A PROJECT REPORTON

TEXT SUMMARIZER

BY

KRATI TIWARI: (18223)
MUSKAN GUPTA: (18228)
SUCHITA YADAV: (18252)

UNDER THE SUPERVISION OF
PROF. SUMAN PANDEY

A DISSERTATION IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF DEGREE

BACHELOR OF TECHNOLOGYIN COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

KAMLA NEHRU INSTITUTE OF TECHNOLOGY SULTANPUR (U.P.) – 228118

(An Autonomous State Government Institute)

AFFILIATED TO

DR A. P. J. ABDUL KALAM TECHNICAL UNIVERSITY
LUCKNOW (U.P.) INDIA
2021-2022

Department Of Computer Science and Engineering

Kamla Nehru Institute of Technology

Sultanpur (U.P.)- 228118

Certificate

This is to certify that Krati Tiwari (18223), Muskan Gupta (18228) and Suchita Yadav (18252) have carried out the project work in this report entitled "Text Summarizer" for the award of Bachelor of Technology in Computer Science and Engineering at Kamla Nehru of Technology, Sultanpur, affiliated to Dr A. P. J Abdul Kalam Technical University, Lucknow. This report is the record of candidates' own work carried out by them under our supervision and guidance. This project work is the part of their Bachelor of Technology in Computer Science and Engineering curriculum. Their performance was excellent and we wish them good luck for their future endeavors.

…………………………	**…………………………**
Prof. Suman Pandey	**Prof. Awadhesh Kumar**
(Project guide and project in-charge)	**(Head of Department)**

Declaration

We hereby declare that this submission is our own work and that, to the rest of our knowledge and belief, it contains no material previously or written by another person nor material which to a substantial extent has been accepted for the award of any other degree of the university or other institute of higher learning, except where due acknowledgement has been made in text.

Krati Tiwari
(18223)

Muskan Gupta
(18228)

Suchita Yadav
(18252)

Acknowledgement

We wish to express our sincere gratitude to Prof. Suman Pandey (**Project Guide and Project In-Charge**) for his valuable suggestions and guidance throughout our work at Kamla Nehru Institute of Technology, Sultanpur. She has guided us through the difficulties and made us understood the concepts needed for the project work. Their experimental and theoretical knowledge has been very helpful. We feel privileged in expressing department for their encouragement and moral support.

We would like to thank Prof. Awadesh Kumar, Head of the Department, Computer Science & Engineering for providing us a golden opportunity to work with him. The support and then environment provided to us during our project was more than what anyone would have expected.

We are also deeply indebted to all those without whose firm support, encouragement and guidance, this project would have seen this stage.

Abstract

Text Summarizer is a machine learning and web application project designed to convert a text into a summary. Text summarization is the problem of creating a short, accurate, and fluent summary of a longer text document.

Automatic text summarization methods are greatly needed to address the ever-growing amount of text data available online to both better help discover relevant information and to consume relevant information faster.

There is an enormous amount of textual material, and it is only growing every single day. here is a great need to reduce much of this text data to shorter, focused summaries that capture the salient details, both so we can navigate it more effectively as well as check whether the larger documents contain the information that we are looking for. We (humans) are generally good at this type of task as it involves first understanding the meaning of the source document and then distilling the meaning and capturing salient details in the new description.

As such, the goal of automatically creating summaries of text is to have the resulting summaries as good as those written by humans.

Table of Contents

Chapter 1: Introduction

1.1. What does Text Summarizer do?

This text summarizer is an online tool that wraps up a text to a specified short length. It condenses a long article to main points. The need for text summarizers are increasing day by day, because of time constraints.

People are looking for shortcut methods to learn ideas in lesser time. Even text summarizers are helping them to decide whether a book, a research paper, or an article is worth reading or not.

1.2. Approaches in auto summarization:

Mainly two approaches have been developed over time :

1. ***Extraction Summarization:***

This approach entails the method to extract keywords and phrases from sentences and then join them to produce a compact meaningful summary.

2. ***Abstractive Summarization:***

In this method, algorithms are developed in such a way to reproduce a long text into a shorter one by NLP. It retains its meaning but changes the structure of sentences.

Note: In this project, we are using Extractive Summarization.

1.3. How does this text summarizer work?

Trained by machine learning, our text summarizer uses the concept of abstractive summarization to summarize a book, an article, or a research paper.

It uses NLP to create novel sentences and generates a summary in which the main idea remains intact. IT is an advanced-level tool that uses AI for its work. Therefore, the summary produced by this tool appears to be flawless and inflow.

1.4. How to use our text summarizer?

Our summarizing tool is the best because it is simple to use and efficient also.

- Insert the text (article, url) into the text area.
- Click the “**Summarize**” Button.

1.5. Summary

Even a text summary can help you determine if a book, research paper, or article is worth reading. This approach involves extracting keywords and phrases 4484 from a sentence and putting them together in a meaningful and compact 4484 summary. In this method, an algorithm is developed to use NLP to convert long text to short text. Machine learning trained text summarizers use the concept of abstract summarization to summarize a book, article, or research paper. Use

NLP to create a new sentence and a summary that retains the main idea.

IT is an advanced tool that uses AI to do its job.

Therefore, the summary generated by this tool looks error-free and fluent.

