associated documentation configuration data needed make
programs operate correctly software system usually consists number separate programs configuration files used set programs system documentation describes structure system
user documentation explains use system web sites users download recent product information software engineers concerned developing software products ie software sold
customer two fundamental types software product buy examples type product include software pcs databases word processors drawing packages project management tools
particular customer software contractor develops software especially customer examples type software include control systems electronic devices systems written support
particular business process air traffic control systems may developed particular customer may developed general market aspects software production software support
software custom products specification usually developed controlled engineering organisation buying software software developers must work specification however line types
products becoming increasingly blurred software companies starting generic system customising needs particular customer enterprise resource planning erp systems sap system
best examples approach large complex system adapted company incorporating information business rules processes reports required software engineering engineering discipline
concerned aspects software production early stages system specification maintaining system gone use de'
```

Final Text after removing Stopwords

PoS Tagging

- After processing, now we assign appropriate Part-of-Speech Tags to the words.
- We utilize the pos_tag feature of nltk library and pass our list of tokens into it to get the POS tagged list of tokens.
- The pos_tag() function uses the Penn Treebank Tag Set , which has 36 tags to assign from to the words.
- A tuple consisting of the tokens and tags is returned.

Perform POS Tagging

```
[ ] #POS Tagging
    postTag = nltk.pos_tag(tokens)
    postTag

[('objectives', 'NNS'),
 ('objectives', 'VBZ'),
 ('chapter', 'NN'),
 ('introduce', 'NN'),
 ('software', 'NN'),
 ('engineering', 'NN'),
 ('provide', 'NN'),
 ('framework', 'NN'),
