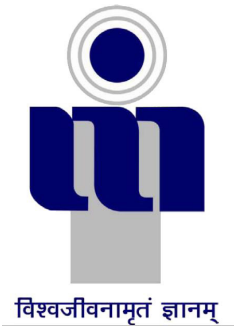
TEXT SUMMARIZATION USING NLTK (TF-IDF MODEL)

A project report,

submitted to **Dr. Debanjan Sadhya**
in the subject of Special Topics in Artificial Intelligence

by

GROUP-3
Amal Shaji (2017BCS-010)
Pobbati Samanyu (2017BCS-021)
Pradyumn Pottapatri (2017BCS-022)
Sandarbh Yadav (2017BCS-027)
Vaibhav Garg (2017BCS-038)
Vikram Chaudhary (2017BCS-039)



ABV INDIAN INSTITUTE OF INFORMATION TECHNOLOGY AND MANAGEMENT
GWALIOR-474 015

TABLE OF CONTENTS

1 Introduction **1**

1.1 Abstractive Summarization 1

1.2 Extractive Summarization 2

2 Methodology **3**

2.1 Text Preprocessing 4

2.1.1 Tokenization 4

2.1.2 Stemming 4

2.1.3 Lemmatization 5

2.1.4 POS Tagging 5

2.1.5 Stopword Removal 5

2.1.6 Lowercasing 5

2.2 Calculating TF-IDF Score 6

2.2.1 Term Frequency 7

2.2.2 Inverse Document Frequency 7

2.2.3 Calculation of TF-IDF score using Python 7

2.3 Calculating Sentence Score 9

2.4 Summary Generation 9

3 Tools Used **10**

3.1 NLTK 10

3.2 PyCharm 11

3.3 Python Standard Library 11

4 Results **12**

5 Conclusion **14**

5.1 Scope for further improvement 14

REFERENCES **15**

CHAPTER 1

Introduction

Text Summarization refers to the technique of generating summary from large chunks of texts by focusing on sections containing useful information. The aim of text summarization is to generate precise and concise summary of given text without losing overall meaning. The two main techniques used for text summarization are: NLP based techniques and Deep Learning based techniques.

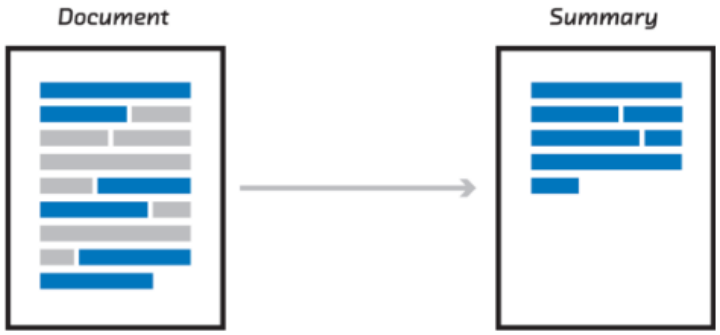


Figure 1.1: Text Summarization

Text Summarization has two basic approaches: Abstractive Summarization and Extractive Summarization.

1.1 Abstractive Summarization

Abstractive summarization refers to the technique of paraphrasing the intent of a given text, just like humans do. New sentences are generated which describe the content of the text. It can be thought of as a pen producing sentences which may not be a part of original text. Advanced deep learning and natural language generation techniques are required for abstractive summarization.

Figure 1.2: Pen = Abstractive and Highlighter = Extractive

1.2 Extractive Summarization

Extractive summarization refers to the technique of selecting important sentences or words from a given text and combining them to obtain summary. The basic procedure is to obtain the importance of sentences, rank them and obtain summary by joining most important sentences. Extractive Summarization can be thought of as a highlighter selecting important parts of text.

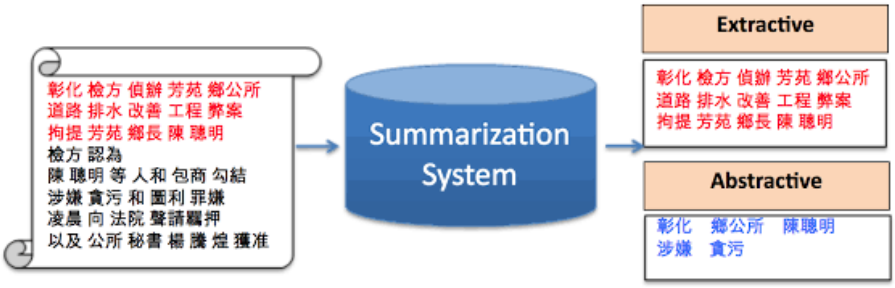


Figure 1.3: Abstractive vs Extractive Summarization

The red text denotes the most important part of the text. This part is used as extractive summary. On the other hand, something new (blue text) is generated as abstractive summary.

