Extractive Summarization for English Text

Submitted in partial fulfillment of the requirements of the degree of

Bachelor of Engineering

by

Tadooru Divya Balkrishna (54)

Waghe Akshay Chandrakant (59)

Waghmare Parth Ramkishan (60)

Supervisor:

Prof. Avinash Gondal

Department of Computer Engineering

Watumull Institute of Electronics Engineering and Computer Technology

2019 – 2020

CERTIFICATE

This is to certify that the project entitled **“Extractive Summarization for English Text”** is a bonafide work of

Tadooru Divya Balkrishna (54)

Waghe Akshay Chandrakant (59)

Waghmare Parth Ramkishan (60)

submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of **Undergraduate** in **Bachelor of Computer Engineering**.

(Name and sign) Supervisor/Guide	(Name and sign) Co-Supervisor/Guide
(Name and sign) Head of Department	Dr. Sunita Sharma Principal

Project Report Approval for B. E.

This project report entitled "***Extractive Summarization for English Text***" by

Tadooru Divya Balkrishna (54)

Waghe Akshay Chandrakant (59)

Waghmare Parth Ramkishan (60)

is approved for the degree of **Undergraduate** in **Bachelor of Computer Engineering**.

Examiners

1.--

2.--

Date: / /2019

Place: Ulhasnagar – Thane

Declaration

I/we declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Tadooru Divya Balkrishna (54):

Waghe Akshay Chandrakant (59):

Waghmare Parth Ramkishan (60):

Date: / / 2019

TABLE OF CONTENTS

Table of Figures

Abstract

An approach for generating short and precise summaries for long text documents is proposed. Lately, the size of information on the internet is increasing. It has become tough for the users to dig into the loads of information to analyze it and draw conclusions. Text summarization solves this problem by generating a summary, selecting sentences which are most important from the document without losing the information. In this work, an approach for Extractive text summarization is designed and implemented for single document summarization. It uses a Ranking algorithm to select important sentences from the text still keeping the summary meaningful and lossless. The text documents used for summarization are in English language. Various sentence and word level features are used to provide meaningful sentences. The summaries are generated using Rank Algorithm. Summaries are then combined and processed using a set of operations to get the final summary of the document. The results show that the designed approach overcomes the problem of text overloading by generating an effective summary.

1.Introduction

Earlier, humans used to summarize the text by their own, but today data available on the internet world is massive and increasing day by day. It is very difficult for human to manage and summarize the data by their own. Extractive test summarization overcomes this issue by generating succinct form of large documents keeping most of the original and important information. The result generates effective summary which solve the problem of time consuming and text overloading. Books and Literature: Google has reportedly worked on projects that attempt to understand novels. Summarization can help consumers quickly understand what a book is about as part of their buying process. Financial Research: Investment banking firms spend large amounts of money acquiring information to drive their decision-making, including automated stock trading. Summarization systems tailored to financial documents like earning reports and financial news can help analysts quickly derive market signals from content. Due to tremendous information available on the Internet. It is very difficult for human beings to analyse this information manually, as you can imagine, is really time-consuming. In order to solve the above problem, extractive text summarization is used. Text summarization is the process of identifying the most important meaningful information in a document or set of related documents and compressing them into a shorter version preserving its overall meanings.

2.Aims and Objectives

The length of textual data is increasing and people have less time. Often the newspaper articles run into a long text of, say 1000 -1200 words. As wearable devices leap to prominence (Google Glass, Apple Watch, to name a few), content must adapt to the limited screen space available on these devices. The task of generating intelligent and accurate summaries for long pieces of text has become a popular research as well as industry problem. Extractive text summarization is all about finding the more important sentences from a document as a summary of that document. Our approach is using the PageRank algorithm to find these 'important' sentences.

3.Literature Survey

[1] This paper represents the extractive summarization for English text for a single document. It solves the problem of users to dig into the loads of information to analyse it and draw conclusions, selecting important sentences without losing the information keeping it meaningful. The paper uses the combination of Restricted Boltzmann Machine and Fuzzy Logic, both summarizes are combined and processed using set of operations to get the final result. These two methods is used as an unsupervised learning algorithm to improve the accuracy of the summary.

[2] The paper conceptualizes extractive summarization as a sentence ranking task. It proposes the algorithm which globally optimizes the ROUGE evaluation metric through a reinforcement learning objective. It creates a summary by identifying the most important sentences in a document. In this, main components are a sentence encoder, document decoder and sentence extractor. It explores the space of candidate summaries while learning to optimize a reward function which is relevant for the task at hand.

4.Existing System

4.1 Text Compactor (Free online automatic text summarization tool)

About: This free online summarization tool was created to help struggling readers process overwhelming amounts of information. However, the general approach will help any busy student, teacher and professional.

How it works: After text is placed on the page, the web app calculates the frequency of each word in the passage. Then, a score is calculated for each sentence based on the frequency count associated with the words it contains. The most important sentence is deemed to be the sentence with the highest frequency count. Obviously, human readers may disagree with this automated approach to text summarization. Automated text summarization works best on expository text such as textbooks and reference material. The results can be skewed when a passage has only a few sentences. Text compactor is not recommended for use with fiction. (i.e stories about imaginary people, places, events.)

5. Proposed System

In the proposed method, feature values are calculated for a sentence to get better relevance. Ranking logic are used for generating the summary of the document. The extractive approach involves picking up the most important phrases and lines from the documents. It then combines all the important lines to create the summary. So, in this case, every line and word of the summary actually belongs to the original document which is summarized.

The proposed method for Extractive text summarization is implemented for single document along with the original method it uses TF-IDF and TD-Matrix. Textrank is an algorithm inspired by Google's PageRank algorithm that helps identify key sentences from a passage. The idea behind this algorithm is that the sentence that is similar to most other sentences in the passage is probably the most important sentence in the passage. Using this idea, one can create a graph of sentences connected with all the similar sentences and run Google's PageRank algorithm on it to find the most important sentences. These sentences would then be used to create the summary. Term-Document matrix(TD matrix) is used to convert an collection of text documents to a matrix of token counts fit_transform method of CountVectorizer. Term Frequency-Inverse Document Frequency (TF-IDF) is used to determine the relevance of a word in the document. The underlying algorithm calculates the frequency of the word in the document (term frequency) and multiplies it by the logarithmic function of the number of documents containing that word over the total number of documents in the dataset (inverse document frequency). Using the relevance of each word, one can compute the relevance of each sentence. Assuming that most relevant sentences are the most important sentences, these sentences can then be used to form a summary of the document.

6. Methodology

Following are the steps for Extractive Text Summarization:

1. Input Document

2. Pre-processing

3. Sentence Feature Extraction

4. Logic

5. Final Summary

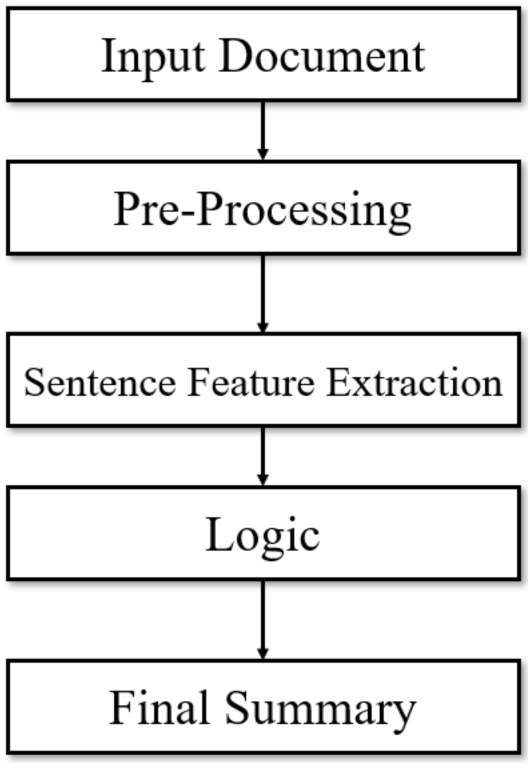


Fig 6

1. Input Document

The very first step in Text Summarization is the input document. The input text document should be in .txt format with the long paragraphs of data contain in it. The data in the text document should be in English language. This text document is imported using the python library.

2. Pre-Processing

The input document imported in the step is get processed in the Pre-Processing step.

There are 3 steps in Pre-Processing-

a. Sentence Segmentation

   In Sentence Segmentation, the whole text is broken down into sentences and stored in an array according to their positions and ranking.

b. Tokenization

   The output of the Sentence Segmentation is the input for the Tokenization. The sentence obtain are further divided and broken down in words for some calculations.

c. Stop Word and Punctuation removal

   The output of Tokenization is the input of this step. As the name suggest, in this step the commonly occurring words such as the, an, a, but, and, or are getting removed along with all the punctuations.

3. Sentence Feature Extraction

In the Sentence Feature Extraction, the sentence feature is getting calculated to find the sentence score. There are many factors which contribute to decide the sentence score.

a. Sentence Position

b. Sentence Length

c. Numerical Token

d. Term Frequency- Inverse Document Frequency (TF-IDF)

e. CountVectorizer

4. Logic

a. Ranking Algorithm:

1. Page Rank:

PageRank (Brin and Page, 1998) is perhaps one of the most popular ranking algorithms, and was designed as a method for Web link analysis. Unlike other ranking algorithms, PageRank integrates the impact of both incoming and outgoing links into one single model, and therefore it produces only one set of scores:

$$PR(Vi) = (1 - d) + d \ast X\ Vj \in In(Vi)\ PR(Vj)\ |Out(Vj)|$$

where d is a parameter that is set between 0 and 1 1 . For each of these algorithms, starting from arbitrary values assigned to each node in the graph, the computation iterates until convergence below a given threshold is achieved. After running the algorithm, a score is associated with each vertex, which represents the “importance” or “power” of that vertex within the graph. Notice that the final values are not affected by the choice of the initial value, only the number of iterations to convergence may be different.

2. Text Rank:

It is a graph-based unsupervised method for keyword and sentence extraction”. Because it is an unsupervised ranking method, it does require any training and supervised data. Graph-based ranking algorithms are essentially a way of deciding the importance of a vertex within a graph, based on information drawn from the graph structure. In this section, we present three graph-based ranking algorithms – previously found to be successful on a range of ranking problems. We also show how these algorithms can be adapted to undirected or weighted graphs, which are particularly useful in the context of text-based ranking applications.

Let G = (V, E) be a directed graph with the set of vertices V and set of edges E, where E is a subset of V × V . For a given vertex Vi , let In(Vi) be the set of vertices that point to it (predecessors), and let Out(Vi) be the set of vertices that vertex Vi points to (successors).

5. **Final Summary**

   After performing the above steps, the output is the final summary of the input text document.