 ('understanding', 'VBG'),
 ('rest', 'JJ'),
 ('book', 'NN'),
 ('read', 'VBP'),
 ('chapter', 'NN'),
 ('understand', 'NN'),
 ('software', 'NN'),
 ('engineering', 'NN'),
 ('important', 'JJ'),
 ('know', 'VBP'),
 ('answers', 'NNS'),
 ('key', 'JJ'),
 ('questions', 'NNS'),
 ('provide', 'VBP'),
 ('introduction', 'NN'),
 ('software', 'NN'),
 ('engineering', 'NN'),
 ('understand', 'JJ'),
 ('ethical', 'JJ'),
 ('professional', 'NN'),
 ('issues', 'NNS'),
 ('important', 'JJ'),
 ('software', 'NN'),
 ('engineers', 'NNS'),
 ('contents', 'NNS'),
 ('virtually', 'RB'),
 ('countries', 'NNS'),
 ('depend', 'VBP'),
 ('complex', 'JJ'),
 ('computerbased', 'VBN'),
 ('systems', 'NNS'),
 ('national', 'JJ'),
```

First Part

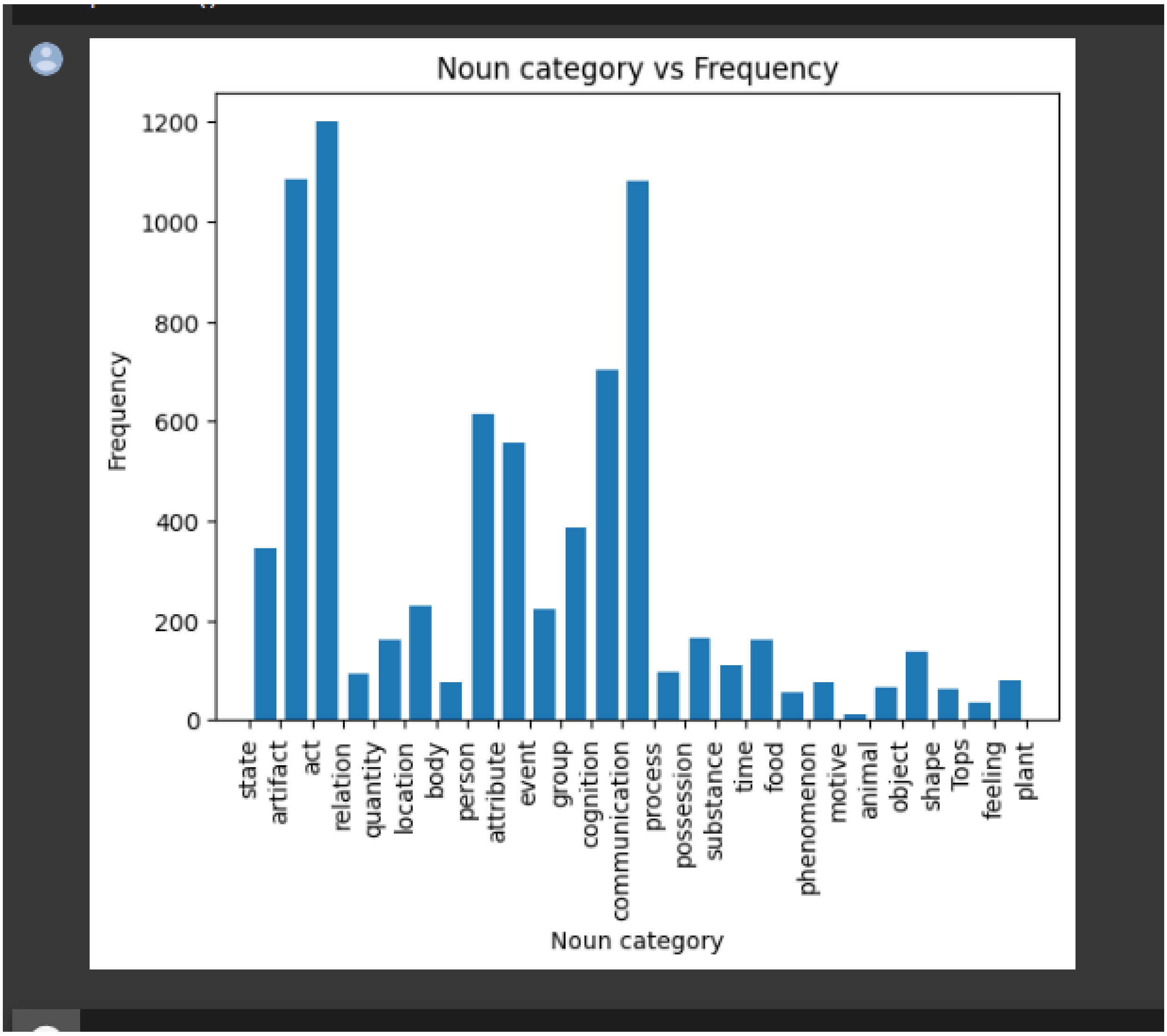
Categorising Nouns and plotting their frequencies

- Separate the nouns from the text, out of all the assigned tags, check for the words having NN tag.
- Now iterate through all 25 categories of nouns using:
  for syn in wn.synsets(word[0], wn.NOUN):
- Record the frequency of each category by incrementing the count in a dictionary for every occurrence.

```
cat_freq = {}
f = []
for word in set(postTag):
  if (word[1] == 'NN'):
    for syn in wn.synsets(word[0], wn.NOUN):
      key = syn.lexname().split('.')[1]
      if (key in cat_freq):
        cat_freq[key] += 1
      else:
        cat_freq[key] = 1
      f.append(key)
      print(word[0], syn.lexname())
```

```
judgement noun.communication
judgement noun.cognition
judgement noun.cognition
judgement noun.cognition
judgement noun.attribute
judgement noun.act
judgement noun.act
pm noun.act
pm noun.substance
pm noun.person
pm noun.communication
computer noun.artifact
computer noun.person
convergence noun.event
convergence noun.cognition
convergence noun.cognition
convergence noun.act
knowing noun.cognition
initialisation noun.communication
session noun.communication
session noun.time
session noun.act
session noun.group
manipulation noun.act
manipulation noun.act
medication noun.artifact
medication noun.act
press noun.state
press noun.communication
```

- Now plot a histogram using matplotlib.pyplot
- X axis depicts Noun category.
- Y axis depicts Frequency of category.