In this project, we have used NLP based technique (TF-IDF Model) to generate extractive summaries of given text.

CHAPTER 2

Methodology

The basic steps in Text Summarization using TF-IDF model are:

- Text Preprocessing
- Calculating TF-IDF Score
- Calculating Sentence Score
- Summary Generation

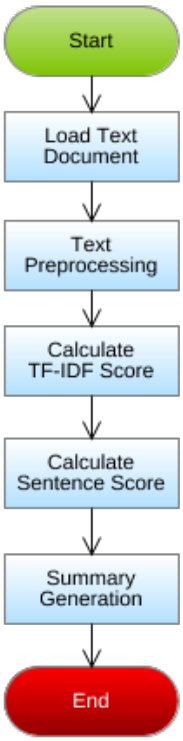


Figure 2.1: Flowchart

2.1 Text Preprocessing

Text Preprocessing refers to the technique of converting text into such a form that algorithms can be applied in a convenient manner. It is an application dependent task. One application's ideal preprocessing might turn out to be a nightmare for other application. Text preprocessing involves techniques such as tokenization, stemming, lemmatization, lowercasing, stopword removal etc.

2.1.1 Tokenization

Tokenization refers to the technique of splitting long strings into small pieces called tokens. Paragraphs can be tokenized into sentences and sentences can be tokenized into words. Tokenization is generally the first step of text preprocessing. [1]

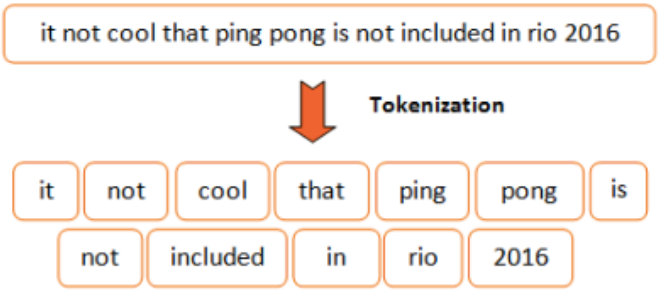


Figure 2.2: Tokenization

2.1.2 Stemming

Stemming refers to the technique of reducing words to their root form. In stemming, the ends of the words are chopped off in hope that the correct root words are generated.

Original word	Stemmed word
connects	connect
connected	connect
connecting	connect
connection	connect

The root need not be a real word but just the canonical form of the word. 'Troubles', 'troubled' and 'troubling' get reduced to 'troubl' instead of 'trouble'.

There are many algorithms for stemming. Porter's algorithm is the most popular algorithm for English language. It is also known as Porter Stemmer.

2.1.3 Lemmatization

Lemmatization is also the technique of mapping words to their root forms. The main difference between stemming and lemmatization is that lemmatization transforms the words to their correct roots instead of just chopping off the ends of the words. 'Better' and 'best' are mapped to 'good'. 'Troubles', 'troubled' and 'troubling' get reduced to 'trouble' and not 'troubl'.

2.1.4 POS Tagging

POS Tagging refers to the technique of mapping words to their parts of speech tag. In this project we have restricted ourselves to nouns and verbs only.

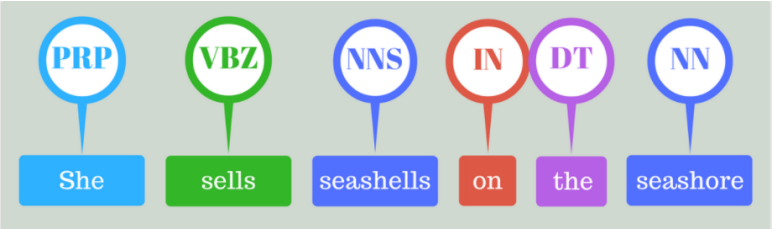


Figure 2.3: POS Tagging

2.1.5 Stopword Removal

Stopwords refer to the most commonly used words in a language. In English, some examples of stopwords are 'a', 'is', 'the' etc. The main reason of removing stopwords is that they contain very less information and after their removal, we can focus on important words.