7.Requirements Software and Hardware

7.1 Software Requirements

Programming Language – Python 3.6 and above with Jupyter Notebook as a platform. Python is developed under an OSI-approved open source license, making it freely usable and distributable, even for commercial use. Project Jupyter is a non-profit, open source project, born out of the IPython Project in 2014 as it evolved to support interactive data science and scientific computing across all programming languages.

- **UI and File Import** – Tkinter Python Library.
  The Tkinter module ("Tk interface") is the standard Python interface to the Tk GUI toolkit. Both Tk and Tkinter are available on most Unix platforms, as well as on Windows systems. (Tk itself is not part of Python: it is maintained at ActiveState.)
- **Pre-processing** – Natural Language Toolkit.
  The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing for English written in the Python programming language.

7.2 Hardware Requirements

- The hardware will not required a high-performance CPU and GPU to perform the complex operations in the software model. Minimum requirements:
  - ✓ CPU: intel i3 @ 2.0 GHz processor
  - ✓ RAM: 4GB RAM

8. Design Details

8.1GUI Model

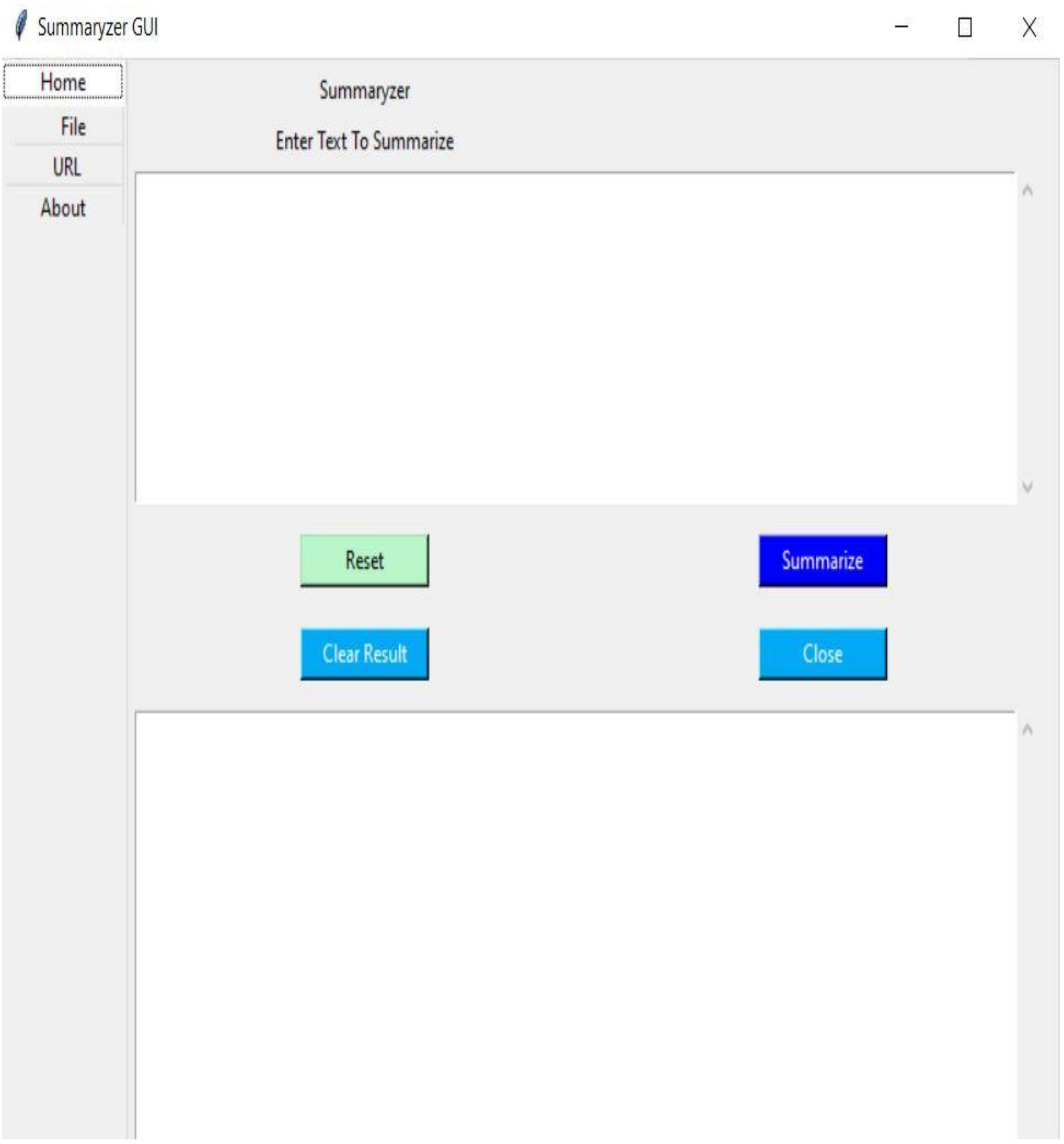


Fig 8.1.1

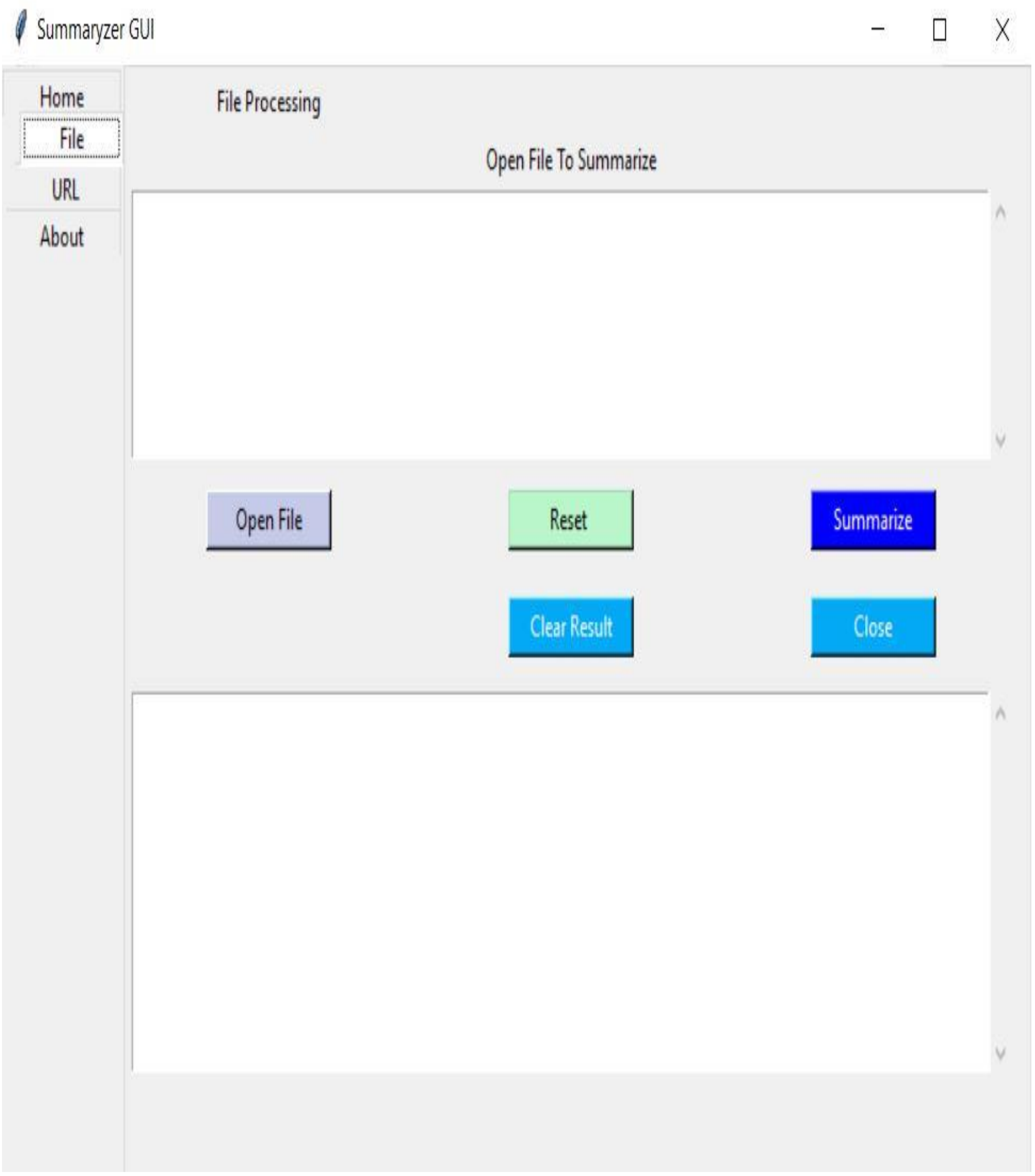

Summaryzer GUI
Home
File
URL
About
File Processing
Open File To Summarize
Open File
Reset
Summarize
Clear Result
Close

