Major Project II

Analysis & Comparison of Text Summarization Approaches

Project Report

*s*ubmitted in partial fulfillment of the

*r*equirements for the award of

BACHELOR OF TECHNOLOGY

Submitted by

Aksh Sharma - 195019
Khushwant Kaswan - 195030
Khem Singh - 195038
Priyanshu Dhiman - 195049
Anupam Kumar - 195050
Astha Dad - 195076
Ravi Kant - 195115

Supervised by

Dr. Jyoti Srivastava

DEPARTMENT OF COMPUTER SCIENCE
NIT HAMIRPUR
INDIA

Analysis & Comparison
of
Text Summarization Approaches

Declaration

We hereby declare that our work submitted in the partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** in the Department of Computer Science and Engineering of the National Institute of Technology Hamirpur, titled as **"Analysis & Comparison of Text Summarization Approaches"** is an record of our work carried out during the **VIII Semester** under the guidance of **Dr. Jyoti Srivastava** , Assistant Professor , DoCSE.

The matter presented in this report has not been submitted by us for the award of any other degree of this or any other Institute/University.

Aksh Sharma - 195019
Khushwant Kaswan - 195030
Khem Singh - 195038
Priyanshu Dhiman - 195049
Anupam Kumar - 195050
Astha Dad - 195076
Ravi Kant - 195115

This is to certify that the above statement made by the candidates is true to the best of my knowledge and belief.

Dr. Jyoti Srivastava

Date: May 9, 2023

Acknowledgement

We would like to start by expressing our sincerest gratitude to our Project supervisor **Dr. Jyoti Srivastava**, Assistant Professor, Department of Computer Science at the National Institute of Technology Hamirpur, for her expertise, guidance, and enthusiastic involvement during our coursework.

We are highly obliged to faculty members of the Computer Science and Engineering Department because without their valuable insights and constructive opinions during evaluations, our project would not have yielded the significant results and led us to explore a myriad of use cases that we have put forward in this report.

We express our special thanks to our parents for their encouragement, constant moral Support, and to our friends and colleagues for being inquisitive and supportive during the course of this project.

Aksh Sharma - 195019
Khushwant Kaswan - 195030
Khem Singh - 195038
Priyanshu Dhiman - 195049
Anupam Kumar - 195050
Astha Dad - 195076
Ravi Kant - 195115

Abstract

This project report focuses on analyzing and comparing different approaches for text summarization. As the volume of digital data has increased rapidly, summarization techniques have grown more crucial to extract relevant information from large volumes of textual digital data . The extraction-based and abstraction-based approaches to text summarization are both covered in this report. In terms of ROUGE measures, we are evaluating how well these summarising approaches will perform.˙
[0.1in] In view of the enormous amount of data on the internet, a tool that can summarize texts is valuable for condensing digital documents such as government data, medical reports, news articles, and research papers. The volume of text available has exceeded the capacity of an individual user to read, and a simple web search yields thousands of related pages. Although search engines have made recent improvements to rank documents more accurately, the documents themselves are often too long to read in a short amount of time. Therefore, having summarization tools for this purpose is incredibly crucial.

Additionally, the project report describes the implementation of several extractive and some abstractive approaches of text summarization on the DeepMind CNN Stories Dataset. We provide a detailed description of the dataset and the steps taken to create a corpus from the dataset that is ready for summary evaluation purpose. We have evaluated the performance of these algorithms using ROUGE metrics along with a comparative analysis of the results. Finally, the limitations and prospective future directions of our text summarization project have been discussed.

This project's contributions are twofold.:

1. On the DeepMind CNN Stories Dataset, we assessed and compared the efficacy of various text summarization algorithms.

2. We have created a GUI for summarizing user's text.

Keywords

Summary, Summarization, Extractive, Abstractive, CNN, ROUGE

Contents

List of Figures

List of Tables

Introduction

1.1 Motivation

Text Summarization is an active area of study within the Information Retrieval and Natural Language Processing communities. Text Summarization is increasingly being used in the commercial sector such as Telephone communication industry, data mining of text databases, for web-based information retrieval, in word Processing tools. Many approaches differ on the behaviour of their problem formulations. Automatic text summarization is an important step for information management tasks. It solves the problem of selecting the most important portions of the text. High quality summarization requires sophisticated NLP techniques.

1.2 Importance and Applications

Propelled by the modern technological innovations, data is to this century what oil was to the previous one. Today, our world is parachuted by the gathering and dissemination of huge amounts of data.
In fact, the International Data Corporation (IDC) projects that the total amount of digital data circulating annually around the world would sprout from 4.4 zettabytes in 2013 to hit 180 zettabytes in 2025. Thats a lot of data!
With such a big amount of data circulating in the digital space, there is need to develop machine learning algorithms that can automatically shorten longer texts and deliver accurate summaries that can fluently pass the intended messages.

- Summarizing reduces the time it takes to read.
- Summaries facilitate the process of selecting documents during research.
- Automatic summarization enhances the efficiency of indexing.
- Summarization algorithms generated by machines are less prejudiced compared to human summarizers.
- Personalized summaries are beneficial in question-answering systems since they provide customized information.
- The use of automatic or semi-automatic summarization systems allows commercial abstract services to handle a larger volume of texts.

Furthermore, applying text summarization reduces reading time, accelerates the process of researching for information, and increases the amount of information that can fit in an area.

1.3 Terminology

1.3.1 Extractive Text Summarization

The extractive text summarization technique takes the text, ranks all the sentences according to the understanding and relevance of the text, and presents you with the most important sentences.

This method does not create new words or phrases, it just takes the already existing words and phrases and presents only that.

Here is an example:
Source text: **Joseph and Mary** rode on a donkey to **attend** the annual **event** in **Jerusalem**. In the city, **Mary** gave **birth** to a child named **Jesus**.

Extractive summary: Joseph and Mary attend event Jerusalem. Mary birth Jesus.

As you can see above, the words in bold have been extracted and joined to create a summary although sometimes the summary can be grammatically strange.