Frequency distribution graph for noun categories

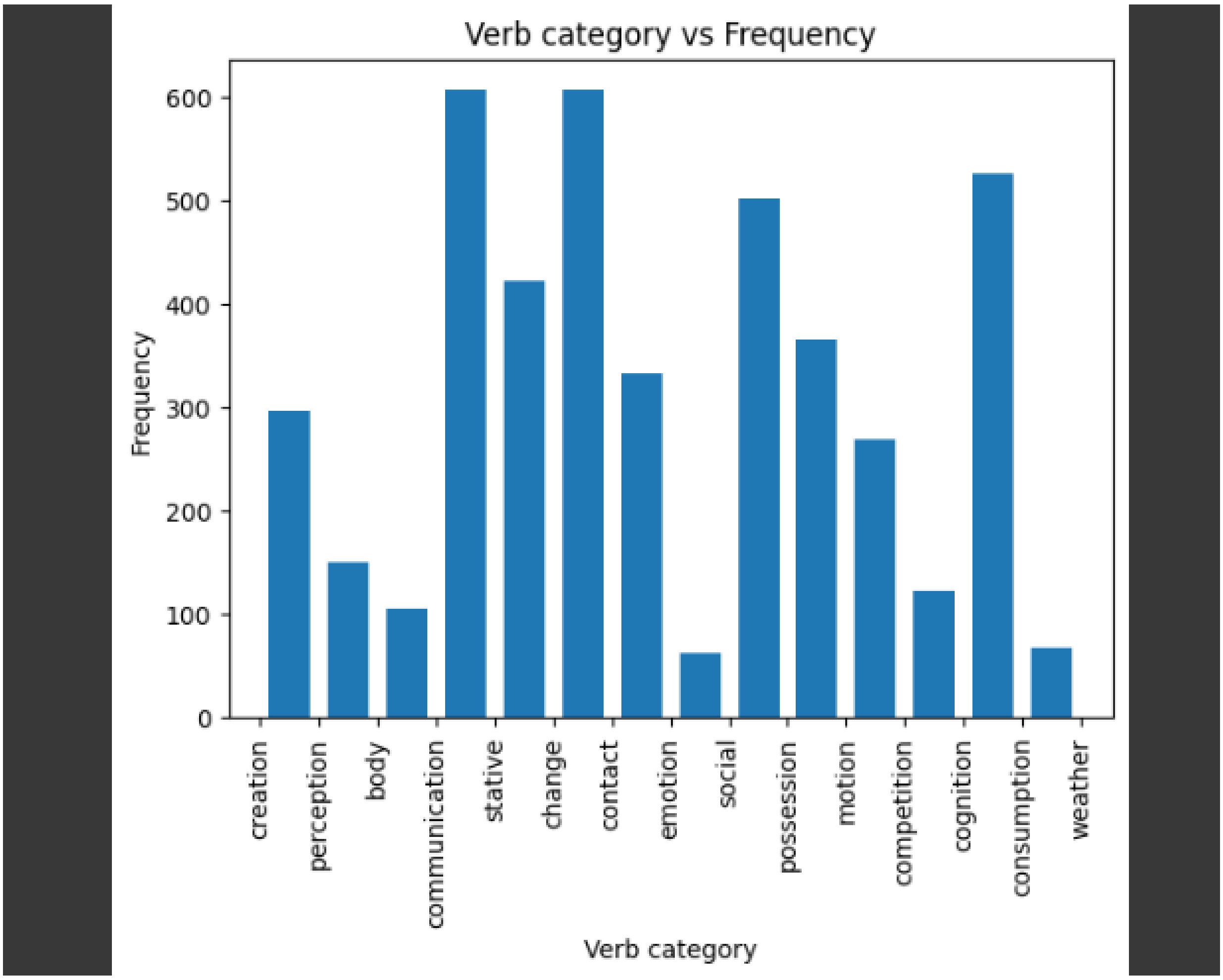
Categorising verbs and plotting their frequencies

- Separate the verbs from the text, out of all the assigned tags, check for the words having VB tag.