Fig 8.1.2

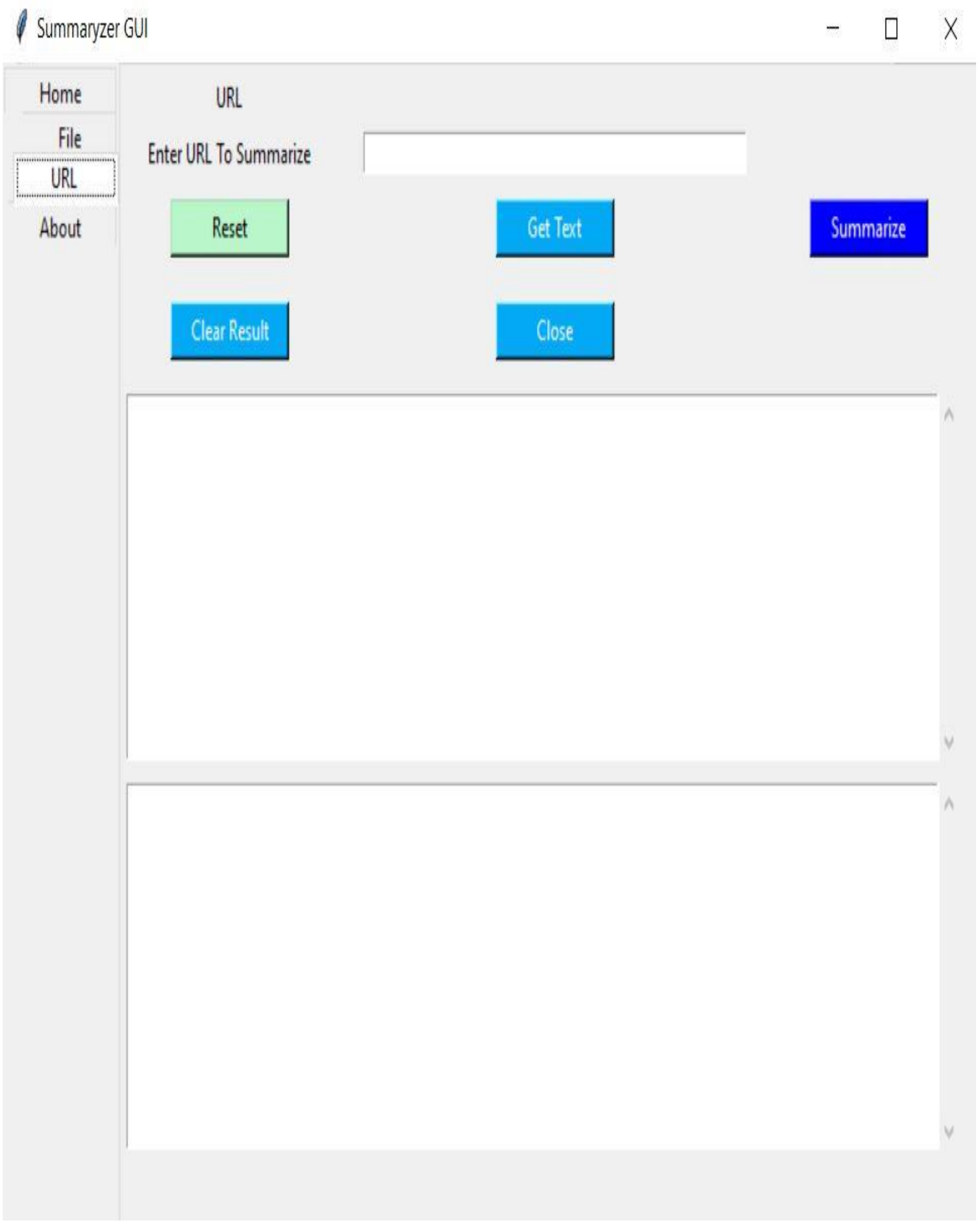

Summaryzer GUI
Home
File
URL
About
URL
Enter URL To Summarize
Reset
Get Text
Summarize
Clear Result
Close

Fig 8.1.3

8.2 Flow Diagram

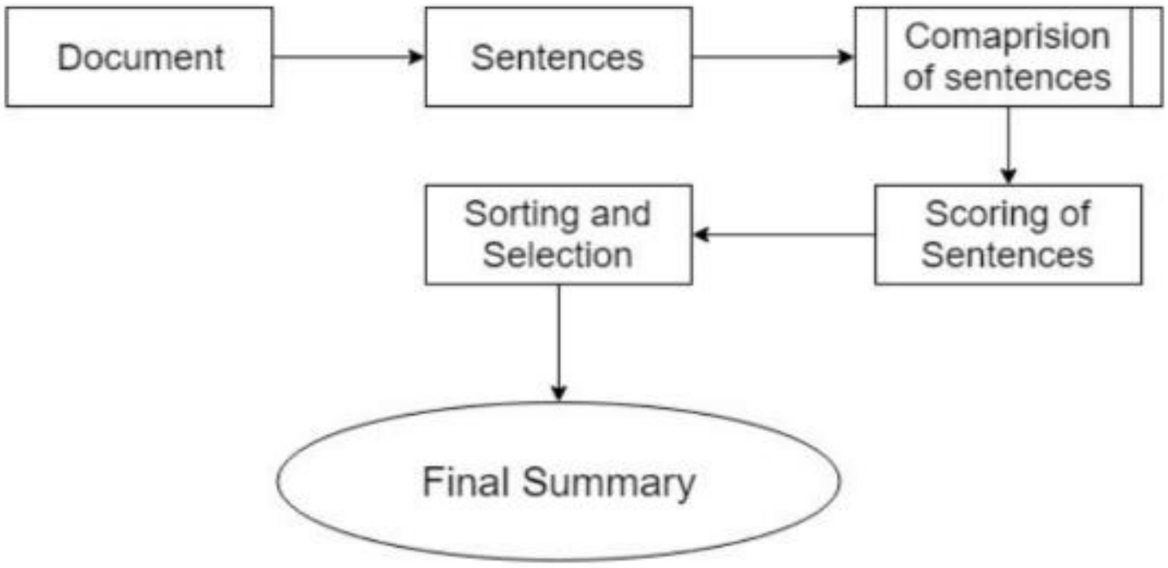


Fig 8.2

8.3 Usecase Diagram

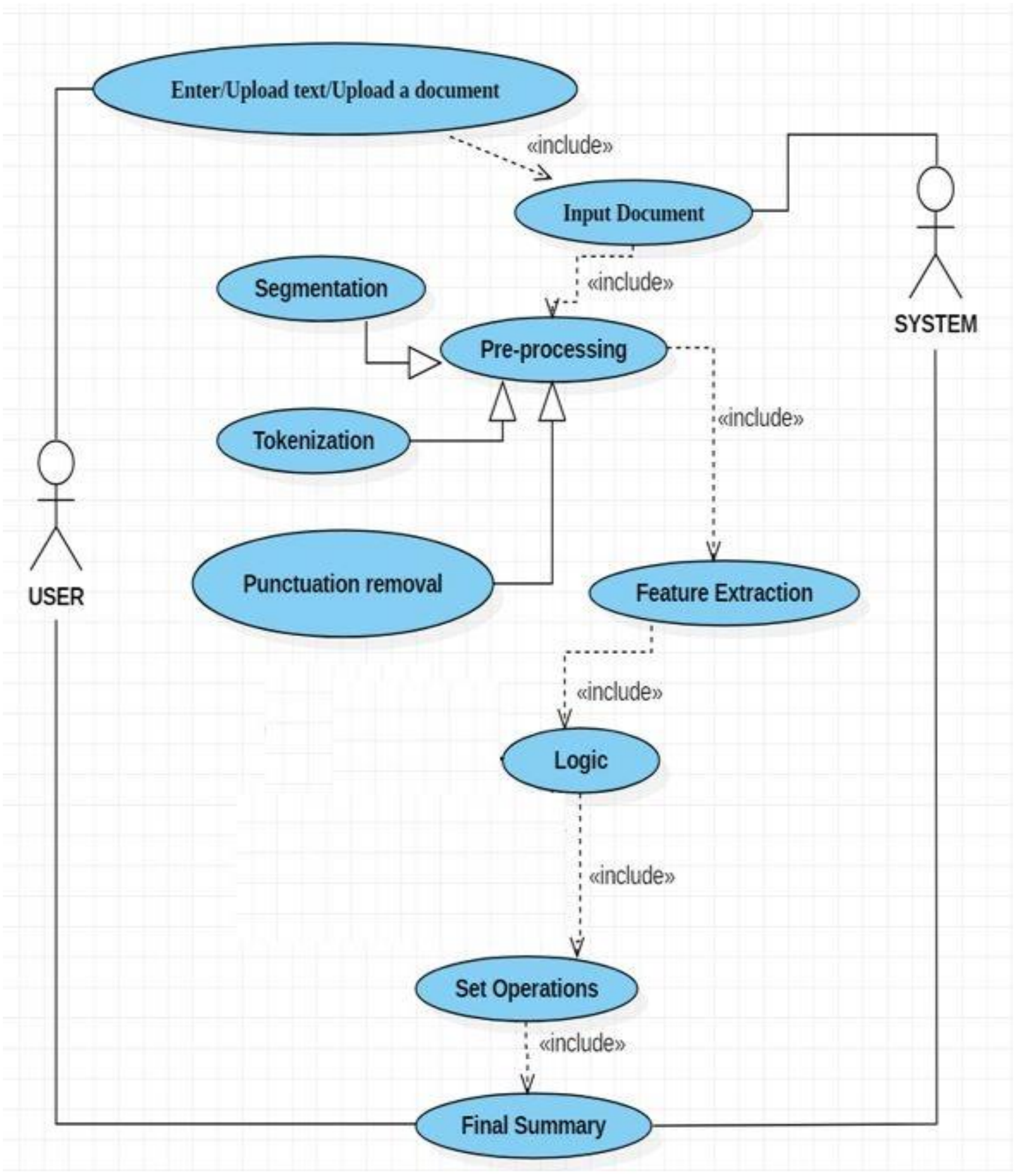
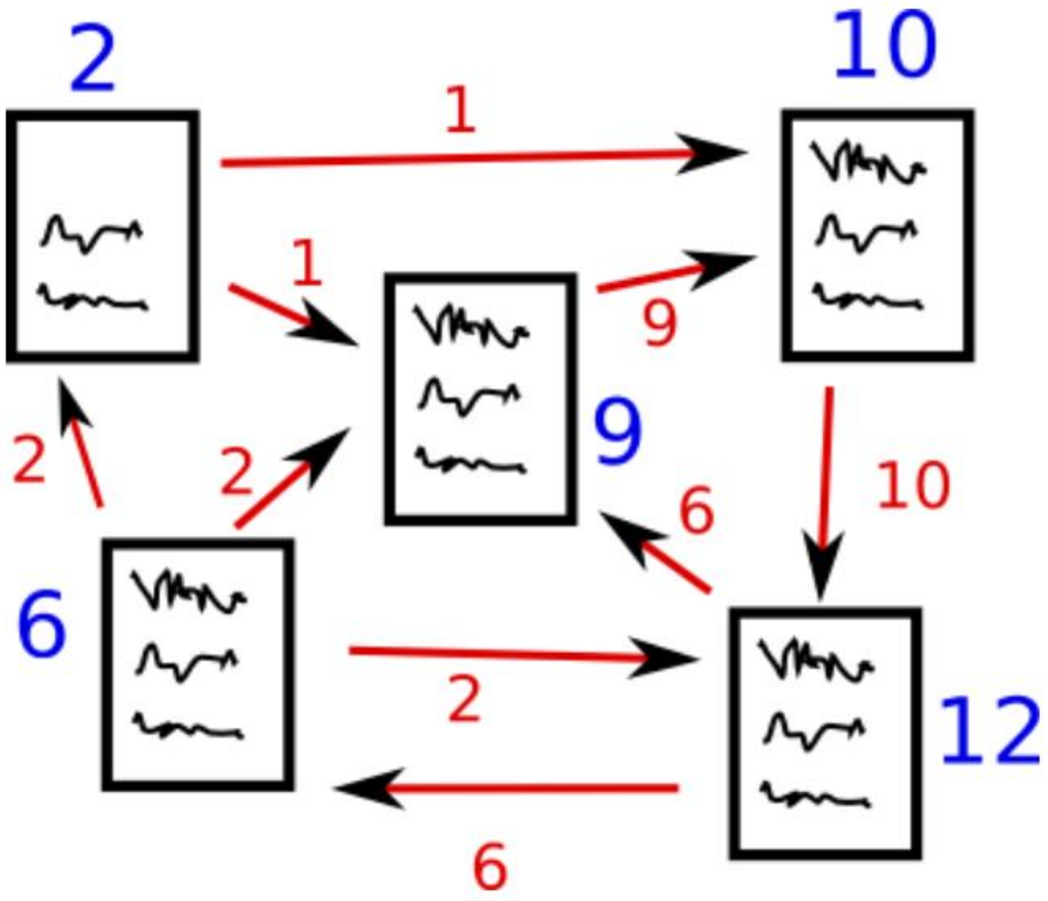


Fig 8.3

9. Implementation

9.1 Algorithm

a. PageRank



Suppose we have 4 web pages — w1, w2, w3, and w4. These pages contain links pointing to one another. Some pages might have no link – these are called dangling pages.

webpage	links
w1	[w4, w2]
w2	[w3, w1]
w3	[ ]
w4	[w1]

- Web page w1 has links directing to w2 and w4
- w2 has links for w3 and w1
- w4 has links only for the web page w1
- w3 has no links and hence it will be called a dangling page

In order to rank these pages, we would have to compute a score called the PageRank score. This score is the probability of a user visiting that page.

