

A Project Report

AUTOMATIC TEXT SUMMARIZATION

Submitted in partial fulfillment of the requirements for the award of degree

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

by

MOHAMMED SAFI AMMAR (160117733032)



**Department of Computer Science and Engineering,
Chaitanya Bharathi Institute of Technology (A),**

**(Affiliated to Osmania University,
Hyderabad) Hyderabad,
TELANGANA (INDIA) –500 075
[2020-2021]**



**CHAITANYA BHARATHI
INSTITUTE OF TECHNOLOGY (A)**

Kokapet (Village), Gandipet, Hyderabad, Telangana-500075. www.cbti.ac.in



ISO Certified
9001:2015



CERTIFICATE

This is to certify that the project titled **“Automatic Text Summarization”** is the bonafide work carried out by **Mohammed Safi Ammar (160117733032)**, a student of B.E. (CSE) of Chaitanya Bharathi Institute of Technology(A), Hyderabad, affiliated to Osmania University, Hyderabad, Telangana(India) during the academic year 2020-2021, submitted in partial fulfillment of the requirements for the award of the degree in **Bachelor of Engineering (Computer Science and Engineering)** and that the project has not formed the basis for the award previously of any other degree, diploma, fellowship or any other similar title.

Supervisor,
Dr. Kolla Morarjee
(Assoc. Professor)

Head, CSE Department,
Dr. Y. Rama Devi
(Professor)

Place: Hyderabad

Date: 23rd May 2021

DECLARATION

I hereby declare that the project entitled “**Automatic Text Summarization**” submitted for the B.E (CSE) degree is our original work, and the project has not formed the basis for the award of any other degree, diploma, fellowship, or any other similar titles.

Mohammed Safi Ammar
(160117733032)

Place: Hyderabad

Date: 23rd May 2021

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success. They have been a guiding light and source of inspiration towards the completion of the project. I would like to express our sincere gratitude and indebtedness to our project guide, **Dr. Kolla Morarjee**, Assoc. Professor who has supported us throughout our project with patience and knowledge. I am also thankful to Head of the department **Dr. Y. Rama Devi** for providing excellent infrastructure and a conducive atmosphere for completing this project successfully. I am also extremely thankful to our Project coordinators **Dr. K. Sagar**, Professor and **Smt. G. Vanitha**, Asst. Professor, and Dept. of CSE, for their valuable suggestions and interest throughout the course of this project. I convey our heartfelt thanks to the lab staff for allowing us to use the required equipment whenever needed. I sincerely acknowledge and thank all those who gave directly or indirectly their support in completion of this work.

Mohammed Safi Ammar
(160117733032)

ABSTRACT

Going through a huge paragraph or an article in times of need can be quite overwhelming, and hence a short summary of the same can come in handy. Many attempts have been made to automate text summarization to save time. The automatic text summarization work has become more and more important because the amount of data on the Internet is increasing fast, and automatic text summarization work can extract useful information and knowledge what user's need that could be easily handled by humans and used for many purposes.

Text summarization methods are classified into two categories: Extractive and abstractive. The extractive method consists of extracting important sentences or paragraph from some source of text and rejoining them to get the summarized form of the source content. The criteria for evaluating an importance of a sentence or paragraph are based on the statistical features and parameters of the sentences. To summarize text from various sources an application is proposed that can summarize news articles, Wikipedia pages, and youtube videos using extractive text summarization methods.

TABLE OF CONTENTS

	TITLE OF CHAPTER	PAGE NO.
	Certificate of the Guide	I
	Declaration of the Students	II
	Acknowledgement	III
	Abstract	IV
	List of Figures	VII
	List of Tables	VIII
1.	INTRODUCTION	1
	1.1. Problem Definition	1
	1.2. Methodologies	1
	1.3. Outline of the Results	2
	1.4. Scope of the Project	2
	1.5. Organization of the Report	2
2.	LITERATURE SURVEY	4
	2.1 Introduction to the problem domain terminology	4
	2.2 Existing Systems	5
	2.3 Related Works	5
	2.4 Tools and Technologies used	7
3.	DESIGN OF THE PROPOSED SYSTEM	10
	3.1 Block Diagram	10
	3.2 Module Description	11
	3.3 Theoretical Algorithms	12
	3.3.1 PageRank Algorithm	12
	3.3.2 Text Rank Algorithm (using Gensim)	13
	3.3.3 Cosine Similarity	15
	3.3.4 Custom Frequency Distribution Algorithm	16

4.	IMPLEMENTATION OF THE PROPOSED SYSTEM	17
	4.1 Flowchart	17
	4.2 UML Diagrams	18
	4.2.1 Activity Diagram	18
	4.2.2 Use Case Diagram	19
	4.3 Steps of System Design	19
	4.4 Algorithms and Pseudo Code	20
	4.4.1 Libraries Used	20
	4.4.2 Gensim (Text Rank)	21
	4.4.3 Custom Frequency Distribution Algorithm	23
	4.5 Data Description	24
5.	RESULTS AND DISCUSSIONS	26
	5.1 News Article Module	27
	5.2 Wikipedia Pages Module	30
	5.3 YouTube Video Module	31
	5.4 Custom Text Input Module	33
6.	CONCLUSION AND FUTURE SCOPE	35
	6.1 Conclusion	35
	6.1.1 Applications	35
	6.1.2 Limitations	37
	6.2 Future Scope	38
	REFERENCES	40
	APPENDICES	41

LIST OF FIGURES

FIGURE NO.	FIGURE NAME	PAGE NO.
3.1.1	Block Diagram of the proposed system	10
3.3.1.1	Mathematical formula for PageRank	12
3.3.1.2	Example for working of PageRank Algorithm	13
3.3.2.1	Architecture of TextRank Algorithm	14
3.3.3.1	Cosine Similarity Formula	15
4.1.1	Flowchart of the system	17
4.2.1.1	Activity Diagram	18
4.2.2.1	Use Case Diagram	19
4.4.1.1	Libraries Used	21
4.4.2.1	Text Rank Structure	22
4.4.2.2	Text Rank Pseudo Code for Wikipedia pages	23
4.4.3.1	Pseudo Code for News Article Summarization	24
4.5.1	Variables containing URL/videoID	25
5.1	Running the streamlit application	26
5.2	User interface of the application	26
5.1.1	News article URL input	27
5.1.2	Original text of news article	27
5.1.3	Summarized news article with sentence scores	28
5.2.1	Wikipedia Page URL input	30
5.2.2	Generated Wikipedia page summary	31
5.3.1	Youtube Video ID input	31
5.3.2	Youtube Video summary	32
5.4.1	Input of custom article	33
5.4.2	Summary audio playback	33
5.4.3	Custom text input summary	34

LIST OF TABLES

TABLE NO.	TABLE NAME	PAGE NO.
5.1.1	Sentence scores of the news article	29

1. INTRODUCTION

Text summarization methods are classified into two categories: Extractive and abstractive. The extractive method consists of extracting important sentences or paragraph from some source of text and rejoining them to get the summarized form of the source content. The criteria for evaluating an importance of a sentence or paragraph are based on the statistical features and parameters of the sentences.

1.1 Problem Definition

In dire times, going through a huge paragraph or an article can be overwhelming, and hence a short summary of the same can come in handy. The automatic text summarization work has become more and more important because the amount of data on the Internet is increasing so fast, and automatic text summarization work can extract useful information and knowledge. It is very difficult for a person to describe and ingest the whole content. The manual conversion or summarization is very difficult task and hence automation is need. The automation can be achieve using artificial intelligence techniques. Watching a prominent person giving a speech or a video podcast can also get boring very quickly and it would be faster and easier to just read a summary of what has been spoken about. To summarize text from various sources we propose an application that can summarize news articles, Wikipedia pages, and youtube videos.

1.2 Methodologies

Text summarization methods are classified into two categories:

- **Extractive:** Selects important sentences from text based on level of importance. It is suitable for large text data.
- **Abstractive:** Selects important keywords from text and recreates sentences by adding words. This is more suitable for shorter text data.

This paper uses text rank extractive summarization method which extracts important sentences from a source of text and rejoins them to get summarized content.

Automatic text summarization process may be done in three steps:

- **Identification:** This is the first step of the process. In this step, the most important and relevant topics are identified.
- **Interpretation:** In this step, sentences that are identified in the previous step are modified and unwanted characters and stop words are removed without sacrificing the meaning of the sentence.
- **Generation:** In this step all the interpreted sentences are combined to form a summary which is easily understandable.

1.3 Outline of the Results

The application is built on the streamlit framework which is an open-source python library that makes it easy to create and share beautiful, custom web apps for machine learning and data science. It combines an implementation of custom algorithm and the textrank algorithm. It provides various ways to source text from. It also includes summarizing YouTube videos as a module. The sentences selected by the summarizer are evaluated based on important keyword frequency distribution and the sentence score is displayed for news article summaries.

1.4 Scope of the Project

Without summaries it would be practically impossible for human beings to get access to the ever-growing mass of information available online. Although research in text summarization is over 50 years old, some efforts are still needed given the insufficient quality of automatic summaries and absence of proper user interface application.

Today there are numerous documents, papers, reports, and articles available in digital form, but most of them lack summaries. The information in them is often too abundant for it to be possible to manually search, sift and choose which knowledge one should acquire. This information must instead be automatically filtered and extracted to avoid drowning in it. Automatic Text Summarization is a technique where a computer summarizes a text. A text is given to the computer and the computer returns a shorter less redundant extract of the original text. Automatic text summarization is untiring, consistent, and always available.

1.5 Organization of the Report

This introduction section is followed by Literature Survey. The literature survey explains the current existing systems which are commonly based on extractive text summarization. It also introduces domain specific terminology which form the background to understand this project. It discusses in depth about some existing solution's core aspect which also forms the basis for many other solutions. The section also discusses the drawbacks in all the solutions exhaustively. The literature survey section is followed by Design of the Proposed System section. This section discusses the evolution and design of the proposed solution. Next section discusses the implementation of the design discussed in the previous section. The Data Flow Diagrams and Flowcharts are discussed in this section of the project. The algorithm is also discussed in this section. The testing process is also included. The next section deals with the result analysis. The system is executed in real time and the results are analyzed and discussed. The final section deals with limitations and recommendations. The references showing the base papers used in this project are then mentioned.

2. LITERATURE SURVEY

2.1 Introduction to the Problem Domain terminology

Over the span of time, different directions have appeared related to automated text summarization. We can now differentiate between generic and query-based text summarization. Query-based text summarization refers to performing summarization in such a way that the summary relates to the particular query. Whereas generic text summarization tends to provide summary which does not refer to any particular query, but instead to the main topic of the original text, thus preserving its substance and leaving out irrelevant details.

Another main classification of text summarization can be done based on the way the summary is constructed: extractive or abstractive. Extractive summarization refers to the process of generating summaries by identifying relevant pieces of text (e.g., sentences, paragraphs) and literally including them in the summary. On the contrary, abstractive summarization does not rely on existing formulations but rather tends to paraphrase the original text using, for example, synonyms, different linguistic forms etc. While extractive summarization is less complex, both approaches have its own challenges. Abstractive summarization requires deep linguistic knowledge to maintain proper language constructs when reformulating text parts (which relates to emerging field of natural language generation as well).

Yet another classification of summaries exists based on the number of documents to be summarized: in case of only one, it is called single-document summarization; otherwise, we deal with multiple-document summarization. In the cases when the original text is written in different languages and the aim is to make a summary in one language, we are talking about multilingual summarization.

Depending on the type of learning, text summarization can be divided to supervised and unsupervised. The advantage of unsupervised approaches is that they do not require any external source for learning. On the other side, supervised approaches require providing reliable human-made summaries which is a time-consuming and effort-demanding task.