1.3.2 Abstractive Text Summarization

The abstraction technique entails paraphrasing and shortening parts of the source document. When abstraction is applied for text summarization in deep learning problems, it can overcome the grammar inconsistencies of the extractive method.

The abstractive text summarization algorithms create new phrases and sentences that relay the most useful information from the original text just like humans do.

Therefore, abstraction performs better than extraction. However, the text summarization algorithms required to do abstraction are more difficult to develop; thats why the use of extraction is still popular.

For the above example:
Abstractive summary: Joseph and Mary came to Jerusalem where Jesus was born.

1.3.3 Example

Original text:
"According to a recent study, dogs are better than cats at detecting certain scents. The study found that dogs were able to detect the smell of a particular chemical compound at a concentration of one part per trillion, whereas cats could only detect the same compound at a concentration of one part per million. This ability is thought to be related to dogs' highly developed sense of smell, which is around 1,000 times more sensitive than that of humans. The researchers believe that these findings could have practical applications in areas such as search and rescue, where dogs could be trained to locate people trapped under rubble or in other dangerous situations."

Extractive summary:
Dogs are better than cats at detecting certain scents. Dogs were able to detect a chemical compound at a concentration of one part per trillion, whereas cats could only detect the same compound at a concentration of one part per million. This ability is thought to be related to dogs' highly developed sense of smell. The researchers believe that these findings could have practical applications in areas such as search and rescue.

Abstractive summary:
A recent study has found that dogs are much more proficient at detecting scents than cats, with the former being able to detect a particular chemical compound at a concentration of one part per trillion, compared to the latter's detection at one part per million. The researchers believe that this superior ability is linked to dogs' highly sensitive sense of smell, which is believed to be about 1,000 times more powerful than that of humans. They also suggest that this finding could have significant implications in various areas, such as search and rescue operations where dogs could be trained to locate people in perilousăcircumstances.

Dataset

Summarization poses a challenge in terms of evaluation as it necessitates a database containing text and their corresponding summaries that have been written or curated by humans. These summaries, often referred to as **"gold summaries"**, are challenging to obtain in real-world scenarios, and hence, research in this field tends to concentrate on news articles and scientific papers where they are most easily accessible.

The standard method for evaluating summarization performance is ROUGE, which is derived from the metric used in translation evaluation. ROUGE measures the overlap of n-grams between the words in the predicted summary and the gold standard summary.

2.1 CNN/Daily Corpus

The CNN news stories dataset contains 92,579 news articles and their respective summaries termed as highlights in form of seperate paragraphs with @highlight tag.
The dataset was made available by from New York University and can be found here.

`https://cs.nyu.edu/~kcho/DMQA/`

2.1.1 State of Art Summarization Scores

Benchmarks of acadamic studies using the CNN dataset			
Study	**Date**	**Authors**	**F1 score**
Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond	Aug/2016	Ramesh Nallapati	35.5
Get To The Point: Summarization with Pointer-Generator Networks	Apr/2017	Abigail See	39 -40
A Hierarchical Structured Self-Attentive Model for Extractive Document Summarization (HSSAS)	Apr/2018	Kamal Al-Sabahi	42
Fine-tune BERT for Extractive Summarization	Mar/2019	Yang Liu	43

Table 2.1: Benchmarks of acadamic studies using the CNN dataset

	article	summary
82242	FIFA president Sepp Blatter praised South Kore...	Sepp Blatter said Korea's bid to host 2022 Wor...
36828	A Massachusetts teen accused of raping a fello...	Galileo Mondol faces charges that include aggr...
79969	Crippled by the brutal wind and waves of Hurri...	HMS Bounty and its captain, Robin Walbridge, w...
85566	Tatyana Fazlalizadeh had never heard of the te...	Street art campaign "Stop Telling Women to Smi...
50553	In the years before penicillin came into wide ...	Atul Gawande: Medicine developed around the id...

Figure 2.1: Raw Corpus

	count	mean	std	min	25%	50%	75%	max
article_char_count	91590.0	3917.104040	2040.346205	2.0	2302.0	3613.0	5198.0	11722.0
article_word_count	91590.0	659.652145	345.993464	0.0	385.0	608.0	877.0	1899.0
article_sentence_count	91590.0	34.261404	20.379003	0.0	19.0	30.0	45.0	356.0
summary_char_count	91590.0	264.589082	57.644282	52.0	223.0	268.0	310.0	610.0
summary_word_count	91590.0	42.570990	9.795157	7.0	36.0	43.0	50.0	106.0
summary_sentence_count	91590.0	3.627285	0.648160	1.0	3.0	4.0	4.0	10.0

Figure 2.2: Raw Dataset Statistics

Methodology

3.1 Corpus Creation

Our dataset CNN Daily News contains **92,579 .story files**.

Each story contains the string (CNN) and new line escape sequence as the initial part. For creating summaries, we found the all brief highlights using the @highlight tag. We merged all the @highlights tags to create a summary for the whole article.

Example story present in the Dataset :

```
(CNN) -- The 54 men and 14 boys rescued after being found chained this week
at an Islamic religious school in Pakistan have been reunited with their
families or placed in shelters, authorities said.The group was discovered
in an underground room with heavy chains linking them together.
The school, Al-Arabiya Aloom Jamia Masjid Zikirya, which also was a drug
rehab clinic, is in Sohrab Goth, a suburb of Gadap in Karachi.
All 14 boys were returned to their families, senior police official
Ahsanullah Marwat told CNN.
Of the adults, 47 had been released to their families, and seven were handed
over to a shelter for the homeless, he said.
Three people who worked at the facility were arrested, but the four men who
ran the place were still at large, Marwat said.
Officials said the facility was part madrassa and part drug-rehab facility,
and the captives were chained at night apparently to prevent their escape.

@highlight
Captive boys and men were rescued from an Islamic religious school in Pakistan
@highlight
They were reunited with their families this week
@highlight
The facility was a school and drug rehab clinic
@highlight
Authorities say they're searching for the owners; three others arrested at
the facility
```

We parsed the dataset to create stories and their summaries in separated individual files.