2.1.6 Lowercasing

Lowercasing refers to the technique of reducing the text to lowercase. It is one of the simplest and most effective preprocessing technique. It is applicable to many NLP and text mining problems. While lowercasing generally helps, it is not recommended for all applications.

Original word	Lowercased word
India	india
INDIA	india
IndiA	india

Apart from these common preprocessing techniques, we have also removed special characters (including non-English characters) and single letter words. The workflow diagram is given below.

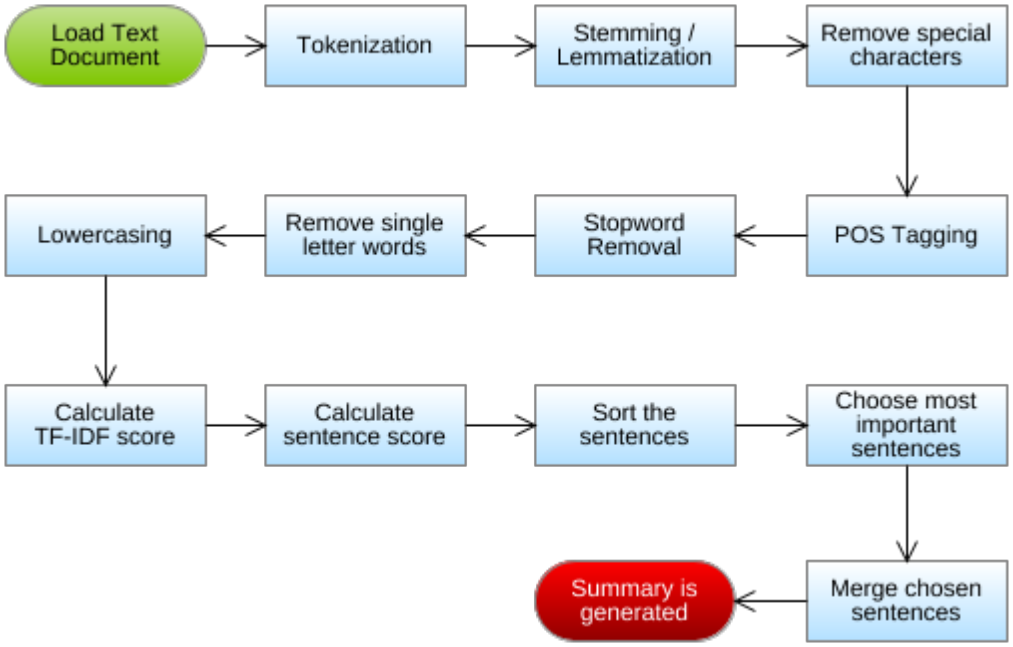


Figure 2.4: Workflow Diagram

2.2 Calculating TF-IDF Score

'TF' in TF-IDF stands for Term Frequency
'IDF' in TF-IDF stands for Inverse Document Frequency

TF-IDF is a weighting technique which assigns a weight to each term and this weight describes the importance of terms in a document. The terms with higher TF-IDF score are considered to be more important. TF-IDF score is the product of Term Frequency (TF) and Inverse Document Frequency (IDF). [2]

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

where t denotes term, d denotes document, D denotes document set.

2.2.1 Term Frequency

Term frequency denotes the number of occurences of a term t in document d. It is usually normalized on logarithmic scale at base 2.

$$tf(t,d) = log(1 + freq(t,d))$$

where t denotes term and d denotes document.

2.2.2 Inverse Document Frequency

Inverse Document Frequency is calculated by taking logarithm of total number of documents divided by number of documents containing the term. It is a measure of how common or rare a word is. Common words have IDF score close to 0.

$$idf(t,D) = log(\frac{N}{count(d \in D : t \in d)})$$

Here, N denotes the number of documents.