2.2 Existing Systems

Some of the key challenges in text summarization include topic identification, interpretation, summary generation, and evaluation of the generated summary. Most practical text summarization systems are based on some form of extractive summarization.

All extraction-based summarizers, irrespective of the differences in approaches, perform the following three relatively independent tasks: capturing key aspects of text and storing it using an intermediate representation, scoring sentences in the text based on that representation, and composing a summary by selecting several sentences.

How the keywords are extracted and how sentences are ranked and selected depends on the statistical approach used by the user. Though these extractive methods present promising results, they are not quite feasible in real world applications and have limited areas of interest. Hence a custom extractive approach is deployed based on the textrank algorithm which is used to summarize text and speech from various sources.

2.3 Related Works

Text Summarization: An Extractive Approach. The authors used textrank algorithm to extract and score sentences and generate summaries. The validation of the model was performed using the bench-marked source text. From the obtained result, it was evident that the summarization model performed well and summarized with precision and cogency. In this paper, a text summarization model using text rank algorithm is proposed and implemented. It is a general-purpose graph-based approach. The model is very helpful for summarizing the large amount of data [1].

Extractive Text Summarization Using Sentence Ranking. J.N. Madhur and R. Ganesh Kumar used a novel statistical method to perform extractive text summarization on a single document. Sentences are ranked by assigning weights and they are ranked based on their weights [2]. In this paper, extractive based text summarization is proposed by using statistical novel approach based on the sentence ranking the sentences are

selected by the summarizer. The sentences which are extracted are produced as a summarized text and it is converted into audio form. The proposed model improves the accuracy when compared traditional approach.

Automatic text summarization approaches Ahmad T. Al-Taani cited the different approaches in the field of text summarization in natural language processing [3]. He classified text summarization into single-document and multi-document summarization methods.

Automatic text summarization of news articles. Prakhar Sethi, Sameer Sonawane, Saumitra Khanwalker, and R. B. Keskar propose a technique of text summarization which focuses on the problem of identifying the most important portions of the text and producing coherent summaries [4]. Without requiring full semantic interpretation of the text, they created a summary using a model of topic progression in the text derived from lexical chains.

Automatic text summarization based on semantic analysis approach for documents in Indonesian language [5]. The authors goal was to define a measurement for text summarization using Semantic Analysis Approach for Documents in Indonesian language. The main idea of semantic analysis is to obtain the similarity between sentences by calculating the vector values of each sentence with the title.

Automatic text summarization based on sentences clustering and extraction. Pei-ying Zhang and Cun-he Li used semantic distance to cluster sentences and calculated the accumulative sentence similarity based on multi-features combination method, and chose sentences by some extraction rules [6].

2.5 Tools and Technologies used

1. Streamlit

Streamlit is an open-source python library for building fast, performant and beautiful data apps. Streamlit lets you turn data scripts into sharable web app very quickly. It is a great tool to develop interactive apps quickly for demonstrating a solution.

Under the hood, it uses React as a frontend framework to render the data on the screen. So, React Developers can easily manipulate the UI with few changes in the code. The biggest advantage of streamlit is that you can build an app just by knowing how to write a python script without needing to know front end development. As the library is still in early stages there is no default feature to add multiple pages in a single app, but we can create this by using some other features.

2. Beautiful Soup

Beautiful Soup is a Python library for pulling data out of HTML and XML files and other markup languages. It works with any parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. Say you have found some webpages that display data relevant to your research, such as date or address information, but that do not provide any way of downloading the data directly. Beautiful Soup helps you pull particular content from a webpage, remove the HTML markup, and save the information. It is a tool for web scraping that helps you clean up and parse the documents you have pulled down from the web.

Three features make it powerful:

- Beautiful Soup provides a few simple methods and Pythonic idioms for navigating, searching, and modifying a parse tree.
- Beautiful Soup automatically converts incoming documents to Unicode and outgoing documents to UTF-8.
- Beautiful Soup sits on top of popular Python parsers like lxml and html5lib, which allows us to try out different parsing strategies or trade speed for flexibility.

3. Urllib

Urllib module is the URL handling module for python. It is used to fetch URLs (Uniform Resource Locators). It uses the urlopen function and is able to fetch URLs using a variety of different protocols.

Urllib is a package that collects several modules for working with URLs, such as:

- urllib.request for opening and reading.
- urllib.parse for parsing URLs
- urllib.error for the exceptions raised
- urllib.robotparser for parsing robot.txt files

4. Youtube transcript API

Youtube transcript API is an open-source software project. It is a python API which allows you to get the transcript/subtitles for a given YouTube video. It works for automatically generated subtitles, supports translating subtitles without requiring the need for a headless browser.

5. Gensim

“Generate Similar” is a popular open-source natural language processing (NLP) library used for unsupervised topic modeling. It uses top academic models and modern statistical machine learning to perform various complex tasks such as –

- Building document or word vectors
- Corpora
- Performing topic identification
- Performing document comparison (retrieving semantically similar documents)
- Analysing plain-text documents for semantic structure

Apart from performing the above complex tasks, Gensim, implemented in Python and Cython, is designed to handle large text collections using data streaming as well as

incremental online algorithms. This makes it different from those machine learning software packages that target only in-memory processing.

6. Nltk

The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for English written in the Python programming language. It is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

7. Heapq (nlargest)

Heap data structure is mainly used to represent a priority queue. In Python, it is available using “heapq” module. The property of this data structure in Python is that each time the smallest of heap element is popped (min heap). Whenever elements are pushed or popped, heap structure is maintained.

- **Nlargest (k, iterable, key):** This function is used to return the k largest elements from the iterable specified and satisfying the key if mentioned.

8. Pyttsx3

pyttsx3 is a text-to-speech conversion library in Python. Unlike alternative libraries, it works offline and is compatible with both Python 2 and 3. An application invokes the `pyttsx3.init()` factory function to get a reference to a `pyttsx3`. Engine instance. it is a very easy to use tool which converts the entered text into speech.

The `pyttsx3` module supports two voices first is female and the second is male which is provided by “sapi5” for windows.

3. DESIGN OF THE PROPOSED SYSTEM

3.1 Block Diagram

A block diagram is a diagram of a system in which the principal parts or functions are represented by blocks connected by lines that show the relationships of the blocks. They are heavily used in engineering in hardware design, electronic design, software design, and process flow diagrams.

Block diagrams are typically used for higher level, less detailed descriptions that are intended to clarify overall concepts without concern for the details of implementation. Contrast this with the schematic diagrams and layout diagrams used in electrical engineering, which show the implementation details of electrical components and physical construction.

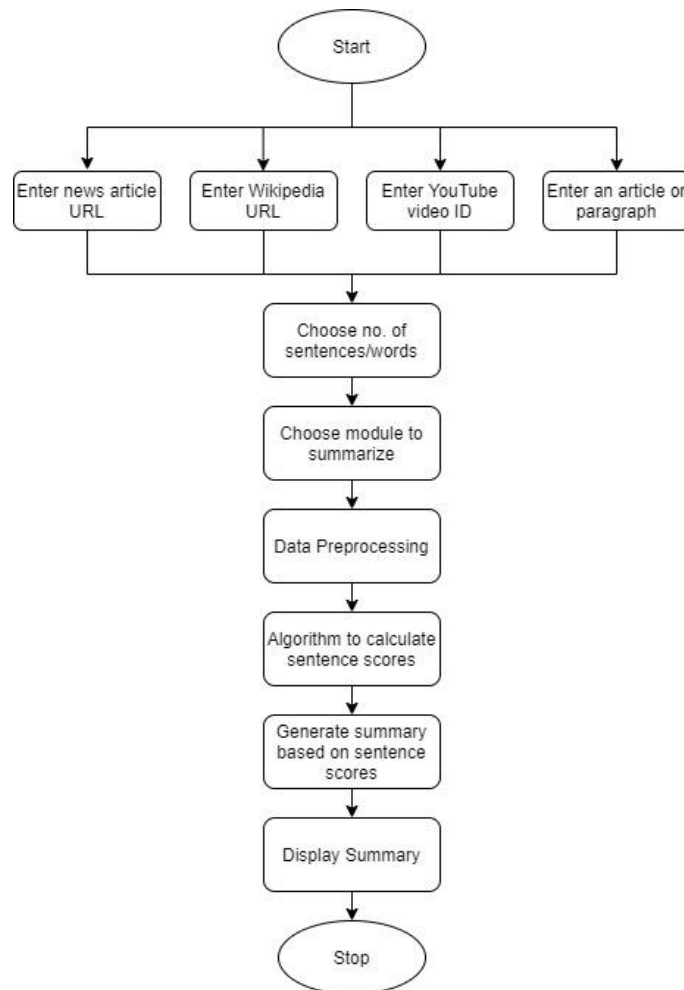


Fig 3.1.1: Block Diagram of the proposed system.

The above block diagram gives a brief flow of the working of the deployed system (Fig 3.1.1). The steps include entering data, choosing number of sentences or words, choosing module to summarize, data preprocessing, generating, and displaying summary.

3.2 Module Description

1. News Article Summarization

This module handles the summarization of news article. A news article URL is first stored in a variable from the user input. The article is extracted from the news article site using a newspaper library in Python. The data is cleaned by removing any unwanted symbols, spaces, and breaks. The text is tokenized into words and sentences. Using a custom frequency distribution algorithm, the scores of each word are calculated and added to get a sentence score. Using the sentence scores a summary of the news is obtained.

2. Wikipedia Page Summarization

This module handles the summarization of Wikipedia pages. A Wikipedia page URL is first stored in a variable from the user input. The text material is extracted from the Wikipedia page using urllib library in Python. The data is cleaned by removing any unwanted symbols, spaces, and breaks. The text is tokenized into words and sentences and stop words are removed from the word tokens. Using gensim library based on the textrank algorithm, the mz_keywords function calculates scores of each word from the tokens and the scores are added to get a sentence score. Using the sentence scores a summary of the Wikipedia page is obtained.

3. YouTube Video Summarization

This module handles the summarization of YouTube videos. A YouTube video ID is first stored in a variable from the user input. The title and text material are extracted from the video ID using re and youtube_transcript_api library respectively in Python. The data is cleaned by removing any unwanted symbols, spaces, and breaks. The text is tokenized into words and sentences and stop words are removed from the word tokens

using the nltk library. Using the custom frequency algorithm, the scores of each word from the tokens are calculated and the scores are added to get a sentence score. Using the sentence scores the sentences are ranked and a summary of the YouTube video is generated based on those ranks.

4. Custom input Text Summarization

This module handles the summarization of custom text input from the user. The text is first stored in a variable from the user input. The data is cleaned by removing any unwanted symbols, spaces, and breaks. The text is tokenized into words and sentences and stop words are removed from the word tokens using the nltk library. Using the custom frequency algorithm, the scores of each word from the tokens are calculated and the scores are added to get a sentence score. Using the sentence scores the sentences are ranked and a summary of the text is generated based on those ranks. This module also presents an added functionality of text to speech of the summary. The summary is spoken to the user using the pyttsx3 text to speech API in python.

3.3 Theoretical Algorithms

3.3.1 Page Rank Algorithm

To implement summarization using text rank algorithm one must understand the page rank algorithm which is the parent algorithm for text rank algorithm.

Page rank algorithm was created to measure the importance of the web pages by ranking them based on the pagerank score. This score indicates the likelihood of a user visiting the web page. It is calculated based on the presence of links between any two web pages. It was first used to rank web pages in the Google search engine. Nowadays, it is used in many different fields, for example in ranking users in social media etc.

$$PageRank\ of\ site = \sum \frac{PageRank\ of\ inbound\ link}{Number\ of\ links\ on\ that\ page}$$

OR

$$PR(u) = (1 - d) + d \times \sum \frac{PR(v)}{N(v)}$$

Fig 3.3.1.1: Mathematical formula for PageRank.