To capture the probabilities of users navigating from one page to another, we will create a square matrix M, having n rows and n columns, where n is the number of web pages.

$$\mathbf{M} = \begin{array}{c|c|c|c|c|} & \mathbf{w1} & \mathbf{w2} & \mathbf{w3} & \mathbf{w4} \\ \hline \mathbf{w1} & & & & \\ \hline \mathbf{w2} & & & & \\ \hline \mathbf{w3} & & & & \\ \hline \mathbf{w4} & & & & \\ \hline \end{array}$$

Each element of this matrix denotes the probability of a user transitioning from one web page to another. For example, the highlighted cell below contains the probability of transition from w1 to w2.

$$\mathbf{M} = \begin{array}{c|c|c|c|c|} & \mathbf{w1} & \mathbf{w2} & \mathbf{w3} & \mathbf{w4} \\ \hline \mathbf{w1} & & 🟨 & & \\ \hline \mathbf{w2} & & & & \\ \hline \mathbf{w3} & & & & \\ \hline \mathbf{w4} & & & & \\ \hline \end{array} \qquad \mathbf{P(w1\ to\ w2)}$$

The initialization of the probabilities is explained in the steps below:

1. Probability of going from page i to j, i.e., M[i][j], is initialized with 1/(number of unique links in web page wi)
2. If there is no link between the page i and j, then the probability will be initialized with 0
3. If a user has landed on a dangling page, then it is assumed that he is equally likely to transition to any page. Hence, M[i][j] will be initialized with 1/(number of web pages)

Hence, in our case, the matrix M will be initialized as follows:

$$
\mathbf{M} = \begin{array}{c|c|c|c|c|}
 & \mathbf{w1} & \mathbf{w2} & \mathbf{w3} & \mathbf{w4} \\
\hline
\mathbf{w1} & 0 & 0.5 & 0 & 0.5 \\
\hline
\mathbf{w2} & 0.5 & 0 & 0.5 & 0 \\
\hline
\mathbf{w3} & 0.25 & 0.25 & 0.25 & 0.25 \\
\hline
\mathbf{w4} & 1 & 0 & 0 & 0 \\
\hline
\end{array}
$$

Finally, the values in this matrix will be updated in an iterative fashion to arrive at the web page rankings.

b. TextRank

Let’s understand the TextRank algorithm, now that we have a grasp on PageRank. I have listed the similarities between these two algorithms below:

- In place of web pages, we use sentences
- Similarity between any two sentences is used as an equivalent to the web page transition probability
- The similarity scores are stored in a square matrix, similar to the matrix M used for PageRank

TextRank is an extractive and unsupervised text summarization technique. Let’s take a look at the flow of the TextRank algorithm that we will be following:

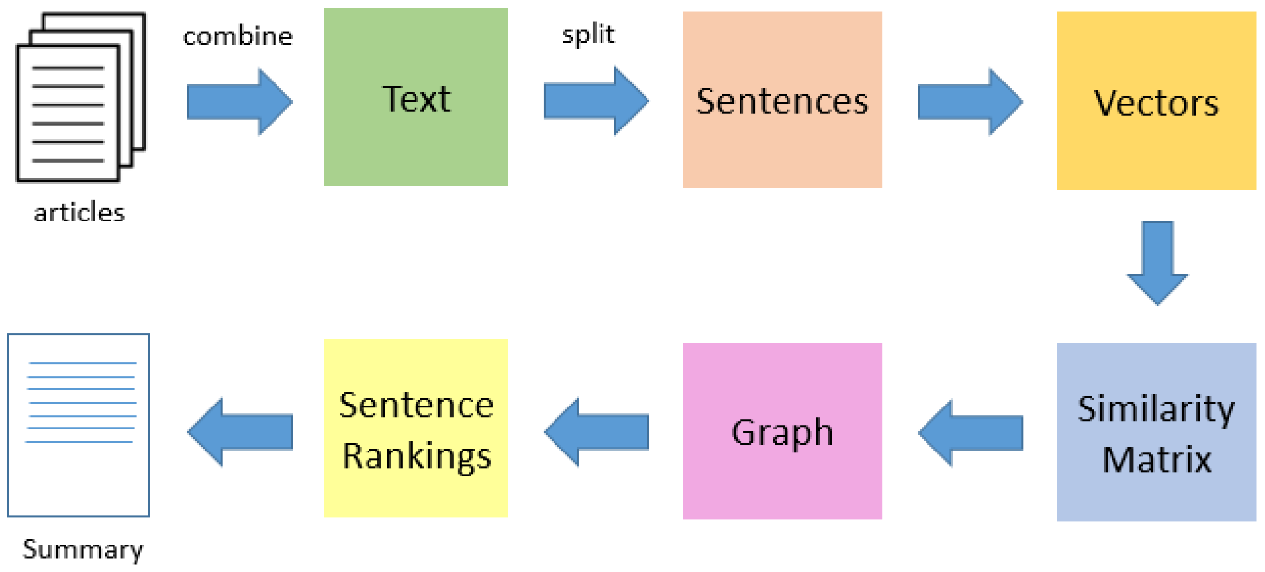


Fig 9.1.b

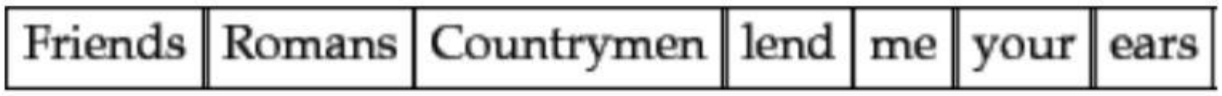
- The first step would be to concatenate all the text contained in the articles.
- Then split the text into individual sentences.
- In the next step, we will find vector representation (word embeddings) for each and every sentence.
- Similarities between sentence vectors are then calculated and stored in a matrix.
- The similarity matrix is then converted into a graph, with sentences as vertices and similarity scores as edges, for sentence rank calculation.
- Finally, a certain number of top-ranked sentences form the final summary.

c. Tokenization

Given a character sequence and a defined document unit, tokenization is the task of chopping it up into pieces, called tokens, perhaps at the same time throwing away certain characters, such as punctuation. Here is an example of tokenization:

Input: Friends, Romans, Countrymen, lend me your ears;

Output:



Friends	Romans	Countrymen	lend	me	your	ears

A token is an instance of a sequence of characters in some particular document that are grouped together as a useful semantic unit for processing. A type is the class of all tokens containing the same character sequence. A term is a (perhaps normalized) type that is included in the IR system's dictionary. It is used to tokenize the sentence into words.

d. Removal of StopWords

A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query. In computing, stop words are words which are filtered out before processing of natural language data (text) Stop words are generally the most common words in a language; there is no single universal list of stop words used by all natural language processing tools, and indeed not all tools even use such a list. Some tools avoid removing stop words to support phrase search. Removal of stop words is therefore used in Hot Topic Analysis which is a way to hide particular words or phrases from appearing in the analysis.