- Now iterate through all 25 categories of nouns using:
  for syn in wn.synsets(word[0], wn.VERB):
- Record the frequency of each category by incrementing the count in a dictionary for every occurrence.

```
cat_freq1 = {}
f1 = []
for word in set(postTag):
  if (word[1] == 'VB'):
    for syn in wn.synsets(word[0], wn.VERB):
      key = syn.lexname().split('.')[1]
      if (key in cat_freq1):
        cat_freq1[key] += 1
      else:
        cat_freq1[key] = 1
      f1.append(key)
      print(word[0], syn.lexname())


cat1 = [x for (x, y) in cat_freq1.items()]
plt.hist(f1, bins=cat1, rwidth=0.7)
plt.xlabel('Word length')
plt.xticks(rotation='vertical')
plt.ylabel('Frequency')
plt.title('Word length vs Frequency')
plt.show()
```

```
confused verb.cognition
confused verb.cognition
confused verb.emotion
confused verb.creation
confused verb.cognition
read verb.cognition
read verb.stative
read verb.cognition
read verb.cognition
read verb.cognition
read verb.cognition
read verb.cognition
read verb.communication
read verb.creation
read verb.cognition
read verb.cognition
secure verb.possession
secure verb.contact
secure verb.possession
secure verb.communication
secure verb.contact
secure verb.contact
signal verb.communication
signal verb.communication
reflected verb.perception
reflected verb.cognition
reflected verb.perception
reflected verb.weather
reflected verb.perception
reflected verb.communication
reflected verb.communication
introduced verb.communication
introduced verb.creation
introduced verb.change
```

- Now plot a histogram using matplotlib.pyplot
- X axis depicts Verb category.
- Y axis depicts Frequency of category.



Frequency distribution graph for Verb categories

Second Part

Manual NER Tagging

- Below is the screenshot for manual NER tagging for a 1000 words random passage from the book
- It can be found in the 'Manual NER.txt' file of the github repo.

```
whereas driver (las vegas)(GPE) may never notice problem difficulty definitions take account severity failure consequences unavailability people naturally concerned system failures serious consequences perception
system reliability influenced consequences example say failure initialisation engine management software causes car engine cut immediately starting operates correctly restart corrects initialisation problem affect
normal operation car many drivers would think repair needed contrast drivers think engine cuts driving high speed per month say unreliable unsafe must repaired strict definition reliability relates system
implementation specification system behaving reliably behaviour consistent defined specification however common cause perceived unreliability system specification match expectations system users unfortunately many
specifications incomplete incorrect left software engineers interpret system behave domain experts may therefore implement behaviour users expect reliability availability compromised system failures may failure