The PageRank theory holds that an imaginary surfer who is randomly clicking on links will eventually stop clicking. The probability, at any step, that the person will continue is a damping factor (d). Various studies have tested different damping factors, but it is generally assumed that the damping factor will be set around 0.85 (Fig 3.3.1.2).



Fig 3.3.1.2: Example for working of PageRank Algorithm.

Few other uses of PageRank Algorithm:

- Finding how well connected a person is on Social Media: One of the unexplored territories in social media analytics is the network information. Using this network information, we can estimate how influential is the user. And therefore, prioritize our efforts to please the most influential customers. Networks can be easily analyzed using Page Rank algorithm.
- Fraud Detection in Pharmaceutical industry: Many countries including US struggle with the problem of high percentage medical frauds. Such frauds can be spotted using Page Rank algorithm.
- Understand the importance of packages in any programming language: Page Rank algorithm can also be used to understand the layers of packages used in languages like R and Python. We will take up this topic in our next article.

3.3.2 Text Rank Algorithm (using Gensim)

Gensim is a free Python library designed to automatically extract semantic topics from documents. The gensim implementation is based on the popular TextRank algorithm. It is an open-source vector space modelling and topic modelling toolkit, implemented

in the Python programming language, using NumPy, SciPy and optionally Cython for performance.

The TextRank algorithm is based on the Page Rank algorithm. The only differences being that the sentences in the TextRank algorithm replace the web pages in the PageRank algorithm. TextRank is a general purpose, graph based ranking algorithm for NLP. Graph-based ranking algorithms are a way for deciding the importance of a vertex within a graph, based on global information recursively drawn from the entire graph. The TextRank keyword extraction algorithm is fully unsupervised. No training is necessary (Fig 3.3.2.1).

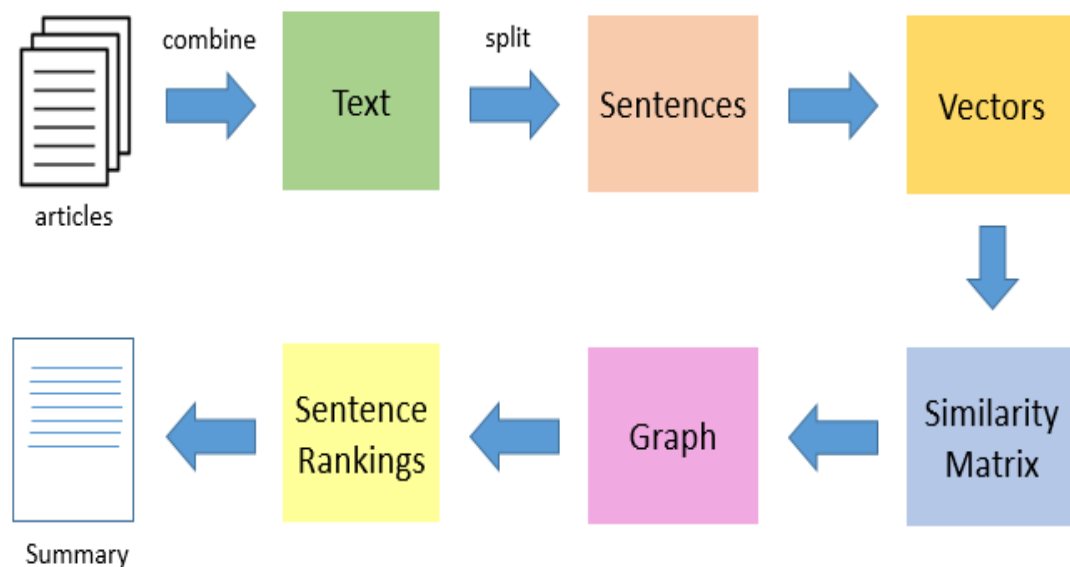


Fig 3.3.2.1: Architecture of TextRank Algorithm.

The basic idea implemented by a graph-based ranking model is that of voting or recommendation. When one vertex links to another one, it is basically casting a vote for that vertex. The higher the number of votes cast for a vertex, the higher the importance of that vertex. This gives us a ranking of sort of all the sentences in the text.

TextRank includes two NLP tasks:

- Keyword extraction task
 - The task of keyword extraction algorithm is to automatically identify in a text a set of terms that best describe the document.
 - The simplest possible approach is to use a frequency criterion.
- Sentence extraction task
 - TextRank is very well suited for applications involving entire sentences, since it allows for a ranking over text units that is recursively computed based on information drawn from the entire text.

Working of Text Rank Algorithm:

The working procedure of text rank algorithm is similar to the page rank algorithm where sentences are used in place of pages. Text rank algorithm used same formula of the page rank algorithm which is as follows:

$$T R(s) = (1 - d) + d (T R(T_1)/C(T_1) + \dots + T R(T_n)/C(T_n))$$

$T R(s)$ is the text rank of the sentence s and d is the damping factor that can be put between 0 and 1. The graph is symmetrical in case of text rank algorithm and does not require any training as it is an unsupervised method. After this algorithm, we have picked up top-ranked sentences and generated the summary from those sentences.

3.3.3 Cosine Similarity

Cosine similarity is a metric used to determine how similar the documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. In this context, the two vectors are arrays containing the word counts of two sentences (Fig 3.3.3.1).

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}},$$

Fig 3.3.3.1: Cosine Similarity Formula.

The cosine similarity is advantageous because even if the two similar sentences are far apart by the Euclidean distance because of the size (like, the word ‘cricket’ appeared 10 times in one sentence and 2 times in another) they could still have a smaller angle between them. Smaller the angle, higher the similarity.

3.3.4 Custom Frequency Distribution Algorithm

A works on the principle of important keywords identification based on their frequency of appearance in the input text. E.g.: A noun appearing the greatest number of times would be classified as an important keyword.

The news article and custom text modules use the custom algorithm to summarize text.

Working of the algorithm:

- Pre-process the given text. This includes stop words removal, punctuation removal, and stemming.
- Tokenize the text into sentences and words.
- Create a frequency distribution table of the words by using the formula:
$$\text{Word Frequency} / \text{Maximum Frequency (Freq of word appearing most in text)}$$
- Calculate sentence scores by adding frequency scores of each word present in the sentence.
- Generate a summary based on the highest sentence scores.

4. IMPLEMENTATION OF THE PROPOSED SYSTEM

4.1 Flowchart

A flowchart is a type of diagram that represents an algorithm, workflow, or process. The flowchart shows the steps as boxes of various kinds, and their order by connecting the boxes with arrows. The two most common types of boxes in a flowchart are: a processing step, usually called activity, and denoted as a rectangular box. a decision usually denoted as a diamond (Fig 4.1.1).

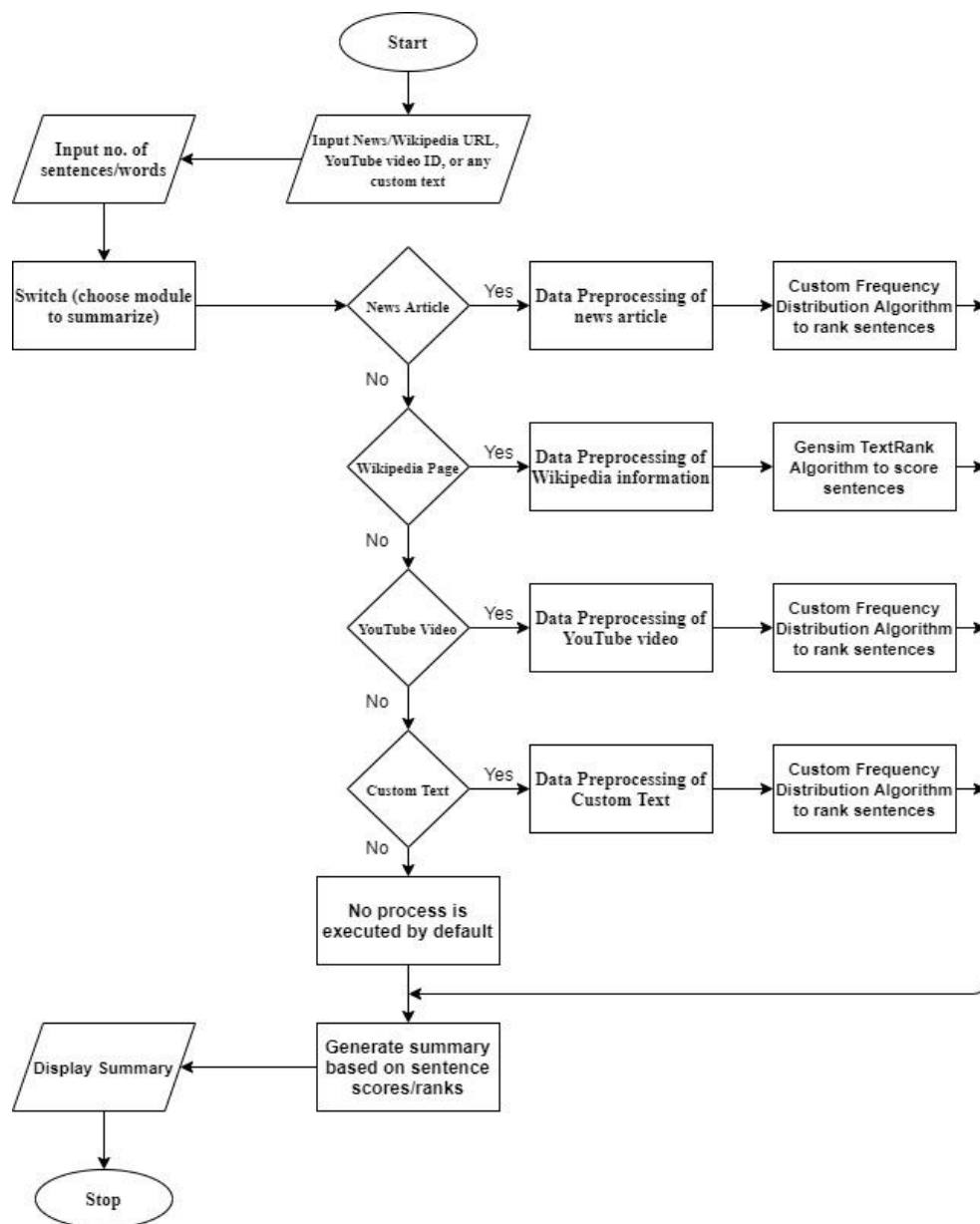


Fig 4.1.1: Flowchart of the system.

4.2 UML Diagrams

UML stands for Unified Modelling Language. UML is a standardized general-purpose modelling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Building blocks of the UML vocabulary encompasses three kinds of blocks.

1. Things
2. Relationships
3. Diagrams

4.2.1 Activity Diagram

An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed. We can depict both sequential processing and concurrent processing of activities using an activity diagram. They are used in business and process modelling where their primary use is to depict the dynamic aspects of a system (Fig 4.2.1.1).

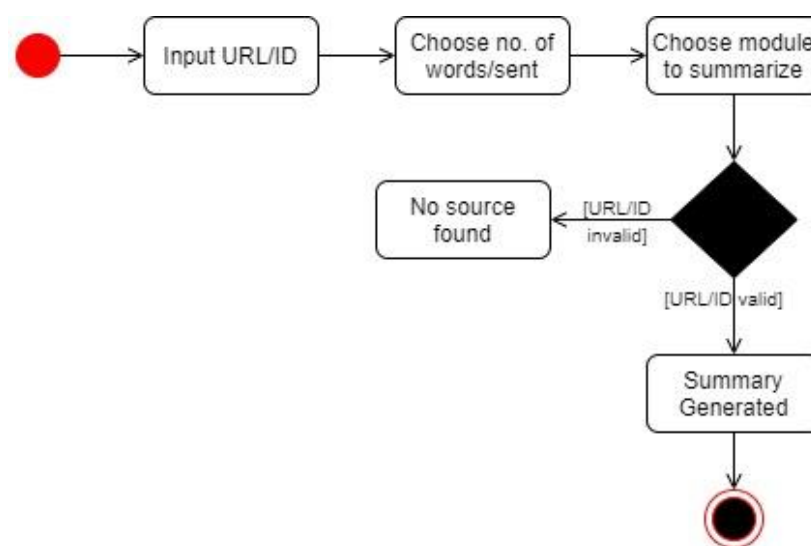


Fig 4.2.1.1: Activity Diagram.