Chapter 2: FRAMEWORKS AND PLATFORMS

2.1. FLASK

a). What is Flask?

Flask is a web application framework written in Python. It is developed by Armin Ronacher, who leads an international group of Python enthusiasts named Pocco. Flask is based on the Werkzeug WSGI toolkit and Jinja2 template engine. Both are Pocco projects.

b). WSGI

Web Server Gateway Interface (WSGI) has been adopted as a standard for Python web application development. WSGI is a specification for a universal interface between the web server and the web applications.

c). Install virtualenv for development environment

Virtualenv is a virtual Python environment builder. It helps a user to create multiple Python environments side-by-side. Thereby, it can avoid compatibility issues between the different versions of the libraries.

The following command installs virtualenv.

```
Python -m venv <virtual_environment_name>
```

Once installed, a new virtual environment is created in a folder.

```
mkdir <project_folder_name>
cd  <project_folder_name>
virtualenv  <virtual_environment_name>
```

To activate corresponding environment, On Windows, following can be used

```
Venv\scripts\activate
```

We are now ready to install Flask in this environment.

```
pip install Flask
```

In order to test Flask installation, type the following code in the editor as Hello.py

```
from flask import Flask
app = Flask(__name__)


@app.route('/')
def hello_world():
   return 'Hello World’
```

```
if __name__ == '__main__':
   app.run()
```

Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application.

Flask constructor takes the name of the current module (__name__) as argument.

The route() function of the Flask class is a decorator, which tells the application which URL should call the associated function.

```
app.route(rule, options)
```

- The rule parameter represents URL binding with the function.
- The options is a list of parameters to be forwarded to the underlying Rule object.

In the above example, '/' URL is bound with the hello_world() function. Hence, when the home page of a web server is opened in the browser, the output of this function will be rendered.

Finally the run() method of Flask class runs the application on the local development server.

```
Python <Flask_app_file _name)
```

A message in Python shell informs you that

```
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Open the above URL (localhost:5000) in the browser. ‘Hello World’ message will be displayed on it.

2.2. WEB SCRAPING

Web scraping is an automatic method to obtain large amounts of data from websites. Most of this data is unstructured data in an HTML format which is then converted into structured data in a spreadsheet or a database so that it can be used in various applications. There are many different ways to perform web scraping to obtain data from websites. These include using online services, particular API’s or even creating your code for web scraping from scratch. Many large websites, like Google, Twitter, Facebook, StackOverflow, etc. have API’s that allow you to access their data in a structured format. This is the best option, but there are other sites that don’t allow users to access large amounts of data in a structured form or they are simply not that technologically advanced. In that situation, it’s best to use Web Scraping to scrape the website for data.

a). How Web Scrapers Work?

Web Scrapers can extract all the data on particular sites or the specific data that a user wants. Ideally, it’s best if you specify the data you want so that the web scraper only extracts that data quickly. For example, you might want to scrape an Amazon page for the types of juicers available, but you might only want the data about the models of different juicers and not the customer reviews.

So, when a web scraper needs to scrape a site, first the URLs are provided. Then it loads all the HTML code for those sites and a more advanced scraper might even extract all the CSS and Javascript elements as well. Then the scraper obtains the required data from this HTML code and outputs this data in the format specified by the user. Mostly, this is in the form of an Excel spreadsheet or a CSV file, but the data can also be saved in other formats, such as a JSON file.

2.3. BEAUTIFUL SOUP

Beautiful Soup is a library that makes it easy to scrape information from web pages. It sits atop an HTML or XML parser, providing Pythonic idioms for iterating, searching, and modifying the parse tree.

There are mainly two ways to extract data from a website:

- Use the API of the website (if it exists). For example, Facebook has the Facebook Graph API which allows retrieval of data posted on Facebook.
- Access the HTML of the webpage and extract useful information/data from it. This technique is called web scraping or web harvesting or web data extraction.

Steps involved in web scraping:

1. Send an HTTP request to the URL of the webpage you want to access. The server responds to the request by returning the HTML content of the

webpage. For this task, we will use a third-party HTTP library for python-requests.

2. Once we have accessed the HTML content, we are left with the task of parsing the data. Since most of the HTML data is nested, we cannot extract data simply through string processing. One needs a parser which can create a nested/tree structure of the HTML data. There are many HTML parser libraries available but the most advanced one is html5lib.
3. Now, all we need to do is navigate and search the parse tree that we created, i.e. tree traversal. For this task, we will be using another third-party python library, Beautiful Soup. It is a Python library for pulling data out of HTML and XML files.

2.4. TEXTRANK

TextRank is an extractive and unsupervised text summarization technique.For the task of automated summarization,Textrank models any document as a graph using sentences as a nodes.A function to compute the similarity of sentences is needed to build edges in between.This function is used to weight the graph edges,the higher the similarity between sentences the more important the edge between them will be in the graph.TextRank determines the relation of similarity between two sentences based on the context that both share.This overlap is calculated simply as the number of common lexical tokens between them,divided by the length of each to avoid promoting the long sentences.

2.5. SPACY SUMMARIZER

a). What is spaCy?

SpaCy is a free, open-source advanced natural language processing library, written in the programming languages Python and Cython. spaCy is mainly used in the development of production software and also supports deep learning workflow via statistical models of PyTorch and TensorFlow.

b). Why spaCy?

spaCy provides a fast and accurate syntactic analysis, named entity recognition and ready access to word vectors. We can use the default word vectors or replace them with any you have. spaCy also offers tokenization, sentence boundary detection, POS tagging, syntactic parsing, integrated word vectors, and alignment into the original string with high accuracy.

Steps involved in spacy summarization are:-

Step: 1 Installation instructions

To install *spaCy*, simply type the following:

```
pip install -U spacy
```

To begin with import *spaCy* and other necessary modules:

```
import spacy
from spacy.lang.en.stop_words import STOP_WORDS
from string import punctuation
from collections import Counter
from heapq import nlargest
```

Next, load the model (English) into spaCy

```
nlp=spacy.load(‘en’)
```

The text we are about to handle is “*Introduction to Machine Learning”* and the string is stored in the variable *doc.*

And the string is,

```
Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to progressively improve their performance on a specific task. Machine learning algorithms build a mathematical model of sample data, known as “training data”, in order to make predictions or decisions without being explicitly programmed to perform the task. Machine learning algorithms are used in the applications of email filtering, detection of network intruders, and computer vision, where it is infeasible to develop an algorithm of specific instructions for performing the task. Machine learning is closely related to computational statistics, which focuses on making predictions using computers. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a field of study within machine learning and focuses on exploratory data analysis through unsupervised learning. In its application across business problems, machine learning is also referred to as predictive analytics.
```

Now, pass the string *doc* into the *nlp* function.

```
doc=nlp(doc)
```

To find the number of sentences in the given string the following function is used,

```
len(list(doc.sents))
```

Next, two lists are created for parts-of-speech and stop words to validate each token followed by filtering of the necessary tokens and saving them in the *keywords* list.

Step: 2 Filtering tokens

```
keyword=[]
stopwords=list(STOP_WORDS)
pos_tag=[‘PROPN’,’ADJ’,’NOUN’,’VERB’]
for token in doc:
     if(token.text in stopwords or token.text in punctuation):
          continue
    if(token.pos_  in pos_tag):
        keyword.append(token.text)
```

Calculate the frequency of each token using the “*Counter”* function, store it in *freq_word* and to view top 5 frequent words, *the most_common* method can be used.

```
freq_word=Counter(keyword)
freq_word.most_common(5)
```

The desired output would be,

```
[('learning', 8), ('Machine', 4), ('study', 3), ('algorithms', 3), ('task', 3)]
```

This frequency can be normalised for better processing and it can be done by dividing the token's frequencies by the maximum frequency.

Step: 3 Normalization

```
max_freq=Counter(keyword).most_common(1)[0][1]
for word in freq_word.keys():
    freq_word[word]=(freq_word[word]/max_freq)
freq_word.most_common(5)
```

The normalised list is,

```
[('learning', 1.0), ('Machine', 0.5), ('study', 0.375), ('algorithms', 0.375), ('task', 0.375)]
```

This is the major part where each sentence is weighed based on the frequency of the token present in each sentence. The result is stored as a key-value pair in *sent_strength* where keys are the sentences in the string *doc* and the values are the weight of each sentence.

Step: 4 Weighing sentences

```
sent_strength={}
   for sent in doc.sents:
        for word in sent:
             if word.text in freq_word.keys():
                  if sent in sent_strength.keys():
                       sent_stren
```

And the output is,

```
{Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to progressively improve their performance on a specific task.: 4.125,
Machine learning algorithms build a mathematical model of sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task.: 4.625,
Machine learning algorithms are used in the applications of email filtering, detection of
network intruders, and computer vision, where it is infeasible to develop an algorithm of specific instructions for performing the task.: 4.25,
Machine learning is closely related to computational statistics, which focuses on making predictions using computers.: 2.625,
The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning.: 3.125,
Data mining is a field of study within machine learning, and focuses on exploratory data analysis through unsupervised learning.: 4.25, In its application across business problems, machine learning is also referred to as predictive analytics.: 2.25}
```

Finally, *nlargest* function is used to summarize the string, it takes 3 arguments, Number of data to extract,An Iterable (List/Tuple/Dictionary), Condition to be satisfied, respectively.

Step: 5 Summarizing the string

```
summarized_sentences=nlargest(3,sent_strength,key=sent_strength.get)
print(summarized_sentences)
```

And the *nlargest* function returns a list containing the top 3 sentences which are stored as *summarized_sentences*.

```
[Machine learning algorithms build a mathematical model of sample data, known as
“training data”, in order to make predictions or decisions without being explicitly programmed to perform the task., Machine learning algorithms are used in the applications of email filtering, detection of network intruders, and computer vision, where it is infeasible to develop an algorithm of specific instructions for performing the task., Data mining is a field of study within machine learning, and focuses on exploratory data analysis through unsupervised learning.]
```

Each sentence in this list is of *spacy.span* type

```
print(type(summarized_sentences[0]))
```

This can be converted to a string by the following lines of code,

```
final_sentences=[w.text for w in summarized_sentences]
summary=’ ‘.join(final_sentences)
print(summary)
```

Resulting in a final summarized output as