2.2.3 Calculation of TF-IDF score using Python

First of all, we calculate the frequency of each unique word (term). This is achieved by creating a Python dictionary with unique words as keys and their frequencies as values.

```
def freq(words):
    words = [word.lower() for word in words]
    dict_freq = {}
    words_unique = []
    for word in words:
        if word not in words_unique:
            words_unique.append(word)
    for word in words_unique:
        dict_freq[word] = words.count(word)
    return dict_freq
```

Figure 2.5: Calculating frequency

After calculating frequencies of unique words, we calculate Term Frequency and Inverse Document Frequency. Here, we have normalised the frequency by dividing it by total number of unique words. The value of term frequency after normalization is between 0 and 1.

```
def tf_score(word, sentence):
    freq_sum = 0
    word_freq_in_sent = 0
    len_sent = len(sentence)
    for word_in_sent in sentence.split():
        if word == word_in_sent:
            word_freq_in_sent = word_freq_in_sent + 1

    tf = word_freq_in_sent / len_sent
    return tf
```

Figure 2.6: Calculating Term Frequency

```
def idf_score(no_of_sent, word, sentences, wordLemmatizer, STOPWORDS):
    no_of_sent_containing_word = 0
    for sentence in sentences:
        sentence = standard.remove_special_chars(str(sentence))
        sentence = re.sub(r'\d+', '', sentence)
        sentence = sentence.split()
        sentence = [word for word in sentence
                    if word.lower() not in STOPWORDS and len(word) > 1]
        sentence = [word.lower() for word in sentence]
        sentence = [wordLemmatizer.lemmatize(word) for word in sentence]

        if word in sentence:
            no_of_sent_containing_word = no_of_sent_containing_word + 1

    idf = math.log10(no_of_sent/no_of_sent_containing_word)
    return idf
```

Figure 2.7: Calculating Inverse Document Frequency

```
def tf_idf_score(tf, idf):
    return tf*idf
```

Figure 2.8: Calculating TF-IDF Score

Using Term Frequency and Inverse Document Frequency, we calculate TF-IDF score.

2.3 Calculating Sentence Score

Each sentence is assigned an initial base score of 0. Sentence score is calculated by adding the TF-IDF scores of the words comprising it. Sentence score denotes how much important a sentence is.

2.4 Summary Generation

After calculation of sentence scores, we sort the sentences in descending order of their scores. The top few sentences with highest scores are selected and returned as summary. In example given below, we retain 50% of the information and hence from a 4 sentence text, 2 sentences with highest scores are returned as summary.

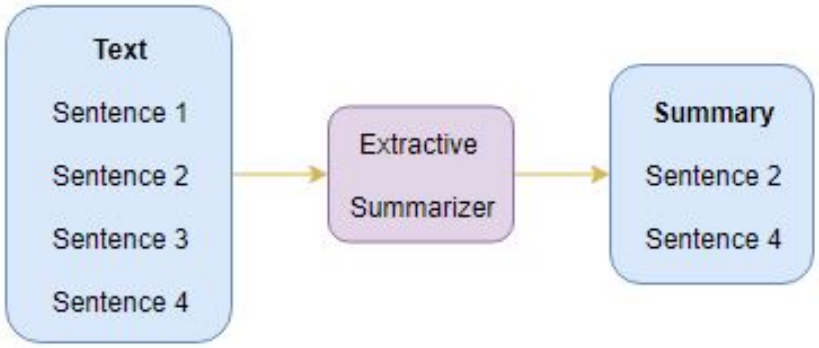


Figure 2.9: Summary Generation

CHAPTER 3

Tools Used

The project has been implemented in Python. The major tools used are NLTK library, Python Standard Library and Python Interpreter (PyCharm).

3.1 NLTK

NLTK stands for Natural Language Toolkit. It is the most popular library for Natural Language Processing in English. It is written in Python and is open-source. It was developed by Steven Bird and Edward Loper. We have used it for word tokenization, sentence tokenization, stemming, lemmatization, stopword removal etc.

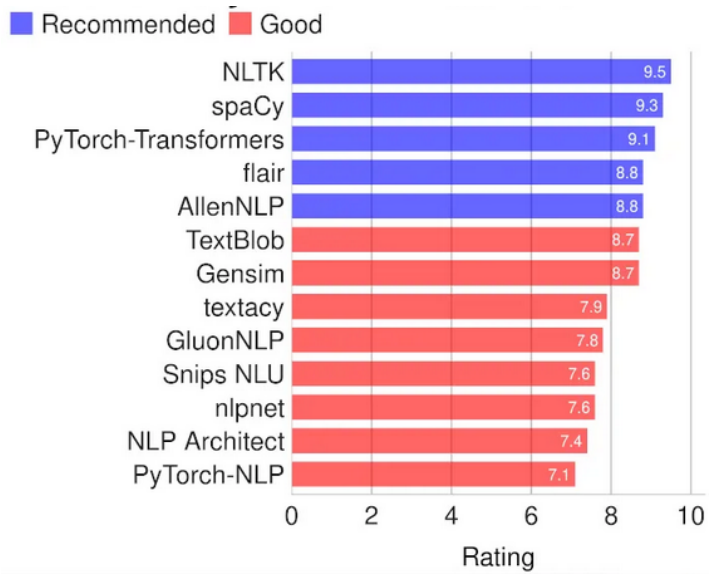


Figure 3.1: NLTK is the most recommended NLP library

```
from nltk.stem import porter
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import sent_tokenize, word_tokenize
```