4.2.2 Use Case Diagram

A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform (Fig 4.2.2.1).

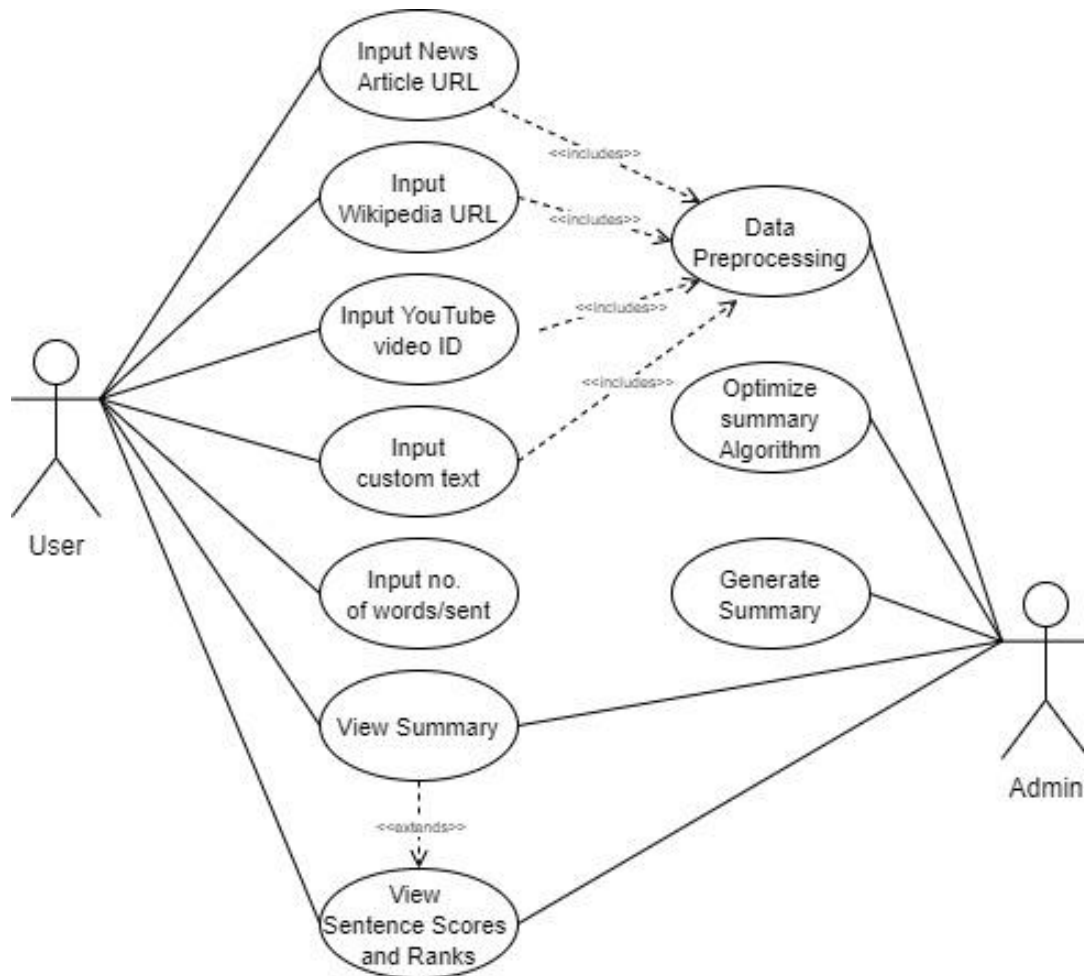


Fig 4.2.2.1: Use Case Diagram.

4.3 Steps of System Design

The project has multiple phases in its design. The first phase involves extracting data from the website URLs or YouTube videos. For extracting data from Wikipedia pages BeautifulSoup is used. It helps to pull particular content from the webpage, remove the HTML markup, and save the information. It is then used to clean up and parse the data

obtained from the web. Similarly, News articles are also extracted, and parsed. YouTube video scripts are extracted through a python API called “youtube_transcript_api”.

Before applying any summarization technique, data has to be pre-processed. The data is pre-processed with the following steps:

- **Sentence’s boundary identification:** In English, the boundary of the sentence is identified with the presence of a dot or question mark at the end of the sentence.
- **Stop-word elimination:** The words which have no contribution for selecting the important sentences such as prepositions, articles, pronouns i.e., words with no semantics are removed.
- **Cleaning:** Unwanted characters and spaces are removed using regular expressions.

Based on the module, the data is then deployed into the algorithm to generate summary with the process of tokenization, sentence ranking, and summary generation. Once the summary is generated, it is output to display in the streamlit web application.

4.4 Algorithms and Pseudo Code

4.4.1 Libraries Used

During the implementation we made use of some python packages which possess many in-built methods that provides many useful functionalities (Fig 4.4.1.1).

- **Streamlit:** It is an open-source python library for building fast, performant and beautiful data apps. Streamlit lets you turn data scripts into sharable web app very quickly. It is a great tool to develop interactive apps quickly for demonstrating a solution.
- **Beautiful Soup:** It helps you pull particular content from a webpage, remove the HTML markup, and save the information. It is a tool for web scraping that helps you clean up and parse the documents you have pulled down from the web.

- **Urllib:** Urllib module is the URL handling module for python. It is used to fetch URLs (Uniform Resource Locators). It uses the urlopen function and is able to fetch URLs using a variety of different protocols.
- **Gensim:** “Generate Similar” is a popular open-source natural language processing (NLP) library used for unsupervised topic modeling. It uses top academic models and modern statistical machine learning to perform various complex tasks.
- **Nltk:** The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for English written in the Python programming language. It is a leading platform for building Python programs to work with human language data.
- **Pyttsx3:** It is a text-to-speech conversion library in Python. Unlike alternative libraries, it works offline and is compatible with both Python 2 and 3.
- **Youtube transcript API:** It is an open-source software project. It is a python API which allows you to get the transcript/subtitles for a given YouTube video.

```
import streamlit as st
from bs4 import BeautifulSoup
import nltk
from newspaper import Article
import pyttsx3
import bs4 as bs
import urllib.request
from gensim.summarization import summarize as su_gs
from gensim.summarization import keywords
from gensim.summarization import mz_keywords
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
```

Fig 4.4.1.1: Libraries Used

4.4.2 Gensim (Text Rank)

The TextRank algorithm is based on the Page Rank algorithm. The only differences being that the sentences in the TextRank algorithm replace the web pages in the PageRank algorithm.

TextRank is a general purpose, graph based ranking algorithm for NLP. Graph-based ranking algorithms are a way for deciding the importance of a vertex within a graph, based on global information recursively drawn from the entire graph. The TextRank keyword extraction algorithm is fully unsupervised. No training is necessary (Fig 4.4.2.1).

Algorithm:

- Pre-process the given text. This includes stop words removal, punctuation removal, and stemming.
- Make a graph with sentences that are the vertices.
- The graph has edges denoting the similarity between the two sentences at the vertices.
- Run PageRank algorithm on this graph. This calculates the cosine similarities between sentences and assigns weight with each sentence as node.
- Pick the highest-scoring vertices and append them to the summary.

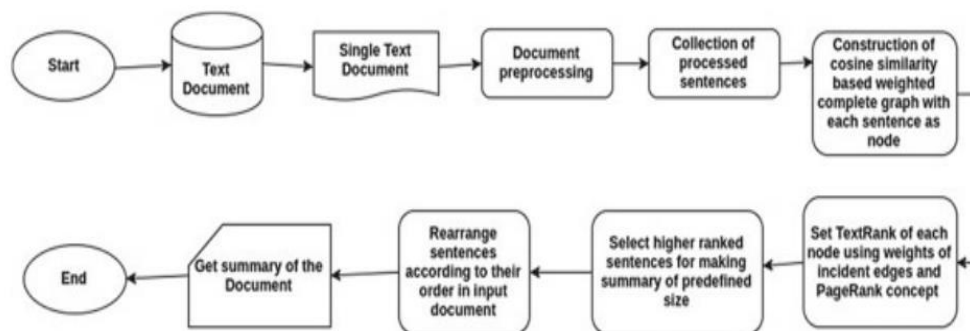


Fig 4.4.2.1: Text Rank Structure

TextRank is very well suited for applications involving entire sentences, since it allows for a ranking over text units that is recursively computed based on information drawn from the entire text. To apply TextRank, we first build a graph associated with the text, where the graph vertices are representative for the units to be ranked. The goal is to rank entire sentences. Therefore, a vertex is added to the graph for each sentence in the text.

```

def summarize(url_topull, num_of_words):

    scraped_data = urllib.request.urlopen(url_topull)
    article = scraped_data.read()

    parsed_article = bs.BeautifulSoup(article, 'lxml')
    paragraphs = parsed_article.find_all('p')
    article_text = ""
    for p in paragraphs:
        article_text += p.text

    stop_words = set(stopwords.words('english'))
    keywords = mz_keywords(article_text, scores=True, threshold=0.003)
    keywords_names = []
    for tuples in keywords:
        if tuples[0] not in stop_words:
            if len(tuples[0]) > 2:
                keywords_names.append(tuples[0])

    pre_summary = su_gs(article_text, word_count=num_of_words)

    summary = re.sub("[\(\[\].*?[\]\]]", "", pre_summary)

```

Fig 4.4.2.2: Text Rank Pseudo Code for Wikipedia pages.

The output summary will consist of the most representative sentences and will also be returned as a string, divided by newlines. If the split parameter is set to True, a list of sentences will be returned (Fig 4.4.2.2).

4.4.3 Custom Frequency Distribution Algorithm

It works on the principle of important keywords identification based on their frequency of appearance in the input text. The news article and custom text modules use the custom algorithm to summarize text (Fig 4.4.3.1).

Algorithm:

- Pre-process the given text. This includes stop words removal, punctuation removal, and stemming.
- Tokenize the text into sentences and words.
- Create a frequency distribution table of the words by using the formula:

Word Frequency / Maximum Frequency (Freq of word appearing most in text)

- Calculate sentence scores by adding frequency scores of each word present in the sentence.
- Generate a summary based on the highest sentence scores.

```
if url and no_of_sentences and st.button('Summarize News Article'):  
    text = ""  
    article = Article(url)  
    article.download()  
    article.parse()  
    nltk.download('punkt')  
    #article.nlp  
    text = article.text  
    text = re.sub(r'\[[0-9]*\]', ' ', text)  
    text = re.sub(r'\s+', ' ', text)  
  
    st.subheader('Original text: ')  
    st.write(text)  
  
    tokens = tokenizer(text)  
    sents = sent_tokenizer(text)  
    word_counts = count_words(tokens)  
    freq_dist = word_freq_distribution(word_counts)  
    sent_scores = score_sentences(sents, freq_dist)  
    summary, summary_sent_scores = summarize(sent_scores, no_of_sentences)  
  
    st.subheader('Summarised text: ')  
    st.write(summary)
```

Fig 4.4.3.1: Pseudo Code for News Article Summarization.

The original text extracted from the news article is displayed first, followed by the summary generated. The sentence scores are displayed below the summary.

4.5 Data Description

Each module uses specific techniques to extract data. The news article module extracts articles using a newspaper library in python. The python newspaper library is used to collect information associated with articles. This includes author name, major images in the article, publication dates, video present in the article, key words describing the article and the summary of the article. This API works with news sites using HTML tags in the base of the code.