Pseudo Code to parse the dataset to stories and their summaries

```
Function read_raw_data(text):
    Set story_end to the index of '@highlight' in the text string

    # Extract the story
```

```
    Set story to the portion of the text string before the '@highlight' marker
    Set index to the index of '(CNN) -- ' in the story string
    If the index is greater than -1:
        Remove the text before '(CNN) -- ' from the story string
    Replace double newlines with single newlines in the story string

    # Extract the highlights

    Split the text string from the '@highlight' marker into a list of strings
                                            using '@highlight' as the delimiter
    Remove any empty strings from the list of highlights
    Remove any leading or trailing whitespace and newlines from each highlight
    Remove any empty strings that may result from whitespace removal
    Append a period and newline character to each highlight string
    Concatenate the highlights into a single string to form the summary

    Return the story and summary strings as a tuple
End Function
```

The seperated stories and their summaries are then used to create a Pandas dataframe which is treated as our raw corpus.

	article	summary
82242	FIFA president Sepp Blatter praised South Kore...	Sepp Blatter said Korea's bid to host 2022 Wor...
36828	A Massachusetts teen accused of raping a fello...	Galileo Mondol faces charges that include aggr...
79969	Crippled by the brutal wind and waves of Hurri...	HMS Bounty and its captain, Robin Walbridge, w...
85566	Tatyana Fazlalizadeh had never heard of the te...	Street art campaign "Stop Telling Women to Smi...
50553	In the years before penicillin came into wide ...	Atul Gawande: Medicine developed around the id...

Figure 3.1: Raw Corpus

3.1.1 Data Cleaning

The analysis of stories showed a pattern that helped us to clean the corpus. Articles with 0 to 2 sentences are problematic and their is also a pattern of conversations of interviews.

```
King: Ever want to do Broadway?
Carrey: Sure, sure. I'll do that.
King: Do a play?
Carrey: I would love to do it. I hope I could do it.
King: How did you and [girlfriend Jenny McCarthy] -- how did that happen?
```

TO REMOVE

- Zero-sentence articles are empty even though most of them have an associated summary in the corpus.

- One or two sentence articles are either referring to another document or is a long list of of nominees for the award shows.
- Other patterns that do not contribute much to the article and can be cleaned from the corpus, such as: (EW.com), CLICK HERE, (CNN), NEW:, 'All rights reserved'.
- In order to clean the interviews ,we implemented regex search to find words starting with a capital letter and finishing with ':' and removing all strings with the word 'interview'.

```
to_drop_len = corpus.loc[corpus['article_sentence_count'] <= 2, 'article'].index

to_drop_interv= corpus.loc[(corpus['article'].str.count('[A-Z]\w*:') > 5) &
             (corpus['article'].str.contains('intervi', case = False)) ].index

patterns_to_remove = ['\(\w+\.com\)', 'CLICK HERE.*', '\(CNN\)', 'NEW\:',
                      'All righs reserved\.']
```

3.1.2 Exploratory Data Analysis

Below are the examples of analysis of the distributions between articles and summaries.