```
Machine learning algorithms build a mathematical model of sample data, known as “training data”, in order to make predictions or decisions without being explicitly programmed to perform the task. Machine learning algorithms are used in the applications of email filtering, detection of network intruders, and computer vision, where it is infeasible to develop an algorithm of specific instructions for performing the task. Data mining is a field of study within machine learning, and focuses on
```

exploratory data analysis through unsupervised learning.

2.6. GENSIM

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.

a). Features

- All algorithms are **memory-independent** w.r.t. the corpus size (can process input larger than RAM, streamed, out-of-core)
- **Intuitive interfaces**
  - easy to plug in your own input corpus/datastream (simple streaming API)
  - easy to extend with other Vector Space algorithms (simple transformation API)

- Efficient multicore implementations of popular algorithms, such as online **Latent Semantic Analysis (LSA/LSI/SVD)**, **Latent Dirichlet Allocation (LDA)**, **Random Projections (RP)**, **Hierarchical Dirichlet Process (HDP)** or **word2vec deep learning**.
- **Distributed computing**: can run *Latent Semantic Analysis* and *Latent Dirichlet Allocation* on a cluster of computers.

b). How come gensim is so fast and memory efficient?

Many scientific algorithms can be expressed in terms of large matrix operations (see the BLAS note above). Gensim taps into these low-level BLAS libraries, by means of its dependency on NumPy. So while gensim-the-top-level-code is pure Python, it actually executes highly optimized Fortran/C under the hood, including multithreading (if your BLAS is so configured).

2.7. NLTK

Natural language processing (NLP) is a field that focuses on making natural human language usable by computer programs. NLTK, or Natural Language Toolkit, is a Python package that you can use for NLP.NLTK 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.
Libraries used in project are:-

a). nltk.tokenize: By tokenizing, you can conveniently split up text by word or by sentence. This will allow you to work with smaller pieces of text that are still relatively coherent and meaningful even outside of the context of the rest of the text. It’s your first step in turning unstructured data into structured data, which is easier to analyze.

When you’re analyzing text, you’ll be tokenizing by word and tokenizing by sentence. Here’s what both types of tokenization bring to the table:

- **Tokenizing by word**: Tokenizing your text by word allows you to identify words that come up particularly often. For example, if you were analyzing a group of job ads, then you might find that the word “Python” comes up often. That could suggest high demand for Python knowledge, but you’d need to look deeper to know more.
- **Tokenizing by sentence**: When you tokenize by sentence, you can analyze how those words relate to one another and see more context. Are there a lot of negative words around the word “Python” because the hiring manager doesn’t like Python? Are there more terms from the domain of herpetology than the domain of software development, suggesting that you may be dealing with an entirely different kind of python than you were expecting?

Here’s how to import the relevant parts of NLTK so you can tokenize by word and by sentence:

```
>>> from nltk.tokenize import sent_tokenize, word_tokenize
```

Now that you’ve imported what you need, you can create a string to tokenize. Here’s a quote from *Dune* that you can use:

```
>>> example_string = """
... Muad'Dib learned rapidly because his first training was in how to learn.
... And the first lesson of all was the basic trust that he could learn.
... It's shocking to find how many people do not believe they can learn,
... and how many more believe learning to be difficult."""
```

You can use **sent_tokenize()** to split up **example_string** into sentences:

```
>>> sent_tokenize(example_string)
["Muad'Dib learned rapidly because his first training was in how to learn.",
'And the first lesson of all was the basic trust that he could learn.',
"It's shocking to find how many people do not believe they can learn, and how many
more believe learning to be difficult."]
```

Tokenizing **example_string** by sentence gives you a list of three strings that are sentences:

1. "Muad'Dib learned rapidly because his first training was in how to learn."
2. 'And the first lesson of all was the basic trust that he could learn.'
3. "It's shocking to find how many people do not believe they can learn, and how many more believe learning to be difficult."

Now try tokenizing **example_string** by word:

```
>>> word_tokenize(example_string)
["Muad'Dib",
 'learned',
 'rapidly',
 'because',
 'his',
 'first',
 'training',
 'was',
 'in',
 'how',
 'to',
 'learn',
 '.',
 ['And','the','first','lesson','of','all','was','the','basic','trust',
 'that','he','could','learn','.','It',"'s", 'shocking', 'to',
  'find','how','many','people','do','not','believe','they','can',
 'learn',',','and','how','many','more','believe','learning','to',
 'be','difficult','.']
```

b). nltk.corpus:-_Stopwords are words that you want to ignore, so you filter them out of your text when you're processing it. Very common words like 'in', 'is', and 'an' are often used as stop words since they don't add a lot of meaning to a text in and of themselves.

Here's how to import the relevant parts of NLTK in order to filter out stop words:

```
>>> nltk.download("stopwords")
>>> from nltk.corpus import stopwords
>>> from nltk.tokenize import word_tokenize
```

2.8. SUMY

Extractive text summarization techniques perform summarization by picking portions of texts and constructing a summary.Sumy summarizer does extractive text summarization.Techniques used by sumy summarizer are:-

1. Lex Rank
2. Luhn
3. LSA
4. Text Rank

We are using Lex Rank for summarization in our project.

LexRank is an unsupervised approach to text summarization based on graph-based centrality scoring of sentences. The main idea is that sentences "recommend" other similar sentences to the reader. Thus, if one sentence is very similar to many others, it will likely be a sentence of great importance. The importance of this sentence also stems from the importance of the sentences "recommending" it. Thus, to get ranked highly and placed in a summary, a sentence must be similar to many sentences that are in turn also similar to many other sentences. This makes intuitive sense and allows the algorithms to be applied to any arbitrary new text.

Chapter 3: Module Description

In this project, we are providing two ways in which the user can upload the text needed to summarize, i.e.; by inserting the text itself or by pasting the URL of the website.

3.1 Home Page

The Home Page consists of various components and elements such as navbar, footer, text area, buttons etc. The navbar is the first component of the home page which consists of a Logo and two nav-links, Home and Compare summarizers.

A refresh button will appear below the heading **"Summarization simplified".** By clicking on this button, all the already present text will disappear from the page and the user will be able to enter the text they want to summarize. Now, the user can enter either text or any URL they wish to summarize in the respective text area accordingly. Thus, by clicking on the summarize button, the user will get the summary of the respective text.

There are two ways by which, the user can obtain the summary and they are as follows:

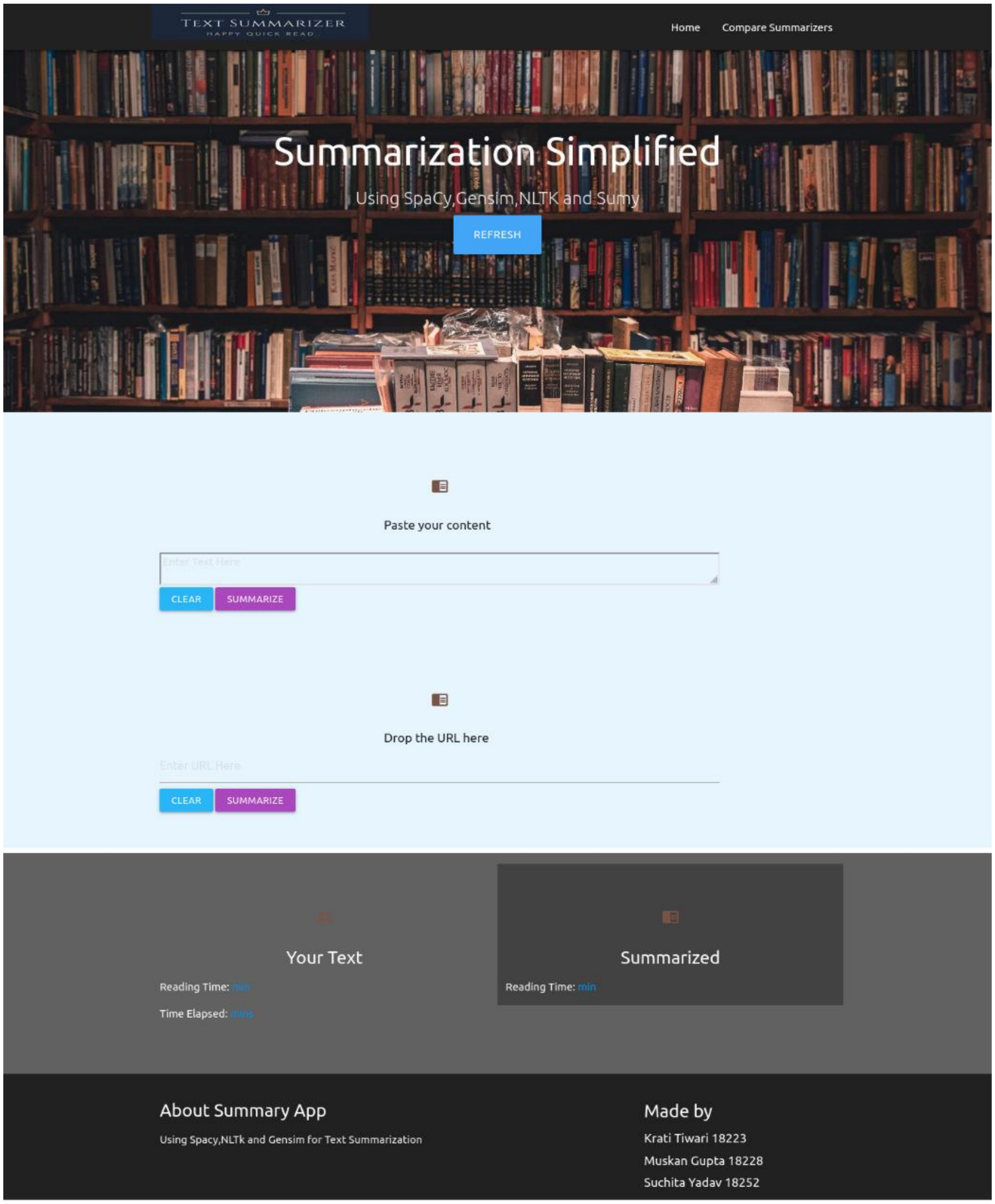
a). Text Summarizer

In this summarization, we will either copy or type the text by ourselves in the text area shown with the heading "**Paste your content**". Then, by clicking on summarize, we will get the summary of the respective text pasted/typed in the textbox along with the reading time in minutes**.**

b). URL Summarizer

In this summarization, we will either copy the URL in the text area shown with the heading "**Drop the URL here**". Then, by clicking on summarize, we will get the summary of the respective website in the textbox along with the reading time in minutes.

IMG: Home Page



About Summary App

Using Spacy,NLTk and Gensim for Text Summarization

Made by

Krati Tiwari 18223

Muskan Gupta 18228

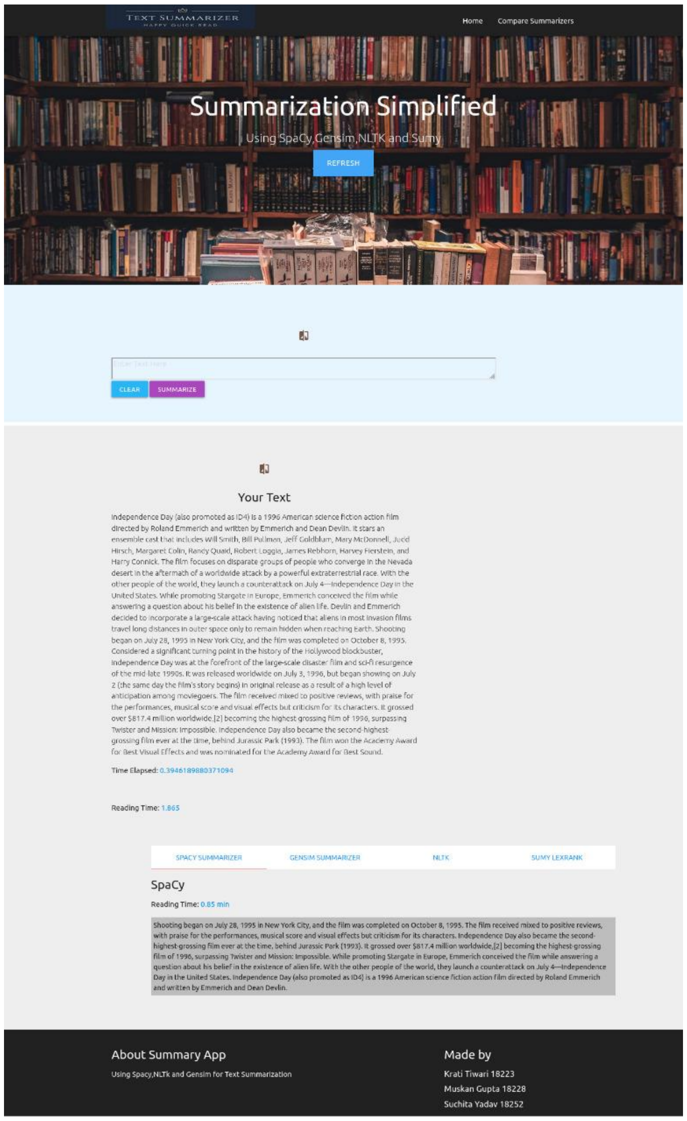
Suchita Yadav 18252

IMG: Home Page

3.2. Compare Summarizer

This is another page which can be reached by clicking "**Compare summarizers"** nav-link present on the navbar. On this page, there are four algorithms given with the help of which we can summarize our text. Each and every algorithm has a unique way of summarizing the text. The four algorithms are:

- Spacy summarizer
- Gensim summarizer
- NLTK
- Sumy Lexrank



Your Text

Independence Day (also promoted as ID4) is a 1996 American science fiction action film directed by Roland Emmerich and written by Emmerich and Dean Devlin. It stars an ensemble cast that includes Will Smith, Bill Pullman, Jeff Goldblum, Mary McDonnell, Judd Hirsch, Margaret Colin, Randy Quaid, Robert Loggia, James Rebhorn, Harvey Fierstein, and Harry Connick. The film focuses on disparate groups of people who converge in the Nevada desert in the aftermath of a worldwide attack by a powerful extraterrestrial race. With the other people of the world, they launch a counterattack on July 4—Independence Day in the United States. While promoting Stargate in Europe, Emmerich conceived the film while answering a question about his belief in the existence of