The Wikipedia pages module extracts articles using the urllib and beautiful soup library in python. Urllib module is the URL handling module for python. It is used to fetch URLs. It uses the urlopen function and is able to fetch URLs using a variety of different protocols. We use the urllib.request function to open the URL for extraction, and use beautiful soup to extract all lxml tagged data. Then a search for all paragraph tags in this data reveals the Wikipedia information required for summarization. This technique allows any Wikipedia page to be summarized.

For the YouTube video module, the title and text material are extracted from the video ID using re and youtube_transcript_api library respectively in Python. Youtube transcript API is an open-source software project. It is a python API which allows you to get the transcript/subtitles for a given YouTube video. It works for automatically generated subtitles, supports translating subtitles without requiring the need for a headless browser. We use re to parse through the YouTube page to search for the title, which is then extracted and stored into a variable. This technique works with most of the YouTube videos which have user defined subtitles. Since, auto-generated subtitles of YouTube are still in the beta stage, they cannot be relied upon to summarize the data.

```
url = st.text_input('\nEnter URL of a news article: ')
wikiurl = st.text_input('\nEnter URL of Wikipedia: ')
video_id = st.text_input("\nEnter a Youtube Video Id: ")
textfield123 = st.text_area('\nEnter an article or paragraph you want to summarize: ')
```

Fig 4.5.1: Variables containing URL/videoID.

The custom text input does not require data extraction and can take input of any text. However, articles, or paragraphs are only supported because any other random text is not summarization capable (Fig 4.5.1).

5. RESULTS AND DISCUSSIONS

Each module is run in real time with latest news and updates Wikipedia pages along with speech related YouTube videos. A sample custom text input is also considered to demonstrate the text to speech capabilities of the application.

```
C:\Users\ammar\OneDrive\ImportantFiles\MSA\Projects\Automatic Text Summary
\Build\ATS\finalv>streamlit run ats.py

You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.0.112:8501
```

Fig 5.1: Running the streamlit application.

The web application is deployed on a local host after running the main module of the code (Fig 5.1). The user interface is displayed in Fig (5.2).

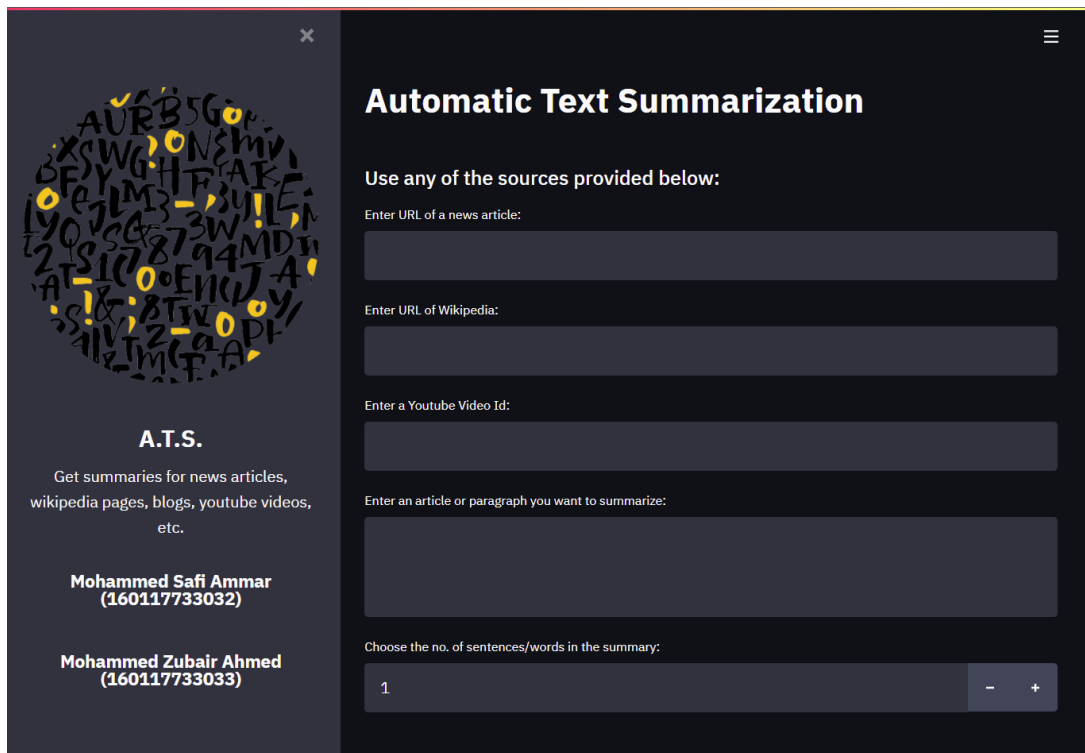
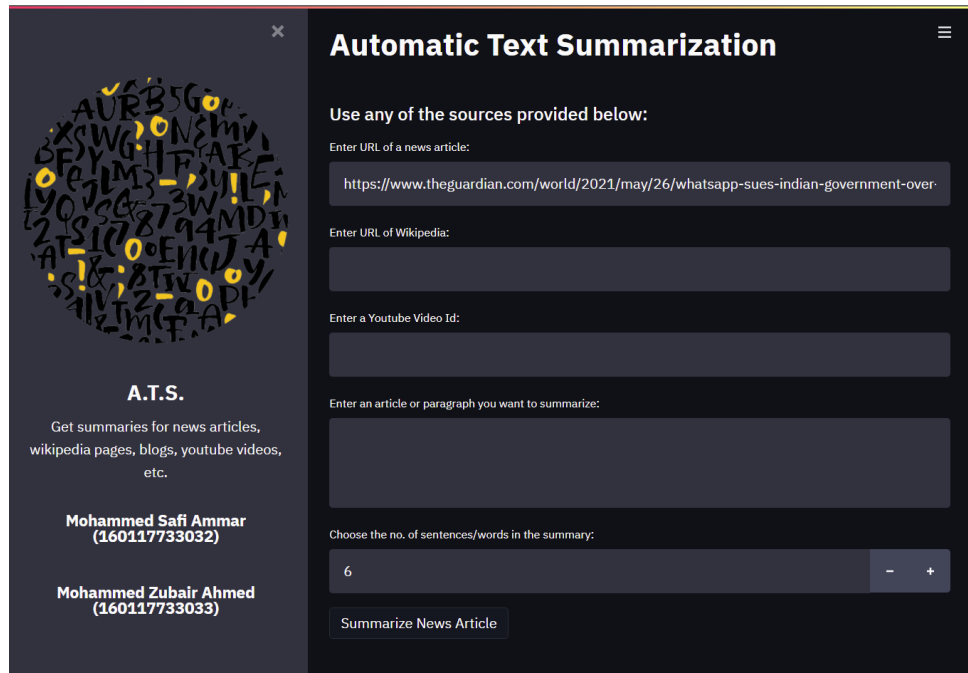


Fig 5.2: User interface of the application.

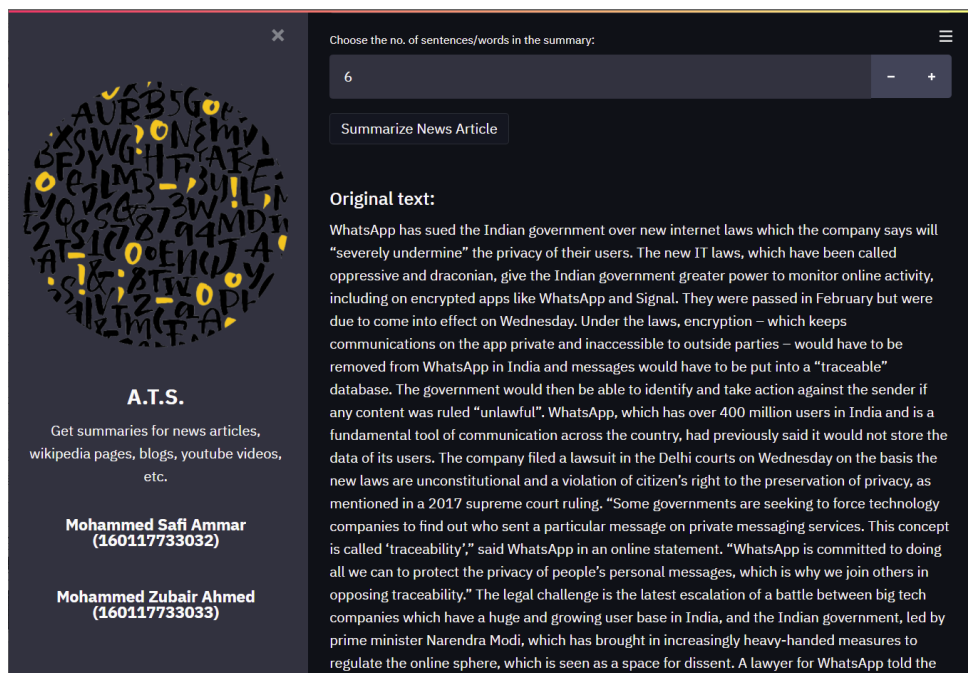
5.1 News Article Module



The screenshot shows a web application titled "Automatic Text Summarization". On the left is a sidebar with a circular logo containing various characters and the text "A.T.S. Get summaries for news articles, wikipedia pages, blogs, youtube videos, etc." Below this are the names and contact numbers of Mohammed Safi Ammar and Mohammed Zubair Ahmed. The main area on the right has the title "Automatic Text Summarization" and a sub-header "Use any of the sources provided below:". It contains four input fields: "Enter URL of a news article:" (with a pre-filled URL from The Guardian), "Enter URL of Wikipedia:", "Enter a Youtube Video Id:", and "Enter an article or paragraph you want to summarize:". Below these is a "Choose the no. of sentences/words in the summary:" section with a numeric input set to 6 and minus/plus buttons. A "Summarize News Article" button is at the bottom.

Fig 5.1.1: News article URL input.

The news article shown is from “The Guardian” website, dated on 26th May 2021. The choice of a six-sentence summary is made (Fig 5.1.1).



This screenshot shows the same application after clicking the "Summarize News Article" button. The "Choose the no. of sentences/words in the summary:" input remains at 6. The "Summarize News Article" button is now disabled. The "Original text:" section is expanded, displaying the full text of the article about WhatsApp suing the Indian government. The sidebar on the left remains unchanged.

Fig 5.1.2: Original text of news article.

Summarised text:

The new IT laws, which have been called oppressive and draconian, give the Indian government greater power to monitor online activity, including on encrypted apps like WhatsApp and Signal. WhatsApp has sued the Indian government over new internet laws which the company says will “severely undermine” the privacy of their users. A lawyer for WhatsApp told the Delhi high court: “A government that chooses to mandate traceability is effectively mandating a new form of mass surveillance. The company filed a lawsuit in the Delhi courts on Wednesday on the basis the new laws are unconstitutional and a violation of citizen’s right to the preservation of privacy, as mentioned in a 2017 supreme court ruling.” The Modi government has already clashed repeatedly with Twitter, demanding that the site remove anti-government tweets related to the farmers’ protests earlier this year and more recently tweets which criticised the government’s handling of the pandemic. Under the new IT rules, social media companies have to remove content within 36 hours of a legal order and have to appoint an Indian-based “compliance officer” to deal with any complaints.

Summary sentence score for the top 6 sentences:

	Sentence	Score
0	The new IT laws, which have been called oppressive and draconian, give the Indian government greater power to monitor online activity, including on encrypted apps like WhatsApp and Signal	5.8571
1	WhatsApp has sued the Indian government over new internet laws which the company says will “severely undermine” the privacy of their users	5.1429
2	A lawyer for WhatsApp told the Delhi high court: “A government that chooses to mandate traceability is effectively mandating a new form of mass surveillance	4.8571
3	The company filed a lawsuit in the Delhi courts on Wednesday on the basis the new laws are unconstitutional and a violation of citizen’s right to the preservation of privacy, as mentioned in a 2017 supreme court ruling	4.7143
4	” The Modi government has already clashed repeatedly with Twitter, demanding that the site remove anti-government tweets related to the farmers’ protests earlier this year and more recently tweets which criticised the government’s handling of the pandemic	4.7143

Fig 5.1.3: Summarized news article with sentence scores.