9.2 Working

a. Front End

```
import tkinter as tk

from tkinter import *

from tkinter import ttk

from tkinter.scrolledtext import *

import tkinter.filedialog



# NLP Pkgs

from nltk_summarization import nltk_summarizer



# Web Scraping Pkg

from bs4 import BeautifulSoup

from urllib.request import urlopen



 # Structure and Layout

window = Tk()

window.title("Summaryzer GUI")

window.geometry("700x400")

window.config(background='black')

style = ttk.Style(window)

style.configure('lefttab.TNotebook', tabposition='wn',)
```

```
# TAB LAYOUT

tab_control = ttk.Notebook(window,style='lefttab.TNotebook')

tab1 = ttk.Frame(tab_control)

tab2 = ttk.Frame(tab_control)

tab3 = ttk.Frame(tab_control)

tab4 = ttk.Frame(tab_control)


# ADD TABS TO NOTEBOOK

tab_control.add(tab1, text=f'{"Home":^20s}')

tab_control.add(tab2, text=f'{"File":^20s}')

tab_control.add(tab3, text=f'{"URL":^20s}')

tab_control.add(tab4, text=f'{"About ":^20s}')


label1 = Label(tab1, text= 'Summaryzer',padx=5, pady=5)

label1.grid(column=0, row=0)


label2 = Label(tab2, text= 'File Processing',padx=5, pady=5)

label2.grid(column=0, row=0)

label3 = Label(tab3, text= 'URL',padx=5, pady=5)

label3.grid(column=0, row=0)


label4 = Label(tab4, text= 'About',padx=5, pady=5)
```

```
label4.grid(column=0, row=0)

tab_control.pack(expand=1, fill='both')


# Functions

def get_summary():

	raw_text = str(entry.get('1.0',tk.END))

	final_text = nltk_summarizer(raw_text)

	print(final_text)

	result = '\nSummary:{}'.format(final_text)

	tab1_display.insert(tk.END,result)


# Clear Function

def clear_text():

	entry.delete('1.0',END)


def clear_display_result():

	tab1_display.delete('1.0',END)

# Clear Text  with position 1.0

def clear_text_file():

	displayed_file.delete('1.0',END)


# Clear Result of Functions
```

```
def clear_text_result():
	tab2_display_text.delete('1.0',END)


# Clear For URL
def clear_url_entry():
	url_entry.delete(0,END)


def clear_url_display():
	tab3_display_text.delete('1.0',END)
# Clear entry widget
def clear_compare_text():
	entry1.delete('1.0',END)


def clear_compare_display_result():
	tab1_display.delete('1.0',END)


# Functions for TAB 2 FILE PROCESSER
# Open File to Read and Process
def openfiles():
	file1 = tkinter.filedialog.askopenfilename(filetypes=(("Text
Files",".txt"),("All files","*")))
	read_text = open(file1).read()
```

```
        displayed_file.insert(tk.END,read_text)


def get_file_summary():
        raw_text = displayed_file.get('1.0',tk.END)
        final_text = nlkt_summarizer(raw_text)
        result = '\nSummary:{}'.format(final_text)
        tab2_display_text.insert(tk.END,result)


#URL Functions
# Fetch Text From Urls
def get_text():
        raw_text = str(url_entry.get())
        page = urlopen(raw_text)
        soup = BeautifulSoup(page,'html.parser')
        fetched_text = ' '.join(map(lambda p:p.text,soup.find_all('p')))
        url_display.insert(tk.END,fetched_text)
def get_url_summary():
        raw_text = url_display.get('1.0',tk.END)
        final_text = nlkt_summarizer(raw_text)
        result = '\nSummary:{}'.format(final_text)
        tab3_display_text.insert(tk.END,result)
```

```
# MAIN HOME TAB

l1=Label(tab1,text="Enter Text To Summarize")

l1.grid(row=1,column=0)


entry=ScrolledText(tab1,height=10)

entry.grid(row=2,column=0,columnspan=2,padx=5,pady=5)


#Buttons

button1=Button(tab1,text="Reset",command=clear_text,
width=12,bg='#b9f6ca')

button1.grid(row=4,column=0,padx=10,pady=10)


button2=Button(tab1,text="Summarize",command=get_summary,
width=12,bg='blue',fg='#fff')

button2.grid(row=4,column=1,padx=10,pady=10)


button3=Button(tab1,text="Clear Result",
command=clear_display_result,width=12,bg='#03A9F4',fg='#fff')

button3.grid(row=5,column=0,padx=10,pady=10)


button4=Button(tab1,text="Close",
width=12,command=window.destroy,bg='#03A9F4',fg='#fff')

button4.grid(row=5,column=1,padx=10,pady=10)
```

```
# Display Screen For Result

tab1_display = ScrolledText(tab1)

tab1_display.grid(row=7,column=0, columnspan=3,padx=5,pady=5)


#FILE PROCESSING TAB

l1=Label(tab2,text="Open File To Summarize")

l1.grid(row=1,column=1)


displayed_file = ScrolledText(tab2,height=7)# Initial was Text(tab2)

displayed_file.grid(row=2,column=0, columnspan=3,padx=5,pady=3)


# BUTTONS FOR SECOND TAB/FILE READING TAB

b0=Button(tab2,text="Open File", width=12,command=openfiles,bg='#c5cae9')

b0.grid(row=3,column=0,padx=10,pady=10)

b1=Button(tab2,text="Reset ",
width=12,command=clear_text_file,bg="#b9f6ca")

b1.grid(row=3,column=1,padx=10,pady=10)


b2=Button(tab2,text="Summarize",
width=12,command=get_file_summary,bg='blue',fg='#fff')

b2.grid(row=3,column=2,padx=10,pady=10)
```

```
b3=Button(tab2,text="Clear Result",
width=12,command=clear_text_result,bg='#03A9F4',fg='#fff')

b3.grid(row=5,column=1,padx=10,pady=10)


b4=Button(tab2,text="Close",
width=12,command=window.destroy,bg='#03A9F4',fg='#fff')

b4.grid(row=5,column=2,padx=10,pady=10)


# Display Screen

# tab2_display_text = Text(tab2)

tab2_display_text = ScrolledText(tab2,height=10)

tab2_display_text.grid(row=7,column=0, columnspan=3,padx=5,pady=5)


# Allows you to edit

tab2_display_text.config(state=NORMAL)

# URL TAB

l1=Label(tab3,text="Enter URL To Summarize")

l1.grid(row=1,column=0)


raw_entry=StringVar()

url_entry=Entry(tab3,textvariable=raw_entry,width=50)

url_entry.grid(row=1,column=1)
```

```
# BUTTONS

button1=Button(tab3,text="Reset",command=clear_url_entry,
width=12,bg='#b9f6ca')

button1.grid(row=4,column=0,padx=10,pady=10)



button2=Button(tab3,text="GetText",command=get_text,
width=12,bg='#03A9F4',fg='#fff')

button2.grid(row=4,column=1,padx=10,pady=10)



button3=Button(tab3,text="Clear Result",
command=clear_url_display,width=12,bg='#03A9F4',fg='#fff')

button3.grid(row=5,column=0,padx=10,pady=10)



button4=Button(tab3,text="Summarize",command=get_url_summary,
width=12,bg='blue',fg='#fff')

button4.grid(row=4,column=2,padx=10,pady=10)



button4=Button(tab3,text="Close",
width=12,command=window.destroy,bg='#03A9F4',fg='#fff')

button4.grid(row=5,column=1,padx=10,pady=10)



# Display Screen For Result

url_display = ScrolledText(tab3,height=10)
```

```
url_display.grid(row=7,column=0, columnspan=3,padx=5,pady=5)


tab3_display_text = ScrolledText(tab3,height=10)

tab3_display_text.grid(row=10,column=0, columnspan=3,padx=5,pady=5)


# About TAB

about_label = Label(tab4,text="Summaryzer GUI V.0.0.1 \n",pady=5,padx=5)

about_label.grid(column=0,row=1)


window.mainloop()
```

b. Back End

```
import numpy as np
import PyPDF2
import sys
import matplotlib.pyplot as plt
import networkx as nx
from sklearn.feature_extraction.text import TfidfTransformer, CountVectorizer

def tokenize(document):
    doc_tokenizer = PunktSentenceTokenizer()
    sentences_list = doc_tokenizer.tokenize(document)
    return sentences_list

# ###  Read the document

document = readDoc()
print('The length of the file is:', end=' ')
print(len(document))

# ### Generate a list of sentences in the document
sentences_list = tokenize(document)

# let us print the size of memory used by the list sentences

print('The size of the list in Bytes is: {}'.format(sys.getsizeof(sentences_list)))

# the size of one of the element of the list

print('The size of the item 0 in Bytes is{}'.format(sys.getsizeof(sentences_list[0])))
```

```
# let us see the data type of sentences_list
# It will be list

print(type(sentences_list))

# let us analyse the elements of the sentences
# len() method applies on the list and provides the number of elements in the list

print('The size of the list "sentences" is: {}'.format(len(sentences_list)))

# print the elements of the list
# If the input document is long, which on realistically will be wrong, we would not
like to print the entire document

for i in sentences_list:
    print(i)

# ### Generate term-document matrix (TD matrix) of the data
# Convert a collection of text documents to a matrix of token counts
# fit_transform method of CountVectorizer() class
# Learn the vocabulary dictionary and return term-document matrix.
# I/p: An iterable which yields either str, unicode or file objects.
# O/p: The term-document matrix named cv_matrix

cv = CountVectorizer()
cv_matrix = cv.fit_transform(sentences_list)
```

```
# **So what does CountVectorizer.fit_transform() do?**
# a demo of what CountVectorizer().fit_transform(text) does

cv_demo = CountVectorizer()

# a demo object of class CountVectorizer
# I have repeated the words to make a non-ambiguous array of the document text
matrix

text_demo = ["Ashish is good, you are bad", "I am not bad"]
res_demo = cv_demo.fit_transform(text_demo)
print('Result demo array is {}'.format(res_demo.toarray()))

# Result is 2-d matrix containing document text matrix
# Notice that in the second row, there is 2.
# also, bad is repeated twice in that sentence.
# so we can infer that 2 is corresponding to the word 'bad'

print('Feature list: {}'.format(cv_demo.get_feature_names()))



# printing the cv_matrix type
# and how it is being stored in memory?
# it is stored in the compressed row format
# compressed row format:

print('The data type of bow matrix {}'.format(type(cv_matrix)))
print('Shape of the matrix {}'.format(cv_matrix.get_shape))
print('Size of the matrix is: {}'.format(sys.getsizeof(cv_matrix)))
print(cv.get_feature_names())
print(cv_matrix.toarray())



# Tnormalized: document-term matrix normalized (value 0-1) according to the TF-
IDF
# TF(Term Frequency): the no. of times a term(a word here) appears in the current
document(single sentence here)
# IDF(Inverse Document Frequency): the no. of times a term(a word here) appears
in the entire corpus
# Corpus: set of all sentences

normal_matrix = TfidfTransformer().fit_transform(cv_matrix)
print(normal_matrix.toarray())
print(normal_matrix.T.toarray)
res_graph = normal_matrix * normal_matrix.T
# plt.spy(res_graph)



# ### Generate a graph for the document to apply PageRank algorithm
# drawing a graph to proceed for the textrank algorithm
# nx_graph is a graph developed using the networkx library
# each node represents a sentence
# an edge represents that they have words in common
# the edge weight is the number of words that are common in both of the
sentences(nodes)
# nx.draw() method is used to draw the graph created

nx_graph = nx.from_scipy_sparse_matrix(res_graph)
```

```
nx.draw_circular(nx_graph)
print('Number of edges {}'.format(nx_graph.number_of_edges()))
print('Number of vertices {}'.format(nx_graph.number_of_nodes()))
# plt.show()
print('The memory used by the graph in Bytes is:
{}'.format(sys.getsizeof(nx_graph)))

# ###  Getting the rank of every sentence using pagerank
# ranks is a dictionary with key=node(sentences) and value=textrank (the rank of
each of the sentences)
ranks = nx.pagerank(nx_graph)

# analyse the data type of ranks
print(type(ranks))
print('The size used by the dictionary in Bytes is: {}'.format(sys.getsizeof(ranks)))

# print the dictionary
for i in ranks:
    print(i, ranks[i])

# ### Finding important sentences and generating summary
# enumerate method: returns an enumerate object
# Use of list Comprehensions
# O/p: sentence_array is the sorted(descending order w.r.t. score value) 2-d array
of ranks[sentence] and sentence
# For example, if there are two sentences: S1 (with a score of S1 = s1) and S2 with
score s2, with s2>s1
# then sentence_array is [[s2, S2], [s1, S1]]
```

```
sentence_array = sorted(((ranks[i], s) for i, s in enumerate(sentences_list)),
reverse=True)
sentence_array = np.asarray(sentence_array)

# as sentence_array is in descending order wrt score value
# fmax is the largest score value(the score of first element)
# fmin is the smallest score value(the score of last element)

rank_max = float(sentence_array[0][0])
rank_min = float(sentence_array[len(sentence_array) - 1][0])

# print the largest and smallest value of scores of the sentence
print(rank_max)
print(rank_min)

# Normalization of the scores
# so that it comes out in the range 0-1
# fmax becomes 1
# fmin becomes 0
# store the normalized values in the list temp_array

temp_array = []

# if all sentences have equal ranks, means they are all the same
# taking any sentence will give the summary, say the first sentence
flag = 0
if rank_max - rank_min == 0:
    temp_array.append(0)
    flag = 1
```

```
# If the sentence has different ranks
if flag != 1:
    for i in range(0, len(sentence_array)):
        temp_array.append((float(sentence_array[i][0]) - rank_min) / (rank_max - 
rank_min))

print(len(temp_array))



# Calculation of threshold:
# We take the mean value of normalized scores
# any sentence with the normalized score 0.2 more than the mean value is 
considered to be

threshold = (sum(temp_array) / len(temp_array)) + 0.2

# Separate out the sentences that satiasfy the criteria of having a score above the 
threshold
sentence_list = []
if len(temp_array) > 1:
    for i in range(0, len(temp_array)):
        if temp_array[i] > threshold:
                sentence_list.append(sentence_array[i][1])
else:
    sentence_list.append(sentence_array[0][1])

model = sentence_list

# ### Writing the summary to a new file
# print(sentence_list)

summary = " ".join(str(x) for x in sentence_list)
print(summary)

# save the data in another file, names sum.txt
f = open('sum.txt', 'a+')

#print(type(f))
f.write('-------------------\n')
f.write(summary)
f.write('\n')

f.close
```