alien life. Devlin and Emmerich decided to incorporate a large-scale attack having noticed that aliens in most invasion films travel long distances in outer space only to remain hidden when reaching Earth. Shooting began on July 28, 1995 in New York City, and the film was completed on October 8, 1995. Considered a significant turning point in the history of the Hollywood blockbuster, Independence Day was at the forefront of the large-scale disaster film and sci-fi resurgence of the mid-late 1990s. It was released worldwide on July 3, 1996, but began showing on July 2 (the same day the film's story begins) in original release as a result of a high level of anticipation among moviegoers. The film received mixed to positive reviews, with praise for the performances, musical score and visual effects but criticism for its characters. It grossed over $817.4 million worldwide.[2] becoming the highest grossing film of 1996, surpassing Twister and Mission: Impossible. Independence Day also became the second-highest-grossing film ever at the time, behind Jurassic Park (1993). The film won the Academy Award for Best Visual Effects and was nominated for the Academy Award for Best Sound.

Time Elapsed: 0.3946189880371094

Reading Time: 1.865

SpaCy

Reading Time: 0.85 min

Shooting began on July 28, 1995 in New York City, and the film was completed on October 8, 1995. The film received mixed to positive reviews, with praise for the performances, musical score and visual effects but criticism for its characters. Independence Day also became the second-highest-grossing film ever at the time, behind Jurassic Park (1993). It grossed over $817.4 million worldwide,[2] becoming the highest-grossing film of 1996, surpassing Twister and Mission: Impossible. While promoting Stargate in Europe, Emmerich conceived the film while answering a question about his belief in the existence of alien life. With the other people of the world, they launch a counterattack on July 4—Independence Day in the United States. Independence Day (also promoted as ID4) is a 1996 American science fiction action film directed by Roland Emmerich and written by Emmerich and Dean Devlin.

About Summary App

Using Spacy,NLTk and Gensim for Text Summarization

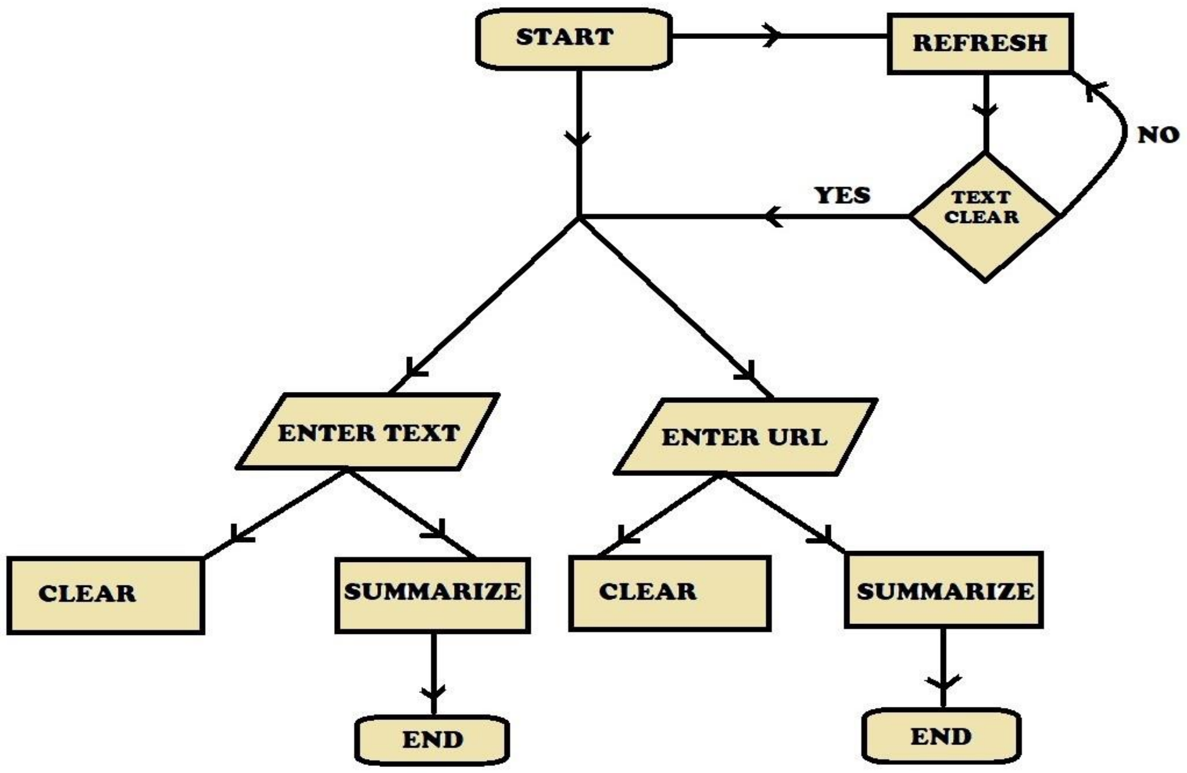
Made by

Krati Tiwari 18223

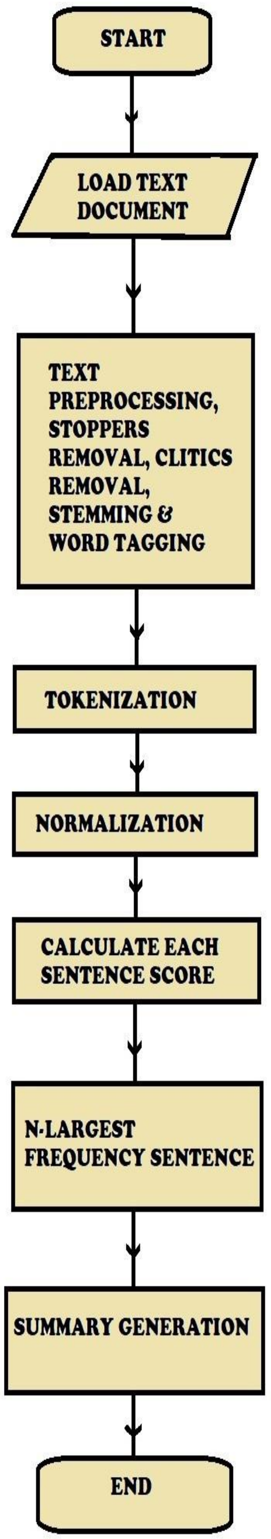
Muskan Gupta 18228

Suchita Yadav 18252

3.3. Architecture



IMG: Architecture as per user’s perspective


START
LOAD TEXT DOCUMENT
TEXT PREPROCESSING, STOPPERS REMOVAL, CLITICS REMOVAL, STEMMING & WORD TAGGING
TOKENIZATION
NORMALIZATION
CALCULATE EACH SENTENCE SCORE
N-LARGEST FREQUENCY SENTENCE
SUMMARY GENERATION
END

IMG: Workflow of Text Summarizer

3.4. Code Snippets

a). Text Summarization using NLTK

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
import heapq

def nltk_summarizer(raw_text):
        stopWords = set(stopwords.words("english"))
        word_frequencies = {}
        for word in nltk.word_tokenize(raw_text):
            if word not in stopWords:
                if word not in word_frequencies.keys():
                    word_frequencies[word] = 1
                else:
                    word_frequencies[word] += 1

        maximum_frequncy = max(word_frequencies.values())

        for word in word_frequencies.keys():
            word_frequencies[word] = (word_frequencies[word]/maximum_frequncy)

        sentence_list = nltk.sent_tokenize(raw_text)
        sentence_scores = {}
        for sent in sentence_list:
            for word in nltk.word_tokenize(sent.lower()):
                if word in word_frequencies.keys():
                    if len(sent.split(' ')) < 30:
```

```
                    if sent not in sentence_scores.keys():
                        sentence_scores[sent] = word_frequencies[word]
                    else:
                        sentence_scores[sent] += word_frequencies[word]



        summary_sentences = heapq.nlargest(7, sentence_scores,
key=sentence_scores.get)

        summary = ' '.join(summary_sentences)
        return summary
```