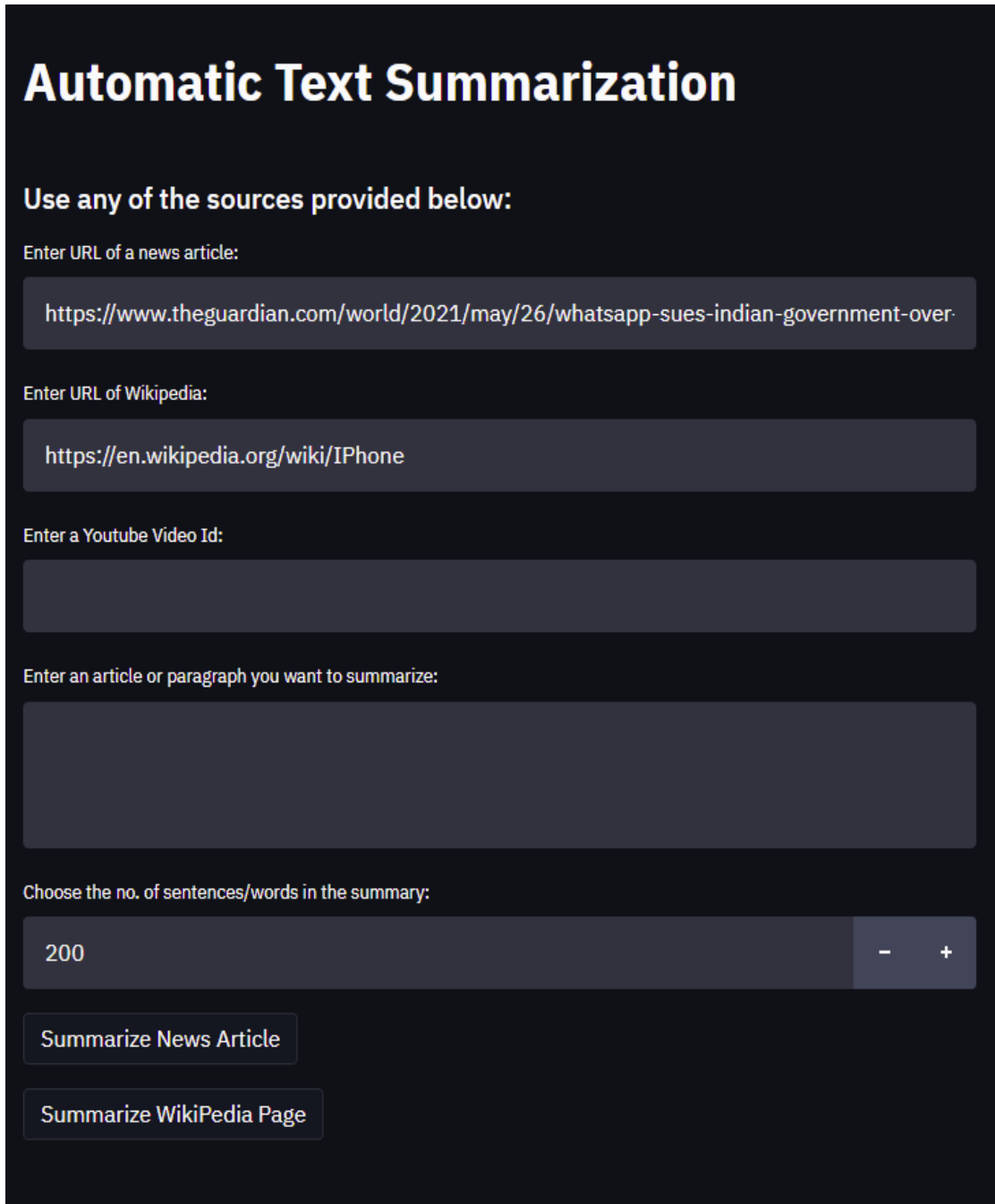
The Fig (5.1.2) shows the original news article text and Fig (5.1.3) shows the summarized news article followed by the sentence scores of each sentence from the original text.

Table 5.1.1: Sentence scores of the news article.

RANK	SENTENCE	SCORE
0	The new IT laws, which have been called oppressive and draconian, give the Indian government greater power to monitor online activity, including on encrypted apps like WhatsApp and Signal	5.8571
1	WhatsApp has sued the Indian government over new internet laws which the company says will “severely undermine” the privacy of their users	5.1429
2	A lawyer for WhatsApp told the Delhi high court: “A government that chooses to mandate traceability is effectively mandating a new form of mass surveillance	4.8571
3	The company filed a lawsuit in the Delhi courts on Wednesday on the basis the new laws are unconstitutional and a violation of citizen’s right to the preservation of privacy, as mentioned in a 2017 supreme court ruling	4.7143
4	The government has already clashed repeatedly with Twitter, demanding that the site remove anti-government tweets related to the farmers’ protests earlier this year and more recently tweets which criticized the government’s handling of the pandemic	4.7143
5	Under the new IT rules, social media companies have to remove content within 36 hours of a legal order and have to appoint an Indian-based “compliance officer” to deal with any complaints	4.1429

The sentence scores are calculated based on the frequency of appearance of each word in the summary i.e., sum frequency distribution values of the words. This result demonstrates the effectiveness of the custom algorithm for news articles. According to the ranks obtained the summary is arranged and displayed on the application interface (Table 5.1.1).

5.2 Wikipedia Pages Module



Automatic Text Summarization

Use any of the sources provided below:

Enter URL of a news article:

<https://www.theguardian.com/world/2021/may/26/whatsapp-sues-indian-government-over->

Enter URL of Wikipedia:

<https://en.wikipedia.org/wiki/IPhone>

Enter a Youtube Video Id:

Enter an article or paragraph you want to summarize:

Choose the no. of sentences/words in the summary:

200 - +

Summarize News Article

Summarize WikiPedia Page

Fig 5.2.1: Wikipedia Page URL input.

The summary is generated using gensim after extraction of data from the Wikipedia page “Iphone”. The choice of a 200-word summary is made (Fig 5.2.1).

Summarised Wikipedia Page:

Jobs unveiled the iPhone to the public on January 9, 2007, at the Macworld 2007 convention at the Moscone Center in San Francisco. The two initial models, a 4 GB model priced at US\$499 and an 8 GB model at US\$599, went on sale in the United States on June 29, 2007, at 6:00 pm local time, while hundreds of customers lined up outside the stores nationwide. The passionate reaction to the launch of the iPhone resulted in sections of the media dubbing it the 'Jesus phone'. Following this successful release in the US, the first generation iPhone was made available in the UK, France, and Germany in November 2007, and Ireland and Austria in the spring of 2008. A 3-axis accelerometer senses the orientation of the phone and changes the screen accordingly, allowing the user to easily switch between portrait and landscape mode. Photo browsing, web browsing, and music playing support both upright and left or right widescreen orientations. Unlike the iPad, the iPhone does not rotate the screen when turned upside-down, with the Home button above the screen, unless the running program has been specifically designed to do so.

Keywords: 'camera', 'iphone', 'sim', 'inches', 'screen', 'battery', 'plus'

Fig 5.2.2: Generated Wikipedia page summary.

The summary of the entire Wikipedia article along with the keywords categorized as important by the gensim function is displayed to the user (Fig 5.2.2).

5.3 YouTube Video Module

Enter a Youtube Video Id:

UUheH1seQuE

Enter an article or paragraph you want to summarize:

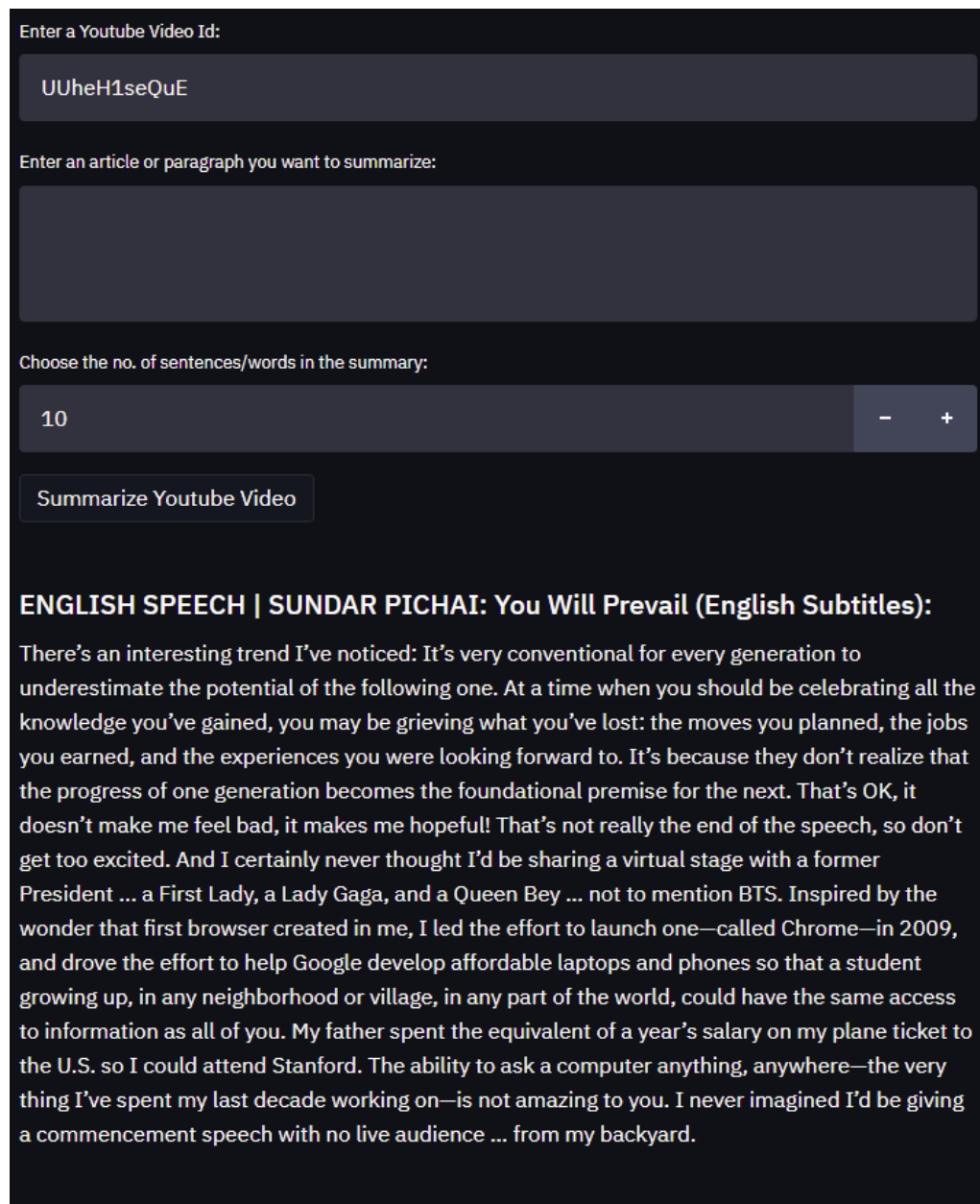
Choose the no. of sentences/words in the summary:

10 - +

Summarize Youtube Video

Fig 5.3.1: Youtube Video ID input.

The summary is generated after extraction of title and captions using the youtube_transcript_api from the video “SUNDAR PICHAI: You Will Prevail”. The choice of a 10-sentence summary is made (Fig 5.3.1).