provide service failure deliver service specified delivery service way unsafe insecure failures consequence specification errors failures associated systems telecommunications system however many failures
consequence erroneous system behaviour derives faults system discussing reliability helpful distinguish terms fault error failure may parts system never used faults necessarily result system errors faulty state may
transient may corrected erroneous behaviour occurs system errors may result system failures behaviour may also transient observable effects system may deliver service expected users unexpected system users error
example failure initialise variable could lead variable wrong value used include protection ensures erroneous behaviour discovered corrected system services affected system faults examples techniques include
avoiding errorprone programming language constructs pointers use static analysis detect program anomalies increase chances faults detected removed system used systematic system testing debugging example
faultdetection technique system errors ensure system errors result system failures incorporation selfchecking facilities system use redundant system modules examples fault tolerance techniques cover development
fault tolerant systems chapter 20 also discuss techniques fault avoidance discuss processbased approaches fault avoidance chapter 27 fault detection chapters 22 23 software faults cause software failures faulty code
executed set inputs expose software fault code works properly inputs program responds producing corresponding output example given input url web browser produces output display requested web page inputoutput
erroneous outputs mapping program erroneous outputs inputs input combinations shown shaded ellipse figure probability particular execution program system input member set inputs cause erroneous output occur input
causing erroneous output associated frequently used part program failures frequent however associated rarely used code users hardly ever see failures user system uses different ways faults affect reliability system
one user may never revealed someone elses mode 1 user 3 however never use inputs erroneous set software always reliable possible inputs user user 3 2 overall reliability program therefore mostly depends number
inputs causing erroneous outputs normal use system users software faults occur exceptional situations little effect systems reliability removing software faults parts system rarely used makes little real difference
reliability seen system users mills et al mills et al 1987 found software removing 60 known errors software led 3 reliability improvement adams adams 1984 study (ibm)(ORG) software products noted many defects
products likely cause failures hundreds thousands months product usage users sociotechnical system may adapt software known faults may share information get around problems may avoid using inputs known cause