b). Text Summarization using spacy

```
import spacy
nlp = spacy.load('en')
# Pkgs for Normalizing Text
from spacy.lang.en.stop_words import STOP_WORDS
from string import punctuation
# Import Heapq for Finding the Top N Sentences
from heapq import nlargest


def text_summarizer(raw_docx):
    raw_text = raw_docx
    docx = nlp(raw_text)
    stopwords = list(STOP_WORDS)
    # Build Word Frequency # word.text is tokenization in spacy
    word_frequencies = {}
```

```
for word in docx:
    if word.text not in stopwords:
        if word.text not in word_frequencies.keys():
            word_frequencies[word.text] = 1
        else:
            word_frequencies[word.text] += 1


maximum_frequncy = max(word_frequencies.values())

for word in word_frequencies.keys():
    word_frequencies[word] = (word_frequencies[word]/maximum_frequncy)
# Sentence Tokens
sentence_list = [ sentence for sentence in docx.sents ]

# Sentence Scores
sentence_scores = {}
for sent in sentence_list:
    for word in sent:
        if word.text.lower() in word_frequencies.keys():
            if len(sent.text.split(' ')) < 30:
                if sent not in sentence_scores.keys():
                    sentence_scores[sent] = word_frequencies[word.text.lower()]
                else:
                    sentence_scores[sent] += word_frequencies[word.text.lower()]

summarized_sentences = nlargest(7, sentence_scores, key=sentence_scores.get)
final_sentences = [ w.text for w in summarized_sentences ]
summary = ' '.join(final_sentences)
return summary
```