Enter a Youtube Video Id:

UUheH1seQuE

Enter an article or paragraph you want to summarize:

Choose the no. of sentences/words in the summary:

10 - +

Summarize Youtube Video

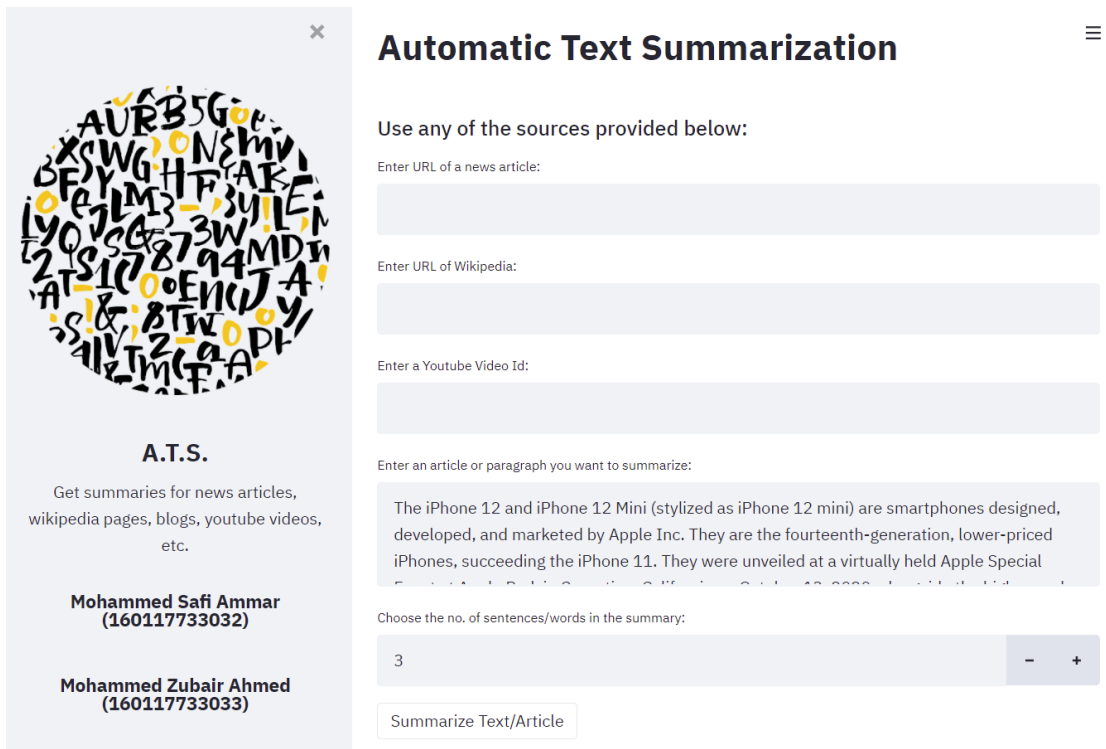
ENGLISH SPEECH | SUNDAR PICHAI: You Will Prevail (English Subtitles):

There's an interesting trend I've noticed: It's very conventional for every generation to underestimate the potential of the following one. At a time when you should be celebrating all the knowledge you've gained, you may be grieving what you've lost: the moves you planned, the jobs you earned, and the experiences you were looking forward to. It's because they don't realize that the progress of one generation becomes the foundational premise for the next. That's OK, it doesn't make me feel bad, it makes me hopeful! That's not really the end of the speech, so don't get too excited. And I certainly never thought I'd be sharing a virtual stage with a former President ... a First Lady, a Lady Gaga, and a Queen Bey ... not to mention BTS. Inspired by the wonder that first browser created in me, I led the effort to launch one—called Chrome—in 2009, and drove the effort to help Google develop affordable laptops and phones so that a student growing up, in any neighborhood or village, in any part of the world, could have the same access to information as all of you. My father spent the equivalent of a year's salary on my plane ticket to the U.S. so I could attend Stanford. The ability to ask a computer anything, anywhere—the very thing I've spent my last decade working on—is not amazing to you. I never imagined I'd be giving a commencement speech with no live audience ... from my backyard.

Fig 5.3.2: Youtube Video summary.

The title is extracted from the video ID first using regular expression package in python followed by the extraction of captions. The captions are then summarized to display the summary (Fig 5.3.2).

5.4 Custom Text Input Module



The screenshot shows a web application titled "Automatic Text Summarization". On the left is a sidebar with a circular logo containing letters and numbers, the text "A.T.S.", and contact information for Mohammed Safi Ammar and Mohammed Zubair Ahmed. The main area has a heading "Automatic Text Summarization" and a subheading "Use any of the sources provided below:". It contains four input fields: "Enter URL of a news article:", "Enter URL of Wikipedia:", "Enter a Youtube Video Id:", and "Enter an article or paragraph you want to summarize:". The last field is populated with a paragraph about the iPhone 12 and iPhone 12 Mini. Below these fields is a "Choose the no. of sentences/words in the summary:" section with a numeric input set to 3 and minus/plus buttons. A "Summarize Text/Article" button is at the bottom.

Fig 5.4.1: Input of custom article.

The summary is generated using the provided article straightaway after data cleaning since custom input requires no data extraction. The choice of a 3-sentence summary is made (Fig 5.4.1). The summary is read aloud by the pyttsx3 text to speech API (Fig 5.4.2).

```
You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.0.112:8501

2021-05-26 21:30:18.140 Imported existing <module 'comtypes.gen' from 'c:\\
users\\ammar\\appdata\\local\\programs\\python\\python37\\lib\\site-package
s\\comtypes\\gen\\__init__.py'>
2021-05-26 21:30:18.140 Using writeable comtypes cache directory: 'c:\\users
\\ammar\\appdata\\local\\programs\\python\\python37\\lib\\site-packages\\comtypes\\ge
n'
Playing summary audio...
Please turn up your volume!
Summary playback ended.
```

Fig 5.4.2: Summary audio playback.

Automatic Text Summarization

Use any of the sources provided below:

Enter URL of a news article:

Enter URL of Wikipedia:

Enter a Youtube Video Id:

Enter an article or paragraph you want to summarize:

The iPhone 12 and iPhone 12 Mini (stylized as iPhone 12 mini) are smartphones designed, developed, and marketed by Apple Inc. They are the fourteenth-generation, lower-priced iPhones, succeeding the iPhone 11. They were unveiled at a virtually held Apple Special Event at Apple Park in Cupertino, California on October 13, 2020, alongside the higher-end iPhone 12 Pro and iPhone 12 Pro Max flagships.

Choose the no. of sentences/words in the summary:

3

- +

Summarize Text/Article

Summarised Text/Article:

The iPhone 12 and iPhone 12 Mini, like the iPhone 12 Pro and iPhone 12 Pro Max, are the first iPhone models from Apple to no longer include a power adapter or EarPods headphones found in prior iPhone models; however, a USB-C to Lightning cable is still included; and this change was retroactively applied to other iPhone models sold by Apple, such as the iPhone 11 and the second iPhone SE. The iPhone 12 and iPhone 12 Mini (stylized as iPhone 12 mini) are smartphones designed, developed, and marketed by Apple Inc. They were unveiled at a virtually held Apple Special Event at Apple Park in Cupertino, California on October 13, 2020, alongside the higher-end iPhone 12 Pro and iPhone 12 Pro Max flagships.

Fig 5.4.3: Custom text input summary.

As shown, the summary audio is played as soon as the summary is generated. The application can be used in both the dark mode, and a light mode owing to the functionality and features of the streamlit framework (Fig 5.4.3).

6. CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

Owing to the speedy growth of Internet and Technology, there is an overload of informatic and textual data. This time-consuming scenario can be made much more effective and efficient if there are strong text summarizers which produce a summary of any document to help a user. In the last few years, Automatic Text Summarization has been a popular research area and attracts a lot of attention from different scientific branches. Summarization methods can be classified into extractive and abstractive summarization.

An extractive summary is selection of important sentences from the original text. Abstractive Summary is a method for novel phrasing describing the content of the text which requires heavy machinery from natural language processing, including grammars and lexicons for parsing and generation. We focus on extractive summarization methods to develop a summarization model which is used as the basis of a web application to generate summaries from numerous sources. The application consists of various modules which summarize different types of content, such as web pages (Wikipedia), news articles, user generated text, and YouTube videos. All the modules deliver an effective, legible, and comprehensible summary.

6.1.1 Applications

Summarization presents plethora of applications when it comes to handling anything related to data.

- **Applications in search engines**

The current search engines are incorporating text summarization into displaying search results. For example, Google makes a short text summarization of the most important item and places the summary at the head of the list of search results in response to the queries.

- **YouTube comments summarization**

YouTube algorithm also considers the comment section for recommending a video or to provide search results. However, the huge amounts of comments need to be summarized into keywords which can then be used by the algorithm.

- **Books and literature**

Large e-commerce websites like Amazon have 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.

- **Internal document workflow**

Large companies are constantly producing internal knowledge, which frequently gets stored and under-used in databases as unstructured data. These companies should embrace tools that let them re-use already existing knowledge. Summarization can enable analysts to quickly understand everything the company has already done in a given subject, and quickly assemble reports that incorporate different points of view.

- **Financial research**

Investment banking firms spend large amounts of money acquiring information to drive their decision-making, including automated stock trading. When you are a financial analyst looking at market reports and news every day, you will inevitably hit a wall and will not be able to read everything. Summarization systems tailored to financial documents like earning reports and financial news can help analysts quickly derive market signals from content.

- **Social media marketing**

Companies producing long-form content, like whitepapers, e-books, and blogs, might be able to leverage summarization to break down this content and make it sharable on social media sites like Twitter or Facebook. This would allow companies to further re-use existing content.

- **Question answering and Bots**

Personal assistants are taking over the workplace and the smart home. However, most assistants are fairly limited to very specific tasks. Large-scale summarization could become a powerful question answering technique. By collecting the most relevant documents for a particular question, a summarizer could assemble a cohesive answer in the form of a multi-document summary.

- **Email overload**

Companies like Slack were born to keep us away from constant emailing. Summarization could surface the most important content within email and let us skim emails faster.

- **Science and R&D & Patent research**

Academic papers typically include a human-made abstract that acts as a summary. However, when you are tasked with monitoring trends and innovation in a given sector, it can become overwhelming to read every abstract. Systems that can group papers and further compress abstracts can become useful for this task. Similarly Researching patents can be a tedious process. Whether you are doing market intelligence research or looking to file a new patent, a summarizer to extract the most salient claims across patents could be a time saver.

6.1.2 Limitations

While the modules of the application work efficiently, some limitations arise when it comes to processing data from miscellaneous websites.

- **Quality of summary**

The quality of the summary can further be improved to rival that of abstractive methods. Though the summaries are more than feasible, they provide no additional input or information. The summaries are still far behind human made summaries. This is one area, where summarization models are dependent on the development of NLP and related fields.

- **Limited availability**

Although NLP provides support for many languages, its still not comparable to the 6000+ languages present worldwide. It can be incredibly difficult to models for all of them.

- **YouTube Video Captioning**

Since the auto-generated captions feature on YouTube is still in beta testing, it produces captions which are inaccurate and unpunctuated. This makes summarizing videos without user defined captions almost impracticable.

6.2 Future Scope

Summarization can present many promising projects and ideas in the future. Ever increasing APIs provided by third party software's can introduce unforeseen ways of summarization.

- **Meetings and Videoconferencing**

With the growth of tele-working, the ability to capture key ideas and content from conversations is increasingly needed. A system that could turn voice to text and generate summaries from your team meetings would be fantastic.

- **Help desk and Customer Support**

Knowledge bases have been around for a while, and they are critical for Software as a Service (SAAS) platforms to provide customer support at scale. Still, users can sometimes feel overwhelmed when browsing help documents. Multi-document summarization could provide key points from help articles and give the user a well-rounded understanding of the issue.