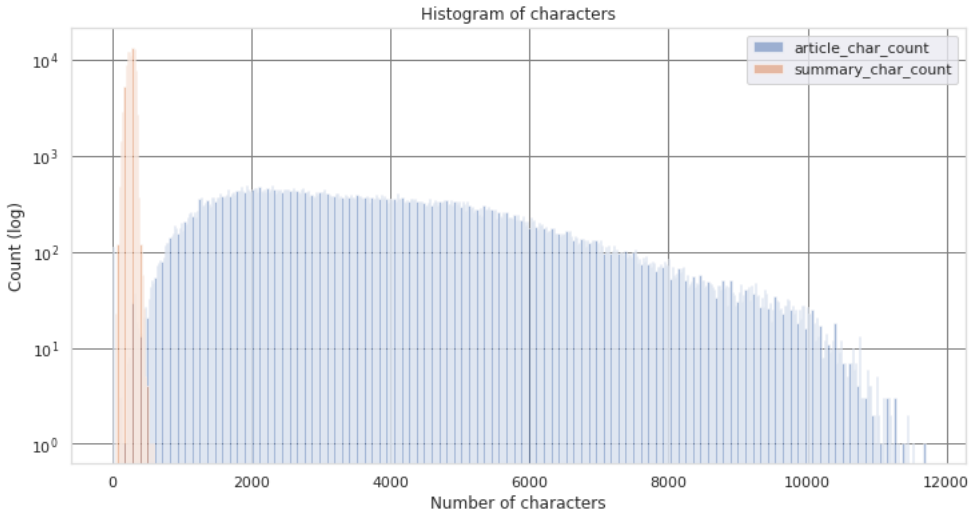

Figure 3.2: Histogram of Characters

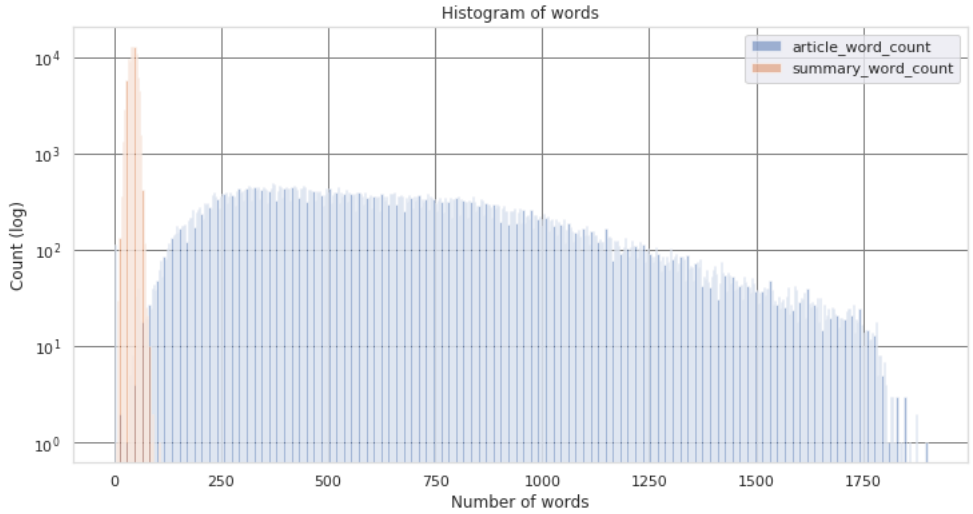

Figure 3.3: Histogram of Words

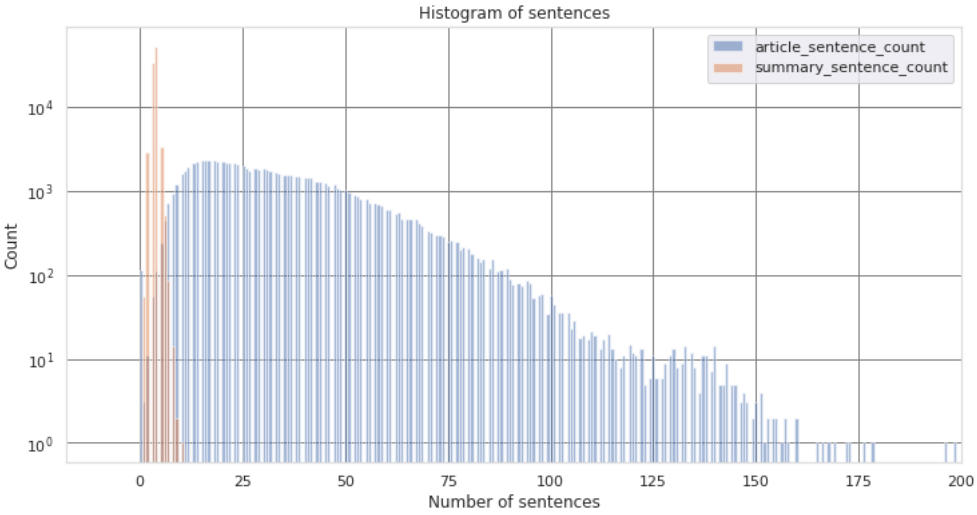

Figure 3.4: Histogram of Sentences

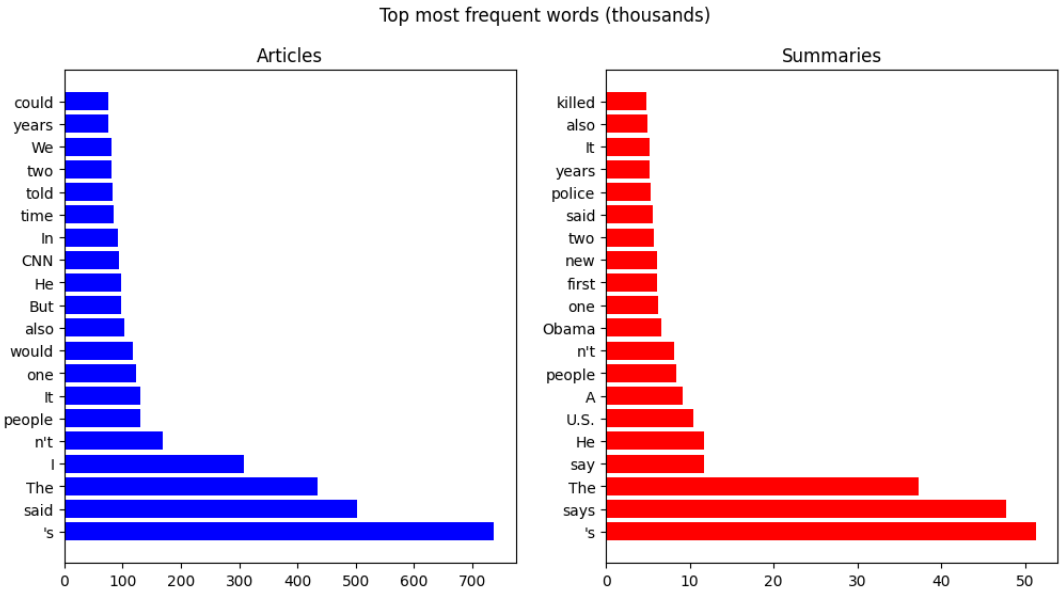

Figure 3.5: Top most frequent words

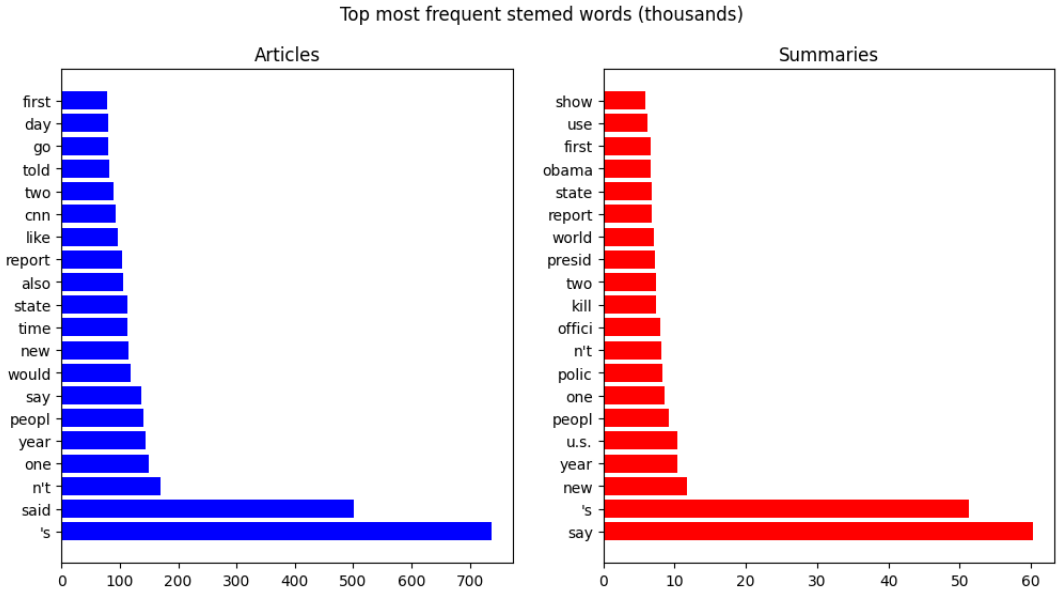

Figure 3.6: Top most frequent stemed words

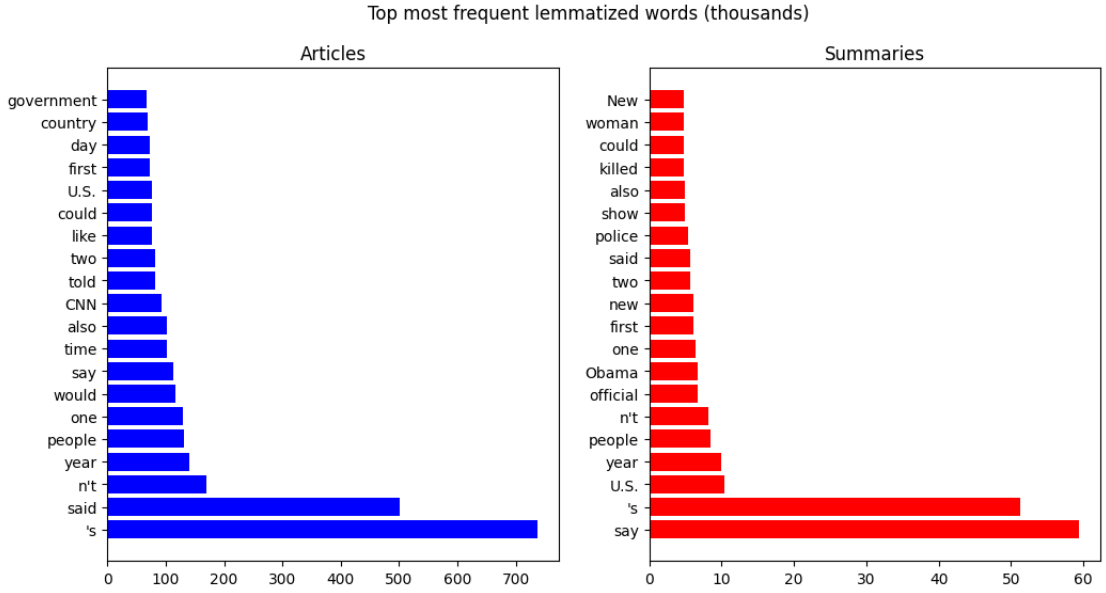

Figure 3.7: Top most frequent lemmatized words

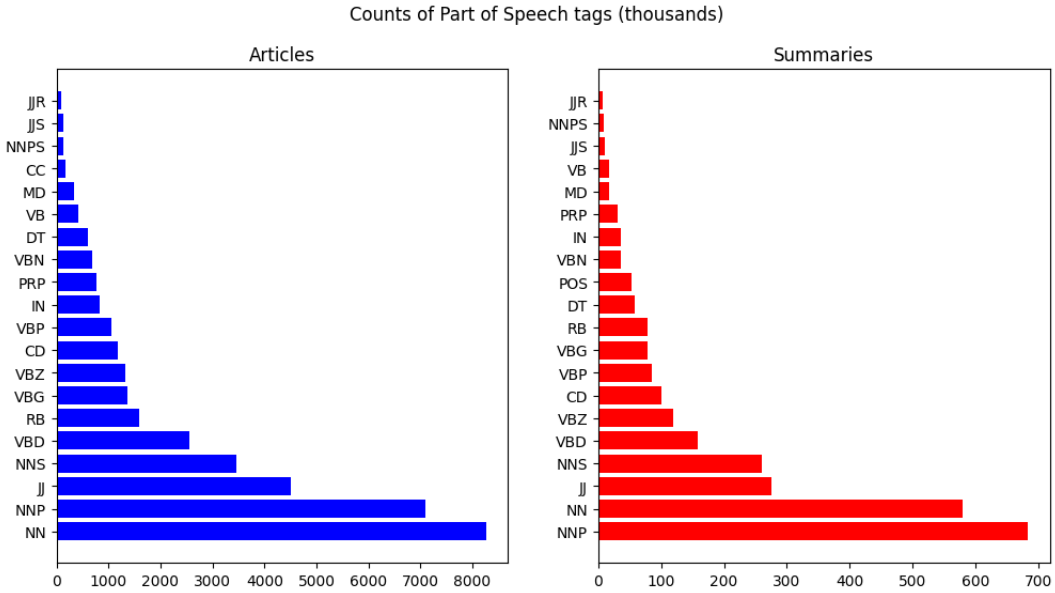

Figure 3.8: Counts of POS Tags

After cleaned, the dataset had the following statistics:

	count	mean	std	min	25%	50%	75%	max
article_char_count	90887.0	3908.98	2030.70	103.0	2299.0	3602.0	5180.0	11722.0
article_word_count	90887.0	657.78	343.73	14.0	385.0	606.0	874.0	1899.0
article_sentence_count	90887.0	34.08	20.10	3.0	19.0	30.0	45.0	356.0
summary_char_count	90887.0	264.68	57.68	52.0	223.0	268.0	310.0	610.0
summary_word_count	90887.0	42.58	9.80	7.0	36.0	43.0	50.0	106.0
summary_sentence_count	90887.0	3.63	0.65	1.0	3.0	4.0	4.0	10.0

Figure 3.9: Detailed Dataset Statistics After Cleaning

3.2 Preprocessing

Preprocessing is one of the major tasks in any kind of text model. During this we aim to deep clean our data by removing Punctuations , Stopwords or converting words to their root or stem words.
We are not coverting everything to lowercase as there might be few problems such as U.S converted to us.

3.2.1 Tokenization

Each individual word in a sentence is referred as a token.Each sentence is split into words either using nltk library or general .split() python methods.

3.2.2 Punctuation Removal

Punctuation are the unnecessary symbols that are in our corpus documents.We are just going to remove these -

```
!\"#$%&()*+-./:;<=>@[\]^_‘{|}~\n
```

3.2.3 Stopwords Removal

Stop words are the most commonly occurring words that dont give any additional value to the document. We can just use the nltk library and iterate over all the words to remove the stop words.