problems program failures never arise furthermore experienced users often work around software faults known cause failures deliberately avoid using system features know cause problems example avoid certain features
automatic numbering word processing system used write book repairing faults features may make practical difference reliability seen users safetycritical systems systems essential system operation always safe system
never damage people systems environment even system fails examples safetycritical systems control monitoring systems aircraft process control systems chemical pharmaceutical plants automobile control systems
hardware control safetycritical systems simpler implement analyse software control however build systems complexity controlled hardware alone software control essential need manage large numbers sensors actuators
complex control laws example complexity found advanced aerodynamically unstable military aircraft require continual softwarecontrolled adjustment flight surfaces ensure crash safetycritical software falls two
classes type software injury examples systems computeraided engineering design systems whose malfunctioning might result design fault object designed fault may cause injury people designed system malfunctions
another example secondary safetycritical system medical database holding details drugs administered patients errors system might result incorrect drug dosage administered system reliability system safety related
separate dependability attributes course safetycritical system reliable conform specification operate without failures may incorporate faulttolerant features provide continuous service even faults occur however
faulttolerant systems necessarily safe software may still malfunction cause system behaviour results accident apart fact never 100 certain software system faultfree faulttolerant several reasons software systems
reliable necessarily safe system critical situations high percentage system malfunctions (nakajo kume)(PER) 1991 (lutz)(PER) 1993 result specification rather design errors study errors embedded systems (lutz)(PER)
concludes difficulties requirements key root cause safetyrelated software errors persisted integration system testing way may present software unanticipated environment components close failure may behave
erratically generate signals outside ranges handled software situations lead system malfunction anecdotal example mechanic instructed utility management software aircraft raise undercarriage software carried
mechanics instruction perfectly unfortunately plane ground time clearly system disallowed command unless plane air specialised vocabulary evolved discuss safetycritical systems key assuring safety ensure either