- **Programming Languages**

There have been multiple attempts to build AI technology that could write code and build websites by itself. It is a possibility that custom “code summarizers” will emerge to help developers get the big picture out of a new project.

- **Automated Content Creation**

If artificial intelligence is able to replace any stage of the content creation process, automatic summarization is likely going to play an important role. Writing a good blog usually goes by summarizing existing sources for a given query. Summarization technology might reach a point where it can compose an entirely original article out of summarizing related search results.

REFERENCES

- [1]. Soni V., Kumar L., Singh A.K., Kumar M., "Text Summarization: An Extractive Approach", 2020 In: Pant M., Kumar Sharma T., Arya R., Sahana B., Zolfagharinia H. "Soft Computing: Theories and Applications", vol 1154. Springer, Singapore. https://doi.org/10.1007/978-981-15-4032-5_57

- [2]. J. N. Madhuri and R. Ganesh Kumar, "Extractive Text Summarization Using Sentence Ranking", 2019 International Conference on Data Science and Communication (IconDSC), 2019, pp. 1-3, doi: 10.1109/IconDSC.2019.8817040.

- [3]. A. T. Al-Taani, "Automatic text summarization approaches," 2017 International Conference on Infocom Technologies and Unmanned Systems (Trends and Future Directions) (ICTUS), 2017, pp. 93-94, doi: 10.1109/ICTUS.2017.8285983.

- [4]. P. Sethi, S. Sonawane, S. Khanwalker and R. B. Keskar, "Automatic text summarization of news articles," 2017 International Conference on Big Data, IoT and Data Science (BIGDATA), 2017, pp. 23-29, doi: 10.1109/BID.2017.8336568.

- [5]. P. P. Tardan, A. Erwin, K. I. Eng and W. Muliady, "Automatic text summarization based on semantic analysis approach for documents in Indonesian language", 2013 International Conference on Information Technology and Electrical Engineering (ICITEE), 2013, pp. 47-52, doi: 10.1109/ICITEED.2013.6676209.

- [6]. P. Zhang and C. Li, "Automatic text summarization based on sentences clustering and extraction", 2009 2nd IEEE International Conference on Computer Science and Information Technology, 2009, pp. 167-170, doi: 10.1109/ICCSIT.2009.5234971.

APPENDIX

Main Python Class

```
import streamlit as st

from bs4 import BeautifulSoup

import requests

import re

from collections import Counter

from string import punctuation

from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS as stop_words

from youtube_transcript_api import YouTubeTranscriptApi as ytapi

import pandas as pd

import bs4 as bs

import urllib.request

from PIL import Image

from gensim.summarization import summarize as su_gs

from gensim.summarization import keywords

from gensim.summarization import mz_keywords

from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize

import os

from nltk.tokenize import sent_tokenize

from nltk.probability import FreqDist
```

```

from heapq import nlargest

from collections import defaultdict

import nltk

from newspaper import Article

import pyttsx3


def audio(summary):

    print("Playing summary audio...")

    engine = pyttsx3.init()

    summary = str(summary)

    print("Please turn up your volume!")

    engine.say(summary)

    engine.runAndWait()

    engine.stop()

    print("Summary playback ended.")


def tokenizer(s):

    tokens = []

    for word in s.split(' '):

        tokens.append(word.strip().lower())

    return tokens


def sent_tokenizer(s):

```

```

sents = []

for sent in s.split('.'):

    sents.append(sent.strip())

return sents


def count_words(tokens):

    word_counts = { }

    for token in tokens:

        if token not in stop_words and token not in punctuation:

            if token not in word_counts.keys():

                word_counts[token] = 1

            else:

                word_counts[token] += 1

    return word_counts


def word_freq_distribution(word_counts):

    freq_dist = { }

    max_freq = max(word_counts.values())

    for word in word_counts.keys():

        freq_dist[word] = (word_counts[word]/max_freq)

    return freq_dist

```

```

def score_sentences(sents, freq_dist, max_len=40):

    sent_scores = {}

    for sent in sents:

        words = sent.split(' ')

        for word in words:

            if word.lower() in freq_dist.keys():

                if len(words) < max_len:

                    if sent not in sent_scores.keys():

                        sent_scores[sent] = freq_dist[word.lower()]

                    else:

                        sent_scores[sent] += freq_dist[word.lower()]

        return sent_scores

def summarize(sent_scores, k):

    top_sents = Counter(sent_scores)

    summary = ""

    scores = []

    top = top_sents.most_common(k)

    for t in top:

        summary += t[0].strip() + ' '

        scores.append((t[1], t[0]))

```

```

return summary[: -1], scores

st.title('Automatic Text Summarization')

st.subheader('Use any of the sources provided below:')


image = Image.open('circle-cropped.png')

st.sidebar.image(image, use_column_width=True)

st.sidebar.markdown('<center> <h1>A.T.S.</h1></center>',unsafe_allow_html=True)

st.sidebar.markdown('<center> Get summaries for news articles, wikipedia pages,
blogs, youtube videos, etc.</center>',unsafe_allow_html=True)

st.sidebar.markdown('<center>          <h3>Mohammed          Safi          Ammar
(160117733032)</h3></center>',unsafe_allow_html=True)


url = st.text_input("\nEnter URL of a news article: ")

wikiurl = st.text_input("\nEnter URL of Wikipedia: ")

video_id = st.text_input("\nEnter a Youtube Video Id: ")

# video_id = "Na8vHaCLwKc"

textfield123 = st.text_area("\nEnter an article or paragraph you want to summarize: ")

no_of_sentences = st.number_input('Choose the no. of sentences/words in the
summary: ', min_value = 1)

```

```

def textfunc():

    content = textfield123

    content = sanitize_input(content)


    sent_tokens, word_tokens = tokenize_content(content)

    sent_ranks = score_tokens(sent_tokens, word_tokens)

    st.subheader('Summarised Text/Article: ')

    st.write(summarize2(sent_ranks, sent_tokens, no_of_sentences))

    text = str(summarize2(sent_ranks, sent_tokens, no_of_sentences))

    audio(text)


def textforYT():

    #transcript_list = ytapi.list_transcripts(video_id)

    #transcript = transcript_list.find_manually_created_transcript(['en'])

    #a = transcript.fetch()

    vurl = 'https://www.youtube.com/watch?v=%s' % video_id

    response = requests.get(vurl).text

    title = re.findall(r'"title": "[^>]*"', response)[0].split(',')[0][9:-1]


    a = ytapi.get_transcript(video_id)

    textstr = ""

    for i in a:

        temp = ""

```

```

temp = '' + i["text"]

textstr += temp #i["text"]

#article = [textstr[i:i + 100] for i in range(0, len(textstr), 100)]

#res = '. '.join(textstr[i:i + 100] for i in range(0, len(textstr), 100))

#content = res

#sent_tokens, word_tokens = tokenize_content(content)

sent_tokens, word_tokens = tokenize_content(textstr)

sent_ranks = score_tokens(sent_tokens, word_tokens)

st.subheader(title+' : ')

#st.write(textstr)

st.write(summarize2(sent_ranks, sent_tokens, no_of_sentences))

```

```

def tokenize_content(content):

```

```

    stop_words = set(stopwords.words('english') + list(punctuation))

    words = word_tokenize(content.lower())

    return (sent_tokenize(content), [word for word in words if word not in stop_words])

```

```

def score_tokens(sent_tokens, word_tokens):

```

```

    word_freq = FreqDist(word_tokens)

    rank = defaultdict(int)

    for i, sent in enumerate(sent_tokens):

```



```

    for word in word_tokenize(sent.lower()):

        if word in word_freq:

            rank[i] += word_freq[word]

    return rank


def sanitize_input(data):

    replace = {

        ord('\f') : ' ',

        ord('\t') : ' ',

        ord('\n') : ' ',

        ord('\r') : None

    }

    return data.translate(replace)


def summarize2(ranks, sentences, length):

    if int(length) > len(sentences):

        st.write('You requested more sentences in the summary than there are in the text.')

        #print('You requested more sentences in the summary than there are in the text.')

        return "

    else:

        indices = nlargest(int(length), ranks, key=ranks.get)

        final_summary = [sentences[j] for j in indices]

```

```
return ' '.join(final_summary)
```

```
if url and no_of_sentences and st.button('Summarize News Article'):
```

```
    text = ""
```

```
    article = Article(url)
```

```
    article.download()
```

```
    article.parse()
```

```
    nltk.download('punkt')
```

```
    #article.nlp
```

```
    text = article.text
```

```
    #r=requests.get(url)
```

```
    #soup = BeautifulSoup(r.content, 'html.parser')
```

```
    #content = soup.find('div', attrs = {'id' : re.compile('content-body-14269002-*')})
```

```
    #for p in content.findChildren("p", recursive = 'False'):
```

```
        # text+=p.text+" "
```

```
    text = re.sub(r'[[0-9]*\]', ' ', text)
```

```
    text = re.sub(r'\s+', ' ', text)
```

```
    st.subheader('Original text: ')
```

```
    st.write(text)
```

```

tokens = tokenizer(text)

sents = sent_tokenizer(text)

word_counts = count_words(tokens)

freq_dist = word_freq_distribution(word_counts)

sent_scores = score_sentences(sents, freq_dist)

summary, summary_sent_scores = summarize(sent_scores, no_of_sentences)


st.subheader('Summarised text: ')

st.write(summary)


subh = 'Summary sentence score for the top ' + str(no_of_sentences) + ' sentences: '

st.subheader(subh)


data = []


for score in summary_sent_scores:

    data.append([score[1], score[0]])


df = pd.DataFrame(data, columns = ['Sentence', 'Score'])


st.table(df)


st.info('An application made by students of CBIT, Hyderabad.')

```

```

def print_usage():

    # Display the parameters and what they mean.

    st.write("""

    Usage:

        summarize.py <wiki-url> <summary length>

    Explanation:

        Parameter 1: Wikipedia URL to pull

        Parameter 2: the number of words for the summary to contain

    """)


def summarize(url_topull, num_of_words):

    scraped_data = urllib.request.urlopen(url_topull)

    article = scraped_data.read()

    parsed_article = bs.BeautifulSoup(article,'lxml')

    paragraphs = parsed_article.find_all('p')

    article_text = ""

    for p in paragraphs:

        article_text += p.text

    stop_words = set(stopwords.words('english'))

    keywords = mz_keywords(article_text,scores=True,threshold=0.003)

```

```

keywords_names = []

for tuples in keywords:

    if tuples[0] not in stop_words:

        if len(tuples[0]) > 2:

            keywords_names.append(tuples[0])


pre_summary = su_gs(article_text,word_count=num_of_words)


summary = re.sub("[\(\[].*?[\)\]]", "", pre_summary)


print_pretty (summary,keywords_names)


def print_pretty (summary, keywords_names):

    columns = os.get_terminal_size().columns


    printable = summary


    st.subheader('Summarised Wikipedia Page: ')


    st.write(printable.center(columns))


    str_keywords_names = str(keywords_names).strip('[]')


    printable2 = str_keywords_names


    st.write('Keywords: ' + printable2.center(columns))


if wikiurl and no_of_sentences and st.button('Summarize WikiPedia Page'):

```

```

if not str(no_of_sentences).isdigit():

    print_usage()

else:

    summarize(wikiurl, int(no_of_sentences))


if video_id and no_of_sentences and st.button('Summarize Youtube Video'):

    if not str(no_of_sentences).isdigit():

        st.write("Use it again. Error occurred summarizing article.")

    else:

        textforYT()


if textfield123 and no_of_sentences and st.button('Summarize Text/Article'):

    if not str(no_of_sentences).isdigit():

        st.write("Use it again. Error occurred summarizing article.")

    else:

        textfunc()

```