3.2.4 Stemming vs Lemmatization

This is the final and most important part of the preprocessing. Stemming converts words to their stem and Lemmatization to root word.
For stemming we have used nltk library's Porter-Stemmer . The words given by the stemmer need not be meaningful few times, but it will be identified as a single token .

Lemmatisation is a way to reduce the word to the root synonym of a word. Unlike Stemming, Lemmatisation makes sure that the reduced word is again a dictionary word .For Lemmatisation we have used nltk library's WordNetLemmatizer

Stemming - need not be a dictionary word
lemmatization - will be a dictionary word. reduces to a root synonym.

3.3 Extractive Text Summarization

3.3.1 Introduction

Extractive Text summarization is the technique of generating relevant summaries from a larger size document by retaining only relevant sentences from the text and the sentences are not changed at all.

Sometimes performance of extractive summaries is better as compared to abstractive summaries. This is because abstractive methods exploit the concept of semantics and new information is inferred which may or may not make sense and it is also comparatively difficult as compared to sentence extraction for extractive based methods.

Steps followed by Extractive based Summarization methods.

- Preprocessing of text for summarization
- A numeric value i.e. score assigned to each of the sentences based on method used.
- Ranking of the sentences using the score either by sorting or other ranking algorithm. Higher ranked sentences are preferred over lower ranked sentence for summary

3.3.2 Frequency Based Approach

The most basic approach to summarize articles is the frequency based approach.The frequency enables us to find the relevant and not-so-relevant parts of the article by determining the importance of words in the article. The most frequent words generally constitutes the topic expressed in the article.

Word Probability Method

Word Probability Score is calculated as frequency of the occurrence of word in the article divided by total count of words in the sentence.

$$Score = \sum P(word)$$

$$P(word) = \frac{frequency(word)}{len(sentence)}$$

After counting the frequency of individual words across the document, well be scoring each sentence by its words i.e., adding the Word Probability Scores of all words in the sentence. To ensure long sentences do not have unnecessarily high scores over short sentences, each score of a sentence is divided by the number of words in that sentence.
Now we select the required number of sentences by sorting this score.

Term Frequency – Inverse Document Frequency (TF-IDF)

The TF-IDF technique, in which uniqueness is given more importance or relevance to create the feature vector and it solves the problem of determining how rare or common a word is

$$TF - IDF(t) = TF(t) * IDF(t)$$

$$TF(t) = \frac{Frequency(t)}{Total\ number\ of\ terms\ in\ the\ document}$$

$$IDF(t) = log_e(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ term\ t\ in\ it})$$

TF-IDF reduces the impact of common occurring words which are more frequent in nature by relating to its corresponding count of occurrences in the document set.

We will be considering each sentence as a document and Total no of documents will be the total no of sentences in the article.

Steps involved in TF-IDF approach :

- Calculate Term Frequency of the words in each sentence
- Calculate the Document Frequency of the words in each sentence
- Calculate IDF value
- Calculate TF-IDF value
- Score the sentences according to the values.
- Select sentences using threshold value or select the top n sentences wrt to TF-IDF values.

```
def run_article_summary(article, tokenizer = 'stem', n_gram = '1-gram', threshold_factor = 1):
    
    frequency_table = _create_dictionary_table(article, tokenizer, n_gram)
    sentences = sentence_tokenize(article)
    sentence_scores = _calculate_sentence_scores(sentences, frequency_table, tokenizer, n_gram)
    average_score = _calculate_average_score(sentence_scores)
    article_summary = _get_article_summary(sentences, sentence_scores, (threshold_factor*average_score))
    return article_summary
```

Figure 3.10: Code - Term Frequency Based Summarization

```
def _calculate_sentence_scores(sentences, frequency_table, tokenizer='stem', n_gram='1-gram') -> dict:
    sentence_weight = dict()
    for sentence in sentences:
        words_list = tokens_without_punctuation(sentence)
        token_list = _create_list_of_tokens(words_list, tokenizer)
        n_gram_list = _create_list_of_ngrams(token_list, n_gram)

        sentence_count = 0

        for n_gram_item in n_gram_list:

            if n_gram_item in frequency_table:
                sentence_count += 1

                if sentence in sentence_weight:
                    sentence_weight[sentence] += frequency_table[n_gram_item]
                else:
                    sentence_weight[sentence] = frequency_table[n_gram_item]

        if sentence in sentence_weight and sentence_weight[sentence] > 0:
            sentence_weight[sentence] = sentence_weight[sentence] / sentence_count
        else:
            sentence_weight[sentence] = 0
    return sentence_weight
```

Figure 3.11: Code - Sentence Scoring using Term Frequency

3.3.3 Feature based approach

A number of features for scoring of sentences at world level, sentence level have been proposed by different researchers.

Linear combination of the weights and feature's value is done for calculating the sentence score.

$$Score = \sum w_i * F_i$$

Following are the features that we have implemented for scoring :