Figure 3.2: Modules of NLTK used in project

3.2 PyCharm

PyCharm is an Integrated Development Environment (IDE) for Python. It is the most popular Python IDE and is recommended by many experts in this domain. However, there is full freedom to use any other IDE like Spyder, IDLE, Eclipse, PyDev etc.

Figure 3.3: Popular Python IDEs

3.3 Python Standard Library

Python's standard library contains script modules (written in C) which can be accessed by Python programs. It simplifies programming by providing commonly used commands and hence removes the need to rewrite them.

CHAPTER 4

Results

For text summarization, we have used 4 articles titled Artificial Intelligence, Chernobyl Disaster, COVID-19 and Industrial Revolution. These 4 documents are kept in the folder titled 'files'. The program randomly chooses an article and returns the summary.

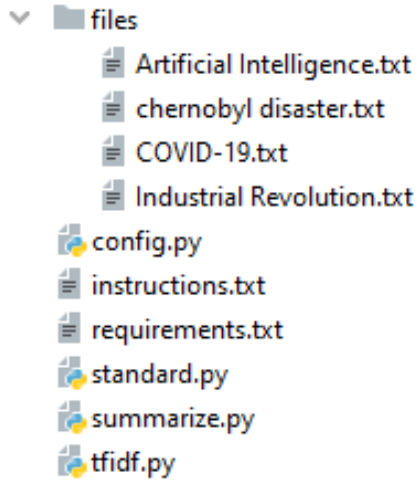


Figure 4.1: Project Files

We have restricted ourselves to POS Tagging of nouns and verbs only. The amount of information retained is set to be 20%. This is done in 'config.py' file.

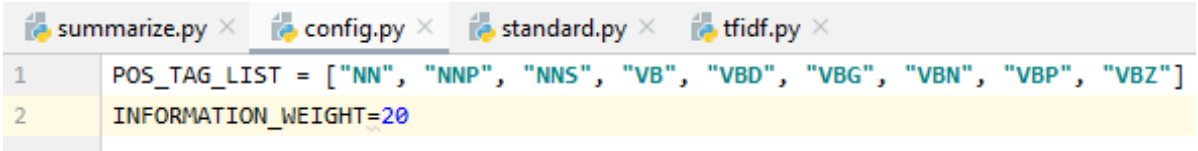


Figure 4.2: config.py

We start the program by running 'summarize.py' file. It randomly chooses COVID-19 article from the 'files' folder. The article contains about 20 sentences and as we are retaining 20% information, we get a summary of 4 sentences.