Chapter 4: Result and analysis

For evaluation purpose we are using two measures namely- Bleu and Rouge

4.1. BLEU (Bilingual Evaluation Understudy)

Measures precision: how much the words (and/or n-grams) in the *machine generated summaries* appeared in the human reference summaries.

```
from nltk.translate.bleu_score import sentence_bleu
score = sentence_bleu(  reference, test)
print(score)
```

Bleu Score: 0.83

4.2. ROUGE (Recall Oriented Understudy for Gisting Evaluation)

Measures recall: how much the words (and/or n-grams) in the *human reference summaries* appeared in the machine generated summaries.
Naturally - these results are complementing, as is often the case in precision vs recall. If you have many words from the system results appearing in the human references you will have high Bleu, and if you have many words from the human references appearing in the system results you will have high Rouge.

```
from rouge_score import rouge_scorer
scorer = rouge_scorer.RougeScorer(['rouge1'])
scorer.score(reference, test)
```

```
scores.append(scorer.score(reference, test))
```

Rogue Score:

```
{'fmeasure': [0.8000000000000002, 0.22222222222222224],
 'precision': [0.8, 0.2],
 'recall': [0.8, 0.25]}
```

Chapter 5: Conclusion and Future scope

5.1. Conclusion

As we can see that the Bleu and Rogue score of our model is more appropriate than the existing models, we can conclude that this model outstands amidst the current models available.

Automatic text summarization methods are greatly needed to address the ever-growing amount of text data available online to both better help discover relevant information and to consume relevant information faster.

5.2. Future Scope

This project will not remain limited to text and URL summarization only.

- In future, we will also add a feature to extract text from an image and summarize it.
- Facility of uploading PDFs and extracting required summary out of it will also be provided.
- Language in which the summary is being obtained will not be limited to English. We will add the feature of converting the text into a summary in as many languages as possible to make this application useful for nearly every population.

Chapter 6: References

a) https://ieeexplore.ieee.org/document/7508049

b) https://ieeexplore.ieee.org/document/9358703

c) https://www.semanticscholar.org/paper/A-Review-Paper-on-Text-Summarization-Gaikwad-Mahender/a075b59cb7fda9cd62ab5946a8cdaba94af430d7

d) https://medium.com/luisfredgs/automatic-text-summarization-with-machine-learning-an-overview-68ded5717a25

e) https://blog.floydhub.com/gentle-introduction-to-text-summarization-in-machine-learning/