1. Term Frequency F1

$$F1 = \frac{Frequency(t)}{Total\ number\ of\ terms\ in\ the\ article}$$

2. Sentence Location F2

$$F2 = 1 - \frac{i-1}{N}$$

3. Sentence length F3

$$F3 = \frac{|S_i|}{max|S|}$$

4. Title Words Similarity F4

$$F4 = No\ of\ words\ in\ S_i\ present\ in\ Title$$

5. Cue Word F5

$$F5 = No\ of\ cue\ words\ present\ in\ S_i$$

6. Proper Noun F6

$$F6 = \frac{No\ of\ Proper\ Nouns\ present\ in\ S_i}{|S_i|}$$

7. Pronouns F7

$$F7 = \frac{No\ of\ Pronouns\ present\ in\ S_i}{|S_i|}$$

8. Numerical Value in Sentence F8

$$F8 = \frac{No\ of\ Numerical\ data\ present\ in\ S_i}{|S_i|}$$

9. Thematic Features F9: The top n frequent words were considered as thematic words.

$$F9 = No\ of\ thematic\ words\ in\ S_i$$

10. Word Co-occurrence matrix Defuzzification F10 : Their may be chances that a few terms are cooccuring in the sentences in the same manner and position. These cooccuring words can be given higher weight so that the important information can be present in final summary.

```
def feature0(sentences,senno, frequency_table) -> dict:

    sentence_weight = dict()

    for sentence in sentences:
        words_list = tokens_without_punctuation(sentence)
        token_list = _create_list_of_tokens(words_list,"lemma")
        count=0

        for token in token_list:
            count+=1
            if token in frequency_table:
                if sentence in sentence_weight:
                    sentence_weight[sentence] += frequency_table[token]
                else:
                    sentence_weight[sentence] = frequency_table[token]

        if sentence in sentence_weight and sentence_weight[sentence] > 0:
            sentence_weight[sentence] = sentence_weight[sentence] / count
        else:
            sentence_weight[sentence] = 0
    return sentence_weight
```

Figure 3.12: Code - Feature Term Frequeny

```
# Title Words Similarity

def feature1(titlewords,sentences,senno):
    f1=[0]
    for i in range (0,senno-1):
        f1.append(0)

    for i in range (0,senno):
        sentences[i]=sentences[i].split(" ")

    for i in range (0,senno):
        for wordtemp in sentences[i]:
            if wordtemp in titlewords:
                f1[i]=f1[i]+1;
        f1[i]=f1[i]/len(titlewords)
    return f1
```

Figure 3.13: Code - Feature Title Word Similarity

```
#Normalised length calculation
def feature2(sentences,senno):
    max=0
    for i in sentences:
        if len(i)>max:
            max=len(i)

    f2=[0]
    for i in range (0,senno-1):
        f2.append(0)

    for i in range(0, senno):
        f2[i]=float(len(sentences[i])/max)
    return f2
```

Figure 3.14: Code - Feature Sentence Length

```
# Sentence Position

def feature3(sentences, senno):
    f3=[0]
    for i in range (0,senno-1):
        f3.append(0)
    for i in range(0, senno):
        f3[i]=(senno-i)/senno
    return f3
```

Figure 3.15: Code - Feature Sentence Position

```
#Numerical data

def feature4(sentences,senno):
    f4=[0]
    for i in range(0, senno-1):
        f4.append(0)
    for i in range(0, senno):
        for word in sentences[i]:
            for char in word:
                if(char.isdigit()):
                    f4[i]=f4[i]+1
        f4[i]= f4[i]/len(sentences[i])
    return f4
```

Figure 3.16: Code - Feature Numerical Data

```
# Proper Noun

def feature5(sentences2,senno):
    f5=[0]
    for i in range(0,senno-1):
        f5.append(0)
    max=0
    for i in range(0,senno):
        sentence=sentences2[i]
        tagged_sent = pos_tag(sentence.split())
        propernouns = [word for word, pos in tagged_sent if pos == 'NNP']
        f5[i]= len(propernouns)/len(sentence.split())
    return f5
```

Figure 3.17: Code - Feature No. of ProperNouns

```
#·Pronouns

def·feature6(sentences2,senno):
····f6=[0]
····for·i·in·range(0,senno-1):
····|···f6.append(0)
····for·i·in·range(0,senno):
····|···sentence=sentences2[i]
····|···tagged_sent·=·pos_tag(sentence.split())
····|···pronouns·=·[word·for·word,·pos·in·tagged_sent·if·(pos·==·'PRP$'·or·pos=='PRP')]
····|···f6[i]=·len(pronouns)/len(sentence.split())
·····
····return·f6
```

Figure 3.18: Code - Feature No. of Pronouns

```
#·Similarity·matrix

def·feature7(sentences,·senno):
····simmat=[[0]*senno·for·x·in·range(senno)]
····
····for·i·in·range(0,senno):
····|···for·j·in·range(i+1,senno):
····|···|···#print(i,j)
····|···|···for·word·in·sentences[j]:·····
····|···|···|···if·word·in·sentences[i]:
····|···|···|···|···simmat[i][j]=simmat[i][j]+1
····|···|···|···|···simmat[j][i]=simmat[i][j]·····
····return·simmat
```

Figure 3.19: Code - Feature Word Co-occurance

```
··#defuzzification·for·similarity·matrix
··f7=[]
··for·i·in·range(0,senno):
··|···f7.append(0)
··
··for·i·in·range(0,senno):
··|···m=·max(simmat[i])
··|···m=m/len(sentences[i])
··|···f7[i]=m
```

Figure 3.20: Code - Defuzzification of Similarity Matrix

```
def·feature8(thematic_words,sentences,senno):
····f8=[0]
····for·i·in·range·(0,senno-1):
····|···f8.append(0)

····for·i·in·range·(0,senno):
····|···sentences[i]=sentences[i].split("·")
·····
····for·i·in·range·(0,senno):····
····|···for·wordtemp·in·sentences[i]:
····|···|···if·wordtemp·in·thematic_words:
····|···|···|···f8[i]=f8[i]+1;
····|···f8[i]=f8[i]/len(thematic_words)
····return·f8
```

Figure 3.21: Code - Feature Thematic Words

```
#·Cue·Phrases·Matching
def·feature9(sentences):
····list_1=sentences

····list_2=cue_phrases
····count_dict={}
····for·l·in·list_1:
····    c=0
····    for·l2·in·list_2:·····
····        if·l.find(l2)≠-1:
····            c=1
····            break···········
····    if·c:#
····        count_dict[l]=1
····    else:
····        count_dict[l]=0

····return·count_dict
```

Figure 3.22: Code - Feature Cue Phrases

3.3.4 Graph based approach

A graph type of structure is best suited for representing relations between sentences. The similarity of sentence is the only feature on which centrally this approach is based.
The inspiration for the graph based summarisation come from Pagerank Algorithm.

Our focus for text summarisation is to rank the sentences according to their score. This score is calculated using the PageRank Algorithm using the **sentence_similarity_graph** created.
After the sentences are ranked and sorted by their ranking, the most important sentences are included in the summary either using threshold values or top n sentences.