accidents occur consequences accident minimal achieved three complementary ways time operate machine avoids hazard operators hands blade pathway human death injury damage property environment computercontrolled
machine injuring operator example accident accident failure sensor detects obstacle front machine example hazard range many people killed result accident minor injury property damage particular hazard hazard
severity range catastrophic many people killed minor minor damage results probability values tend arbitrary range probable say 1100 chance hazard occurring implausible conceivable situations likely hazard could
occur accident risk assessed considering hazard probability hazard severity probability hazard result accident removed result accident example chemical plant system may detect excessive pressure open relief valve
reduce pressure explosion occurs damage may result accident example aircraft engine normally includes automatic fire extinguishers fire occurs often controlled poses threat aircraft accidents generally occur several
things go wrong time analysis serious accidents perrow 1984 suggests almost due combination malfunctions rather single failures unanticipated combination led interactions resulted system failure perrow also suggests
impossible anticipate possible combinations system
```

Manually assigned NER tags

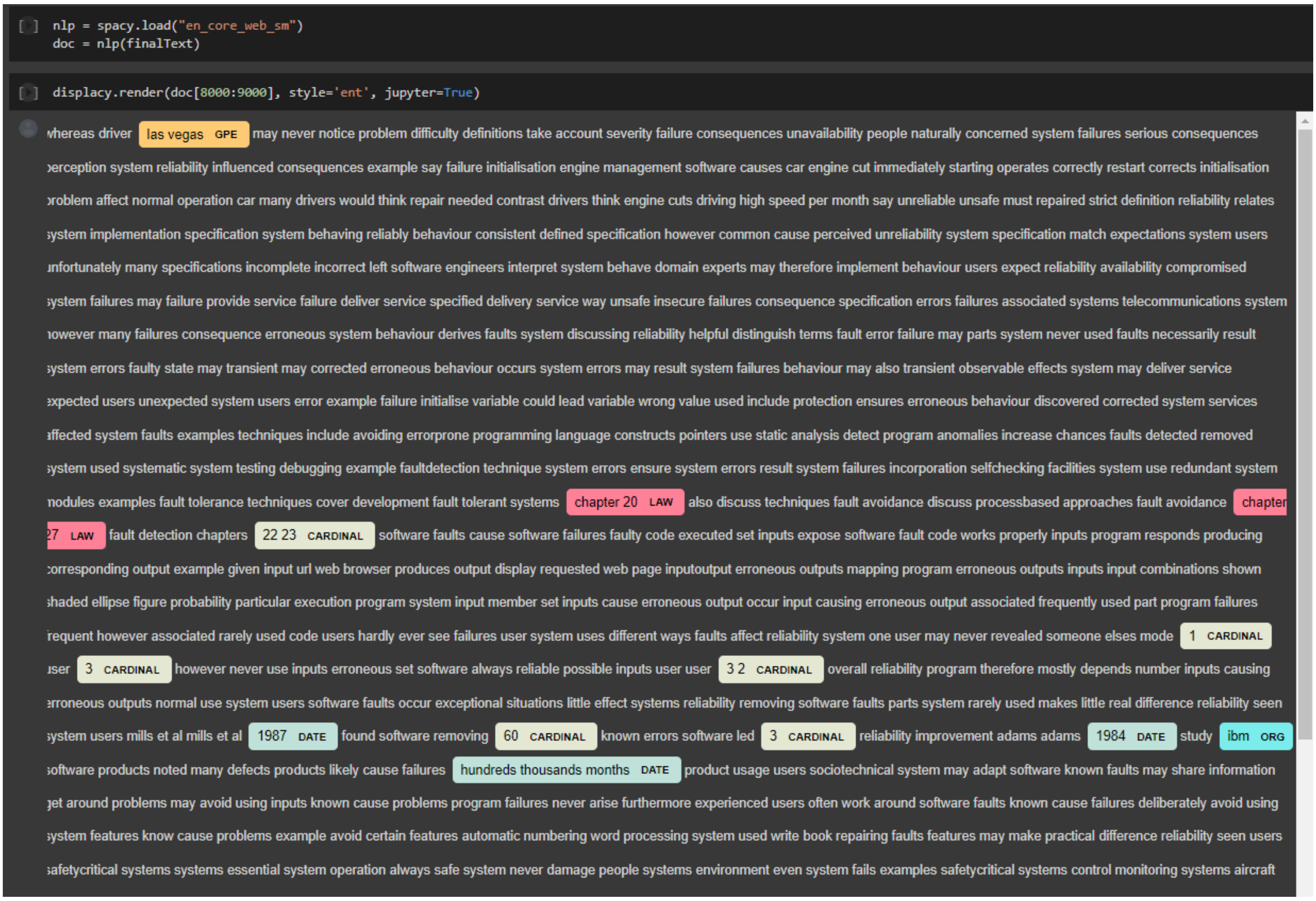
NER Tagging

- Load the wordnet
  - nlp = spacy.load("en_core_web_sm")

- Initialize the text as a part of this wordnet
  - doc = nlp(finalText)

- Display the NER labels as follows:
  - displacy.render(doc, style='ent', jupyter=True)

- Get the required NER labels for 8000th to 9000th word of the book (the random passage):
  - displacy.render(doc[8000:9000], style='ent', jupyter=True



Algorithm generated NER Tags

NER Labelling Evaluation

- Get the values for True Positive, True Negative, False Positive, and False Negative by a comparision with the manual NER
- Get the values for Recall, Precision, Accuracy and F1-Score:
  - recall = TP / (TP + FN)
  - precision = TP / (TP + FP)
  - accuracy = (TP + TN) / (TP + FN + FP + TN)
  - F1Score = 2 * precision * recall / (precision + recall)
- Display accuracy and F1Score

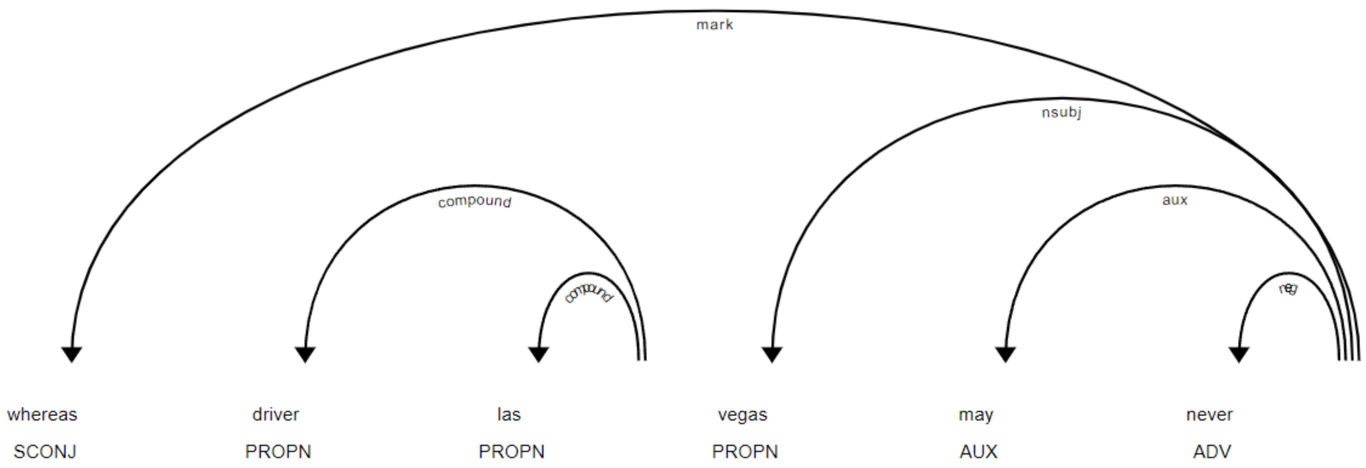
```
[ ]  TP = 3
     FN = 2
     FP = 0
     TN = 995

     accuracy = (TP + TN) / (TP + FN + FP + TN)
     precision = TP / (TP + FP)
     recall = TP / (TP + FN)
     F1Score = 2 * precision * recall / (precision + recall)

     accuracy, F1Score

     (0.998, 0.7499999999999999)
```

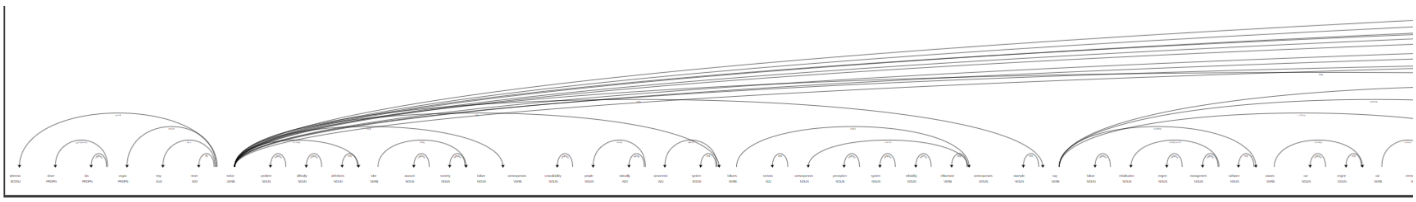
Evaluation scores

Accuracy of Model: 99.8%
F1 Score of Model: 0.7499999

Third Part

Relationship between entities





- Detailed entity relationship chart can be viewed through ipynb notebook.
- Snippets for rough idea are presented here.

GitHub Link: https://github.com/Vatsal32/NLP_Project_Round_1
GitHub Link: https://github.com/Vatsal32/NLP_Project_Round_2

Thank You