10. Testing

10.1 Test Case

Test Case 1 –

Input- Narendra Modi met Donald Trump and they had a long meeting discussing on various future relations between India and America. They decided to improve the financial trade between the two countries.

Output- Narendra Modi met Donald Trump. They decided to improve financial trade.

Test Case 2 –

Input- The Citizenship Amendment Act (Bill) protests, also known as CAA Protest or CAB Protest, occurred after the Citizenship Amendment Act (CAA) was enacted by the Government of India on 12 December 2019. The move sparked a widespread national and overseas ongoing protests against the act and its associated proposals of the National Register of Citizens (NRC). The protests first began in Assam and spread swiftly in other states such as Delhi, Meghalaya, Arunachal Pradesh, and Tripura on 4 December 2019. Protests broke out rapidly across the country, although the concerns of the protesters vary.

The CAA amends the Indian citizenship to illegal migrants who are Hindu, Sikh, Jain, Parsi, Buddhist, and Christian from Afghanistan, Bangladesh and Pakistan, and who entered India before 2014 following the religious persecutions. The bill does not mention Muslims and other communities who fled from the same or other neighbouring countries. Refugees from Sri Lankan Tamils in India, Rohingyas from Myanmar, and Tibetan refugees are not mentioned in the bill. The proposed National Register of Citizens (NRC) will be an official record of all legal citizens of India. Individuals would need to provide a prescribed set of documents before a specified cut off date to be included in it.

Output- The move sparked a widespread national and overseas ongoing protests against the act and its associated proposals of the National Register of Citizens (NRC). The protests first began in Assam and spread swiftly in other states such as Delhi, Meghalaya, Arunachal Pradesh, and Tripura on 4 December 2019. Protests broke out rapidly across the country, although the concerns of the protesters vary. The CAA amends the Indian citizenship to illegal migrants who are Hindu, Sikh, Jain, Parsi, Buddhist, and Christian from Afghanistan, Bangladesh and Pakistan, and who entered India before 2014 following the religious persecutions. The bill does not mention Muslims and other communities who fled from the same or other neighbouring countries.

Test Case 3 –

Input- Video-sharing app Mitron, which has recently been gaining steam as the rival to the popular Chinese app TikTok, is not really 'made in India', according to a report by News18.

The Mitron app had garnered a lot of attention amid the growing clamour for 'Make-in-India' and 'vocal for local'. However, the report says that the app was never really made by an IIT student in the country. Instead, its entire source code, including its features and user interface, was bought from Qboxus, a Pakistani software developer company.

The founder of Qboxus, Irfan Sheikh, told News18 that the company had sold the source code to promoters of the Mitron app for around Rs 2,600. He added that while Qboxus expects customers to use the code and build their own product, Mitron’s developers have taken our exact product, changed the logo and uploaded it on their store.

It had earlier been reported that the Mitron App, which has clocked over 5 million downloads on Google Play Store, was developed by a student of IIT Roorkee and that it works exactly like its Chinese rival TikTok.

Output- Instead, its entire source code, including its features and user interface, was bought from Qboxus, a Pakistani software developer company.

The founder of Qboxus, Irfan Sheikh, told News18 that the company had sold the source code to promoters of the Mitron app for around Rs 2,600. The Mitron app had garnered a lot of attention amid the growing clamour for 'Make-in-India' and 'vocal for local'. However, the report says that the app was never really made by an IIT student in the country.

Test Case 4 –

Input- There are times when the night sky glows with bands of color. The bands may begin as cloud shapes and then spread into a great arc across the entire sky. They may fall in folds like a curtain drawn across the heavens. The lights usually grow brighter, then suddenly dim. During this time the sky glows with pale yellow, pink, green, violet, blue, and red. These lights are called the Aurora Borealis. Some people call them the Northern Lights. Scientists have been watching them for hundreds of years. They are not quite sure what causes them. In ancient times people were afraid of the Lights. They imagined that they saw fiery dragons in the sky. Some even concluded that the heavens were on fire.

Output-The Aurora Borealis, or Northern Lights, are bands of color in the night sky. Ancient people thought that these lights were dragon on fire, and even modern scientists are not sure what they are.

Test Case 5 –

Input- As today's bride and groom celebrate their wedding, they have every excuse for being nervous. They exchange promises of lifelong fidelity and mutual support. However, all around them, they can see that many people do not and cannot keep these promises. Their own marriage has a one in three chance of divorce, if present tendencies continue.Traditional marriage is facing a crisis, at least in Britain. Not only are there more and more divorces, but the number of marriages is falling. Living together is more popular than before. The family is now no longer one man, one woman and their children. Instead, there are more and more families which include parents, half sisters and brothers, or even only one parent on her / his own. Although Britain is still conservative in its attitudes to marriage compared with other countries such as the USA, Sweden and Denmark, the future will probably see many more people living together before marriage - and more divorce. Interestingly, it is women rather than men who apply for divorce. Seven out of ten divorces are given to the wife. Also, one of the main reasons for divorce, chosen by ten times more women than men, is unreasonable or cruel behaviour. Perhaps this means that women will tolerate less than they used to.

Output-Although Britain is still conservative in its attitudes to marriage compared with other countries such as the USA, Sweden and Denmark, the future will probably see many more people living together before marriage - and more divorce. Their own marriage has a one in three chance of divorce, if present tendencies continue. Also, one of the main reasons for divorce, chosen by ten times more women than men, is unreasonable or cruel behaviour. Interestingly, it is women rather than men who apply for divorce. Traditional marriage is facing a crisis, at least in Britain.

Test Case 6 –

Input- Universities, like cathedrals and parliaments, are a product of the Middle Ages. The Greeks and the Romans, strange as it may seem, had no universities in the sense in which the word has been used for the past seven or eight centuries. They had higher education, but the terms are not synonymous. Much of their instruction in law, rhetoric, and philosophy it would be hard to surpass, but it was not organized into the form of permanent institutions of learning. A great teacher like Socrates gave no diplomas; if a modern student sat at his feet for three months, he would demand a certificate, something tangible and external to show for it-an excellent theme, by the way, for a Socratic dialogue. Only in the twelfth and thirteenth centuries do there emerge in the world those features of organized education with which we are most familiar, all that machinery of instruction represented by faculties and colleges and courses of study, examinations and commencements and academic degrees. In all these matters we are the heirs and successors, not of Athens and Alexandria, but of Paris and Bologna.

Output-Only in the twelfth and thirteenth centuries do there emerge in the world those features of organized education with which we are most familiar, all that machinery of instruction represented by faculties and colleges and courses of study, examinations and commencements and academic degrees. A great teacher like Socrates gave no diplomas; if a modern student sat at his feet for three months, he would demand a certificate, something tangible and external to show for it-an excellent theme, by the way, for a Socratic dialogue.

Test Case 7 –

Input- In this essay, “normal science” means research firmly based upon one or more past scientific achievements, achievements that some particular scientific community acknowledges for a time as supplying the foundation for its further practice. Today such achievements are recounted, though seldom in their original form, by science textbooks, elementary and advanced. These textbooks expound the body of accepted theory, illustrate many or all of its successful applications, and compare these applications with exemplary observations and experiments. Before such books became popular early in the nineteenth century (and until even more recently in the newly matured sciences), many of the famous classics of science fulfilled a similar function. Aristotle's Physico, Ptolemy's Almagest, Newton's Principia and Opticks, Franklin's Electricity, Lavoisier's Chemistry, and Lyell's Geology—these and many other works served for a time implicitly to define the legitimate problems and methods of a research field for succeeding generations of practitioners. They were able to do so because they shared two essential characteristics. Their achievement was sufficiently unprecedented to attract an enduring group of adherents away from competing modes of scientific activity. Simultaneously, it was sufficiently open-ended to leave all sorts of problems for the redefined group of practitioners to resolve.

Output-Aristotle's Physico, Ptolemy's Almagest, Newton's Principia and Opticks, Franklin's Electricity, Lavoisier's Chemistry, and Lyell's Geology—these and many other works served for a time implicitly to define the legitimate problems and methods of a research field for succeeding generations of practitioners.In this essay, “normal science” means research firmly based upon one or more past scientific achievements, achievements that some particular scientific community acknowledges for a time as supplying the foundation for its further practice.

Test Case 8 –

Input- Numerous observers have described women's speech as being different from that of men. (We should observe immediately the bias inherent in that observation, since it uses men's speech as the norm against which women's speech is judged. We could just as well ask how men's speech differs from that of women, but investigators have not usually gone about the task of looking at differences in that way.) In the linguistic literature perhaps the most famous example of linguistic differentiation between the sexes is said to occur in the Lesser Antilles of the West Indies among the Carib Indians. Male and female Caribs are said to speak different languages, the result of a long-ago conquest in which a group of invading Carib-speaking men killed the local Arawak-speaking men and mated with the Arawak women. The descendants of these Carib-speaking men and Arawak-speaking women are sometimes described as having different languages for men and women. There are, for example, differences in the genders ascribed to abstract nouns, with these treated grammatically as feminine by males and masculine by females. The differences actually do not result in two “separate” or “different” languages, but rather one language with noticeable sex-based characteristics (Taylor, 1951b). Funher investigations of other languages show us that these kinds of differences between men and women are really quite widespread. The interesting question is why this should be so.