PageRank Algorithm at a glance :
The article is represented in the form of a undirected graph where the incoming and outgoing nodes are considered to be the same. The sentences are considered as the vertices and edges are the relationship between different sentences.The purpose of this being the number of words that are same between two sentences are always equal in a pair irrespective of computed in any direction. The edge weight defines the relationship between the two sentences. Weight stands for the number of words similar between two pairs of sentences. This weight is represented in the edge between the sentences. The summation of all weights computes the total weight of a sentence it has from all edges. The incoming edges of a sentence are the similarity of the sentence with every other sentence in the article.

- Preprocess the text and extract sentences
- Create similarity matrix among all sentences
- Score the sentences in similarity matrix using PageRank Algorithm
- To select the sentences either use threshold value or sort the scores
- get the top n number of sentences based on score obtained from PageRank

To calculate sentence similarity , we will be vectorizing both the sentences and then calculation the **cosine distance between both vectors as a similarity score**.

To vectorize the sentences, we are using two different methods. One involves **Normal vectorization using sentence length** while other using **Glove Embeddings**.

```
#·Extract·word·vectors
word_embeddings·=·{}
f·=·open('./glove/glove.6B.100d.txt',·encoding='utf-8')
for·line·in·f:
····values·=·line.split()
····word·=·values[0]
····coefs·=·np.asarray(values[1:],·dtype='float32')
····word_embeddings[word]·=·coefs
f.close()


def·create_vector(sentence):
····v·=·sum([word_embeddings.get(w,·np.zeros((100,)))·for·w·in·sentence])/(len(sentence)+0.001)
····return·v
```

Figure 3.23: Code - Glove Word Embeddings

```
#·Create·vectors·and·calculate·cosine·similarity·b/w·two·sentences
def·sentence_similarity(sent1,sent2,method):
····if·method=="glove":
····|···vector1·=·create_vector(sent1)
····|···vector2·=·create_vector(sent2)
····|···return·1-cosine_distance(vector1,vector2)
····|····
····else:
····|···sent1·=·[w.lower()·for·w·in·sent1]
····|···sent2·=·[w.lower()·for·w·in·sent2]

····|···all_words·=·list(set(sent1·+·sent2))


····|···all_words·=·list(set(sent1·+·sent2))

····|···vector1·=·[0]·*·len(all_words)
····|···vector2·=·[0]·*·len(all_words)

····|···for·w·in·sent1:
····|···|···if·not·w·in·stopwords:
····|···|···|···vector1[all_words.index(w)]+=1

····|···for·w·in·sent2:
····|···|···if·not·w·in·stopwords:
····|···|···|···vector2[all_words.index(w)]+=1

····|···return·1-cosine_distance(vector1,vector2)
····
····|····
```

Figure 3.24: Code -Vectorization and Sentence Similarity

```
#·Create·similarity·matrix·among·all·sentences
def·build_similarity_matrix(sentences,method):
····similarity_matrix·=·np.zeros((len(sentences),len(sentences)))

····for·idx1·in·range(len(sentences)):
····|···for·idx2·in·range(len(sentences)):
····|···|···if·idx1≠idx2:
····|···|···|···similarity_matrix[idx1][idx2]·=·sentence_similarity(sentences[idx1],sentences[idx2],method)

····return·similarity_matrix
```

Figure 3.25: Code -Creating Similarity Matrix

```
···
···orig_sentences=sentence_tokenize(text)
···sentence_similarity_matrix·=·build_similarity_matrix(orig_sentences,method)
···sentence_similarity_graph·=·nx.from_numpy_array(sentence_similarity_matrix)
···scores·=·nx.pagerank(sentence_similarity_graph,tol=1.0e-3)
···res=statistics.mean(list(scores.values()))

···ranked_sentences·=·sorted(((scores[i],s)·for·i,s·in·enumerate(orig_sentences)),reverse=True)

···for·i·in·range(len(ranked_sentences)):
···|···if·ranked_sentences[i][0]·⩾res:
···|···|···summarize_text.append(ranked_sentences[i][1])
```

Figure 3.26: Code -TextRank Algorithm

3.3.5 Semantic based approach - LSA

The basic idea behind the use of LSA in text summarization is that words that usually occur in related contexts are also related in the same singular space. LSA transforms sentence vectors from a term-space of non-orthogonal features to a concept-space of lower dimensionality with an orthogonal basis. This is done by performing singular value decomposition of term sentence matrix A. This gives us three matrices - U, V and such that :

$$A = U \sum V^T$$

where U and V are orthogonal matrices and is a diagonal matrix whose diagonal elements represent the relative importance of each concept dimension in the basis of the concept space. The columns of U and the rows of V T are respectively the vectors corresponding to the terms and sentences in the concept space. By retaining on top k concepts (in terms of their eigen values), we the best k-rank approximation to A in the least squares sense.

Mathmatically, if v is the sentence vector in the concept space and i is the eigen value of the ith concept, the score of a sentence is given by

$$Score(V) = \sqrt{\sum_i \sigma_i^2 v_i^2}$$

Steps followed :

- For the preprocessed sentences calculate TF-IDF weights
- Get the sentence vectors V using the TF-IDF values.
- Applying SVD to the vectors obtained using TF-IDF weights
- Get the top k singular values S with the help of SVD.
- Apply a threshold-based approach to remove singular values . This is a heuristic, and we can choose values according to our preference or take it as mean of all values.
- To get sentence weights per topic, multiply each term sentence column from V squared with its corresponding singular value from S also squared .
- Compute the sum of the sentence weights across the topics and take the square root of the final score to get the salience scores for each sentence in the document

```
def normalize_document(doc):
    doc = re.sub(r'[^a-zA-Z\s]', '', doc, re.I|re.A)
    doc = doc.lower()
    doc = doc.strip()
    tokens = nltk.word_tokenize(doc)
    filtered_tokens = [token for token in tokens if token not in stop_words]
    porter = PorterStemmer()
    filtered_tokens = [porter.