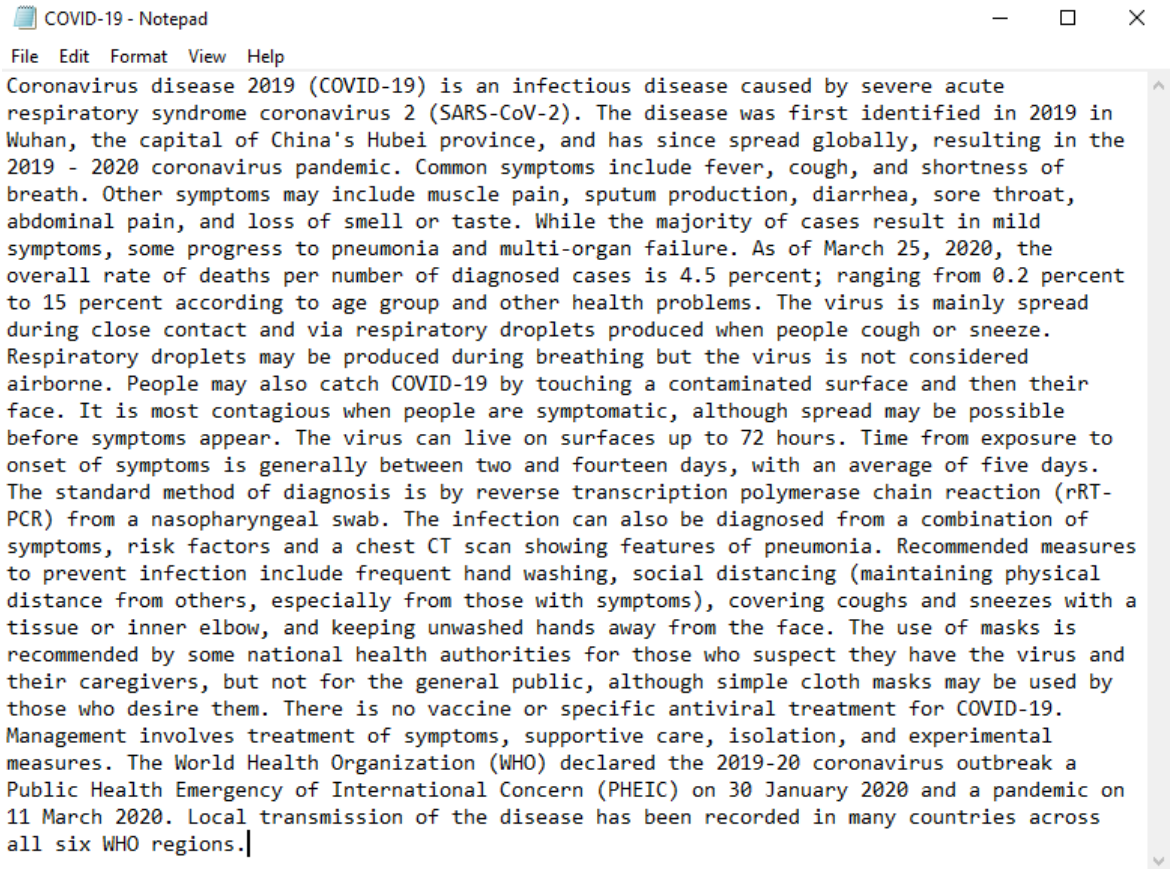


Figure 4.3: Screenshot of Input

```
1 C:\Users\UPLC\AppData\Local\Programs\Python\Python37\python
  .exe C:/Users/UPLC/Downloads/wiki-text-summary/summarize.py
2 [INFO] text: files/COVID-19.txt
3 [INFO] formatting text
4 [INFO] extracting summary
5 [INFO] finished in 6.515 seconds
6 [INFO] percentage of information retained: 20%
7 [INFO] total number of sentences: 4
8 --------------------
9 Summary:
10 Other symptoms may include muscle pain, sputum production,
   diarrhea, sore throat, abdominal pain, and loss of smell or
    taste. While the majority of cases result in mild symptoms
   , some progress to pneumonia and multi-organ failure. As of
    March 25, 2020, the overall rate of deaths per number of
   diagnosed cases is 4.5 percent; ranging from 0.2 percent to
    15 percent according to age group and other health
   problems. The standard method of diagnosis is by reverse
   transcription polymerase chain reaction (rRT-PCR) from a
   nasopharyngeal swab.
11
12 Process finished with exit code 0
```

Figure 4.4: Screenshot of Output

CHAPTER 5

Conclusion

Text Summarization is one of the most fundamental problem of Natural Language Processing. It consumes a lot of time to write summaries manually. Hence, there is need to automate Text Summarization.

The proposed system implements Text Summarization using TF-IDF model. The summary obtained is extractive summary as we have selected text from the article itself. We have obtained a 4 sentence summary of the 20 sentence article as we have set the amount of information retained to be 20%. It took approximately 6.5 seconds to complete the task of text summarization.

5.1 Scope for further improvement

The following points can be considered for making further improvements in the project:

1. We have restricted ourselves to English language. The project can be extended to other popular languages like French, Spanish, Hindi, Mandarin etc.

2. We have restricted ourselves to POS Tagging of nouns and verbs only in order to reduce computation. The project can be extended to POS tagging of all parts of speech but this will come at the cost of increased computation time.

3. We have generated extractive summary using TF-IDF model (NLP based technique). Abstractive summarization using advanced deep learning techniques can be explored.

REFERENCES

[1] D. Jurafsky, *Speech & language processing*. Pearson Education India, 2000.

[2] D. A. Grossman and O. Frieder, *Information retrieval: Algorithms and heuristics*, vol. 15. Springer Science & Business Media, 2012.

[3] https://www.nltk.org/

[4] https://www.wikipedia.org/