Output-We could just as well ask how men's speech differs from that of women, but investigators have not usually gone about the task of looking at differences in that way.) In the linguistic literature perhaps the most famous example of linguistic differentiation between the sexes is said to occur in the Lesser Antilles of the West Indies among the Carib Indians. Male and female Caribs are said to speak different languages, the result of a long-ago conquest in which a group of invading Carib-speaking men killed the local Arawak-speaking men and mated with the Arawak women. The descendants of these Carib-speaking men and Arawak-speaking women are sometimes described as having different languages for men and women. Funher investigations of other languages show us that these kinds of differences between men and women are really quite widespread.

10.2 Types of Testing

10.2.1 Unit Testing

Unit Testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use. Unit tests are short code fragments. Each test case is independent from the others. Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended. It is also known as component testing. We found how similar the transcribed output of the summary and the expected output with the help of the following steps:

1. Tokenization of sentences.

2. Tokenization of words.

3. Create a bag of words - Corpus (a Bag of Words) is basically an object that contains the word id and its frequency in each document, where frequency just lists the number of times each word occurs in the sentence.

4. TFIDF - Term Frequency – Inverse Document Frequency (TF-IDF) is also a bag-ofwords model but unlike the regular corpus, TFIDF down weights tokens (words) that appears frequently across documents.

5. Generate term-document matrix (TD matrix) of the data.

6. Generate a graph for the document to apply PageRank algorithm .

7. Finding the rank of every sentence using PageRank.

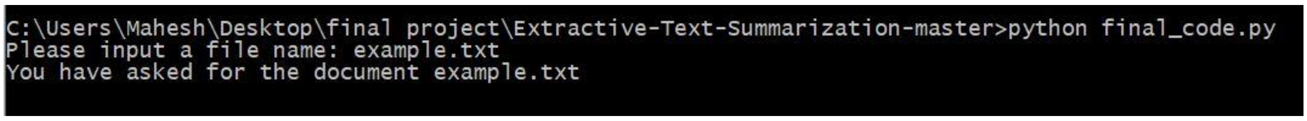
8. Arranging the sentences according to their ranks in descending order.

9. Comparing the meaning of the Input document and the Output document.

11. Result

The result of the extractive summarization is evaluated by comparing the extracted sentences with the sentences in the paragraph with the help of PageRank Algorithm.

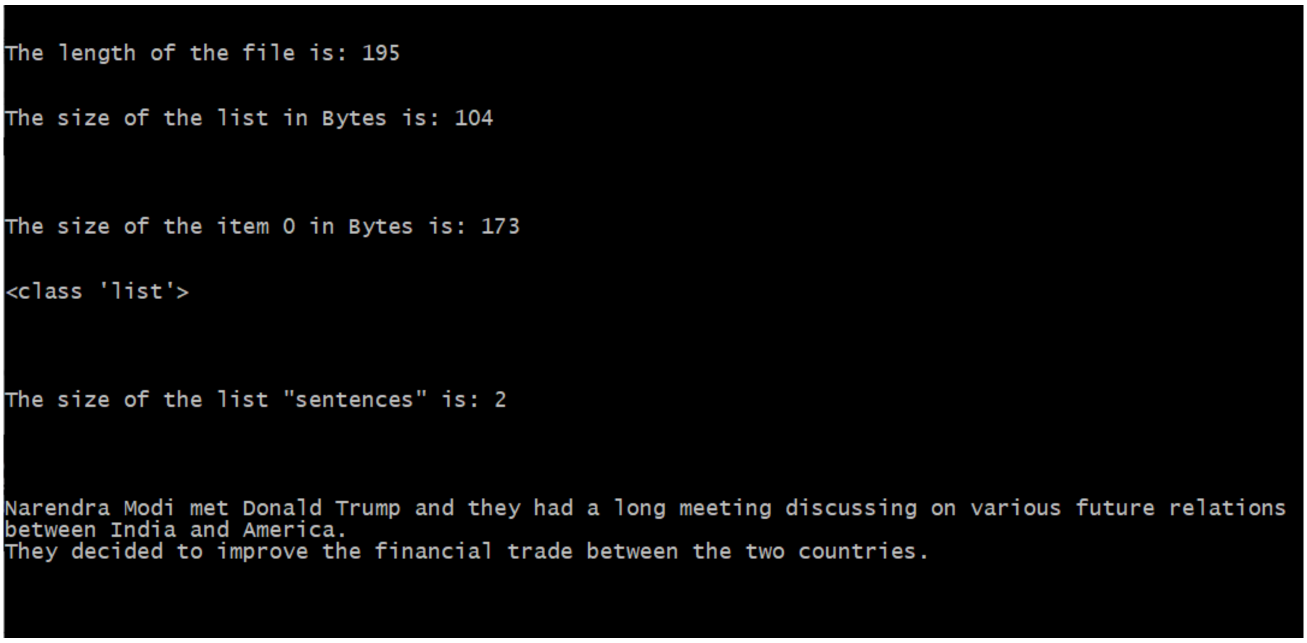
As soon as you start compiling your project it first ask the document with itz format where the document is in .txt, .pdf or in url form (eg. Example.txt) the below image show you the example.



```
C:\Users\Mahesh\Desktop\final project\Extractive-Text-Summarization-master>python final_code.py
Please input a file name: example.txt
You have asked for the document example.txt
```

Fig.11.1

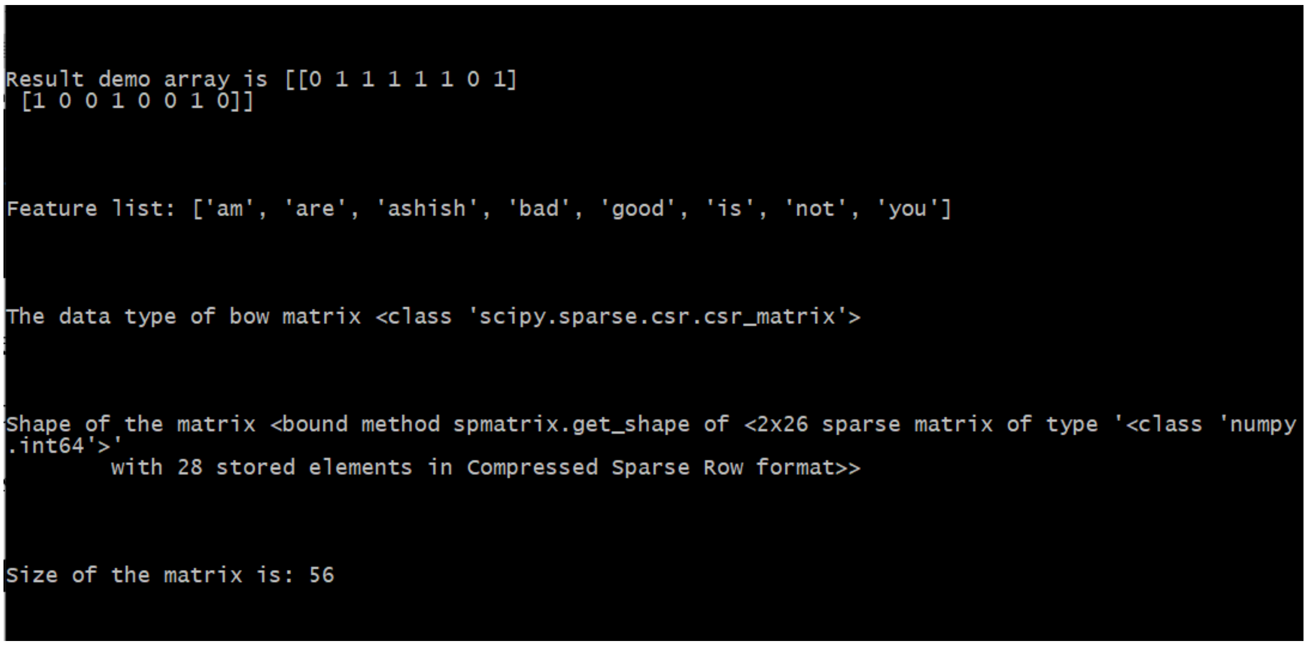
In next step program will read the content of the document. Once the reading part is done it will check the length of the file, size of the list etc..



```

The length of the file is: 195

The size of the list in Bytes is: 104


The size of the item 0 in Bytes is: 173

<class 'list'>


The size of the list "sentences" is: 2


Narendra Modi met Donald Trump and they had a long meeting discussing on various future relations
between India and America.
They decided to improve the financial trade between the two countries.
```

Fig 11.2

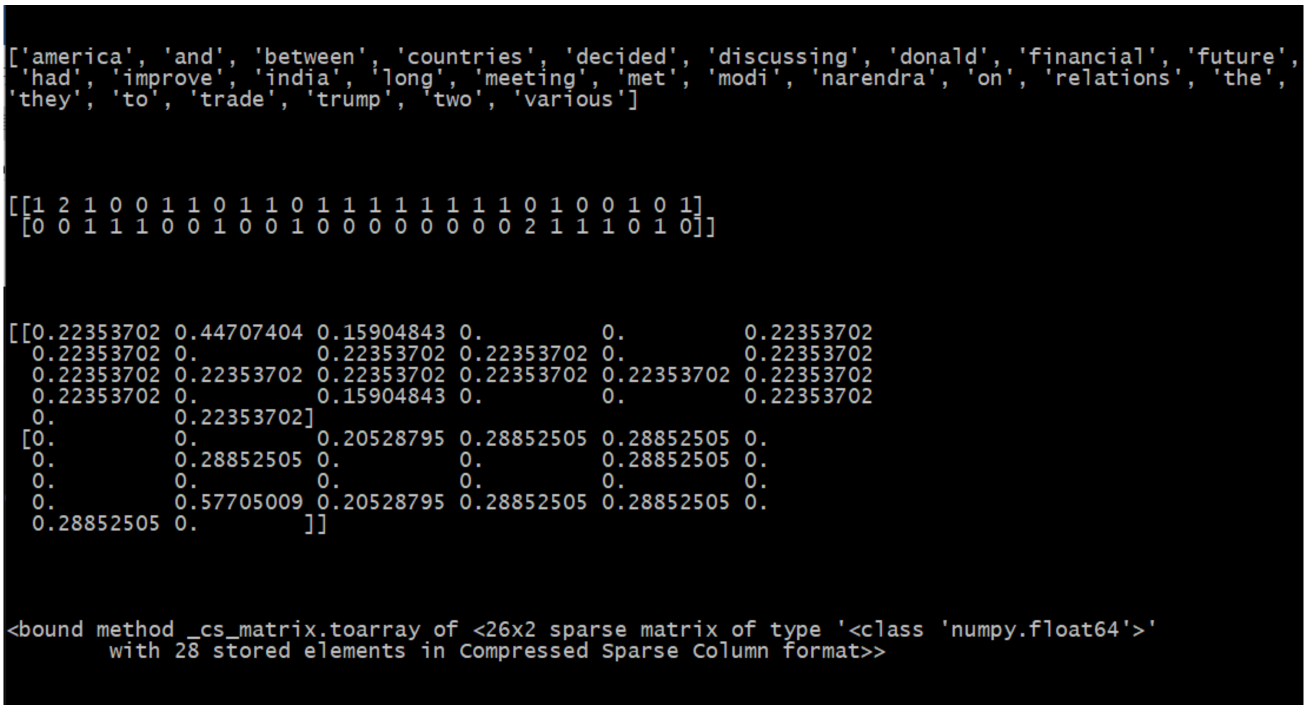
Further it tokenize the sentences n then words. The words which are tokenize are placed in array in the form of matrix. Once all the words are store in the matrix then size of the matrix are been checked.



```
Result demo array is [[0 1 1 1 1 1 0 1]
 [1 0 0 1 0 0 1 0]]


Feature list: ['am', 'are', 'ashish', 'bad', 'good', 'is', 'not', 'you']


The data type of bow matrix <class 'scipy.sparse.csr.csr_matrix'>


Shape of the matrix <bound method spmatrix.get_shape of <2x26 sparse matrix of type '<class 'numpy
.int64'>'
        with 28 stored elements in Compressed Sparse Row format>>


Size of the matrix is: 56
```

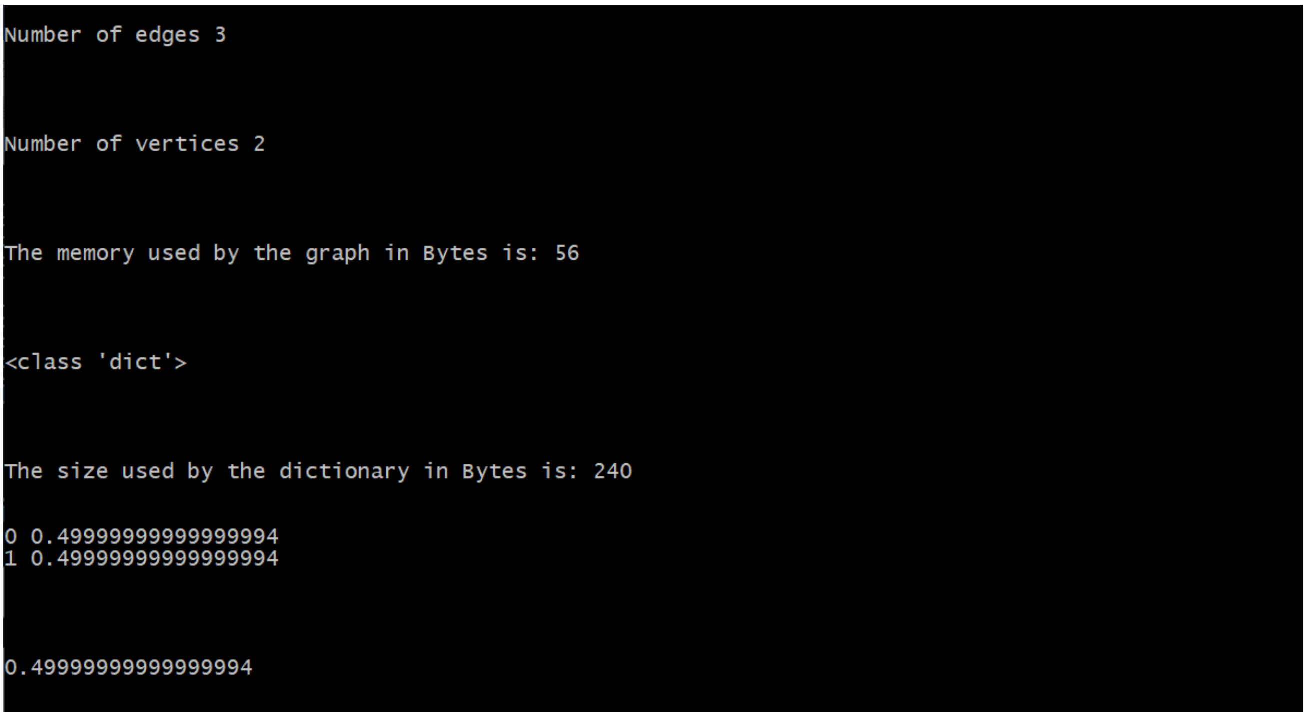
Fig 11.3

In the given below image tokenization, matrix and ranking algorithm is been done and shown.



```
['america', 'and', 'between', 'countries', 'decided', 'discussing', 'donald', 'financial', 'future',
 'had', 'improve', 'india', 'long', 'meeting', 'met', 'modi', 'narendra', 'on', 'relations', 'the',
'they', 'to', 'trade', 'trump', 'two', 'various']


[[1 2 1 0 0 1 1 0 1 1 0 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1]
 [0 0 1 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 2 1 1 1 0 1 0]]


[[0.22353702 0.44707404 0.15904843 0.         0.         0.22353702
  0.22353702 0.         0.22353702 0.22353702 0.         0.22353702
  0.22353702 0.22353702 0.22353702 0.22353702 0.22353702 0.22353702
  0.22353702 0.         0.15904843 0.         0.         0.22353702
  0.         0.22353702]
 [0.         0.         0.20528795 0.28852505 0.28852505 0.
  0.         0.28852505 0.         0.         0.28852505 0.
  0.         0.         0.         0.         0.         0.
  0.         0.57705009 0.20528795 0.28852505 0.28852505 0.
  0.28852505 0.        ]]


<bound method _cs_matrix.toarray of <26x2 sparse matrix of type '<class 'numpy.float64'>'
        with 28 stored elements in Compressed Sparse Column format>>
```

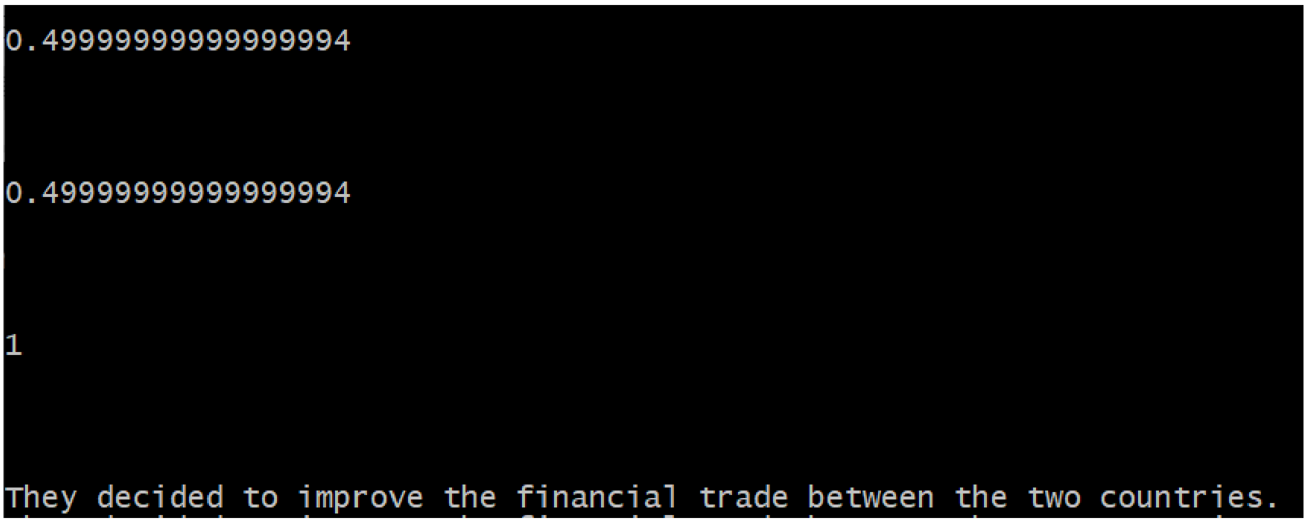
Fig 11.4



```
Number of edges 3


Number of vertices 2


The memory used by the graph in Bytes is: 56


<class 'dict'>


The size used by the dictionary in Bytes is: 240

0 0.49999999999999994
1 0.49999999999999994


0.49999999999999994
```

Fig 11.5

Output of the summary is given below



```
0.49999999999999994


0.49999999999999994


1


They decided to improve the financial trade between the two countries.
```

Fig 11.6

12. Conclusion And Future Scope

With the ever-growing text data, extractive summarization seems to have the potential for reducing the reading time by showing summaries of the text documents that capture the key points in the original documents. This tool performs extractive summarization on the input articles using TF-IDF and PageRank. The extractive summarization tool allows extraction of any number of key sentences from the original articles. The performance of the generated summaries by applying PageRank algorithm with multiple number of iterations. In the process of building the graph as initial rank, and cosine similarity was used to weight the edges between sentences. In summary extraction to prevent redundancy, if the overlapping between the selected sentence and any other sentence in the summary is very high then, this sentence is neglected. A PageRank algorithm was used to extract the summary, this algorithm was used by making the initial rank of the sentence and then sentence are arranged according to their ranks in descending order. After the arrangement finding the important sentences for the summary, then the final summary are written on the new file.

The future scope of our project can be enhanced by expanding the purpose method for multi document summarization. Documents in different languages can be summarized. Various other features can be used and the method can be combined with other methods for improving the nature of the summary. Also it can be used in Abstractive text summarization.

13. References

[1] Nikhil S. Shirwandkar, Dr.Samidha Kulkarni. Extractive Text Summarization Using Deep Learning, 2018.

[2] Shashi Narayan, Shay B.Cohen, Mirella Lapata. Ranking Sentences for Extractive Summarization with Reinforcement Learning, Apr 2018.

[3] H. P. Luhn, "The automatic creation of literature abstracts", IBM Journal of Research Development, vol. 2, no. 2, pp. 159-165, 1958.

[4] H. P. Edmundson, "New methods in automatic extracting", Journal of the ACM, vol. 16, no. 2, pp. 264-285, 1969.

[5] K. R. McKeown, D. R. Radev, "Generating summaries of multiple news articles", Proceedings of SIGIR '95, pp. 74-82, 1995.

[6] Shashi Narayan, Nikos Papasarantopoulos, Shay B. Cohen, & Mirella Lapata. Neural extractive summarization with side information. CoRR abs/1704.04530.

[7] Carbonell, J., Goldstein, J.: The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries. In: Research and Development in Information Retrieval, pp. 335–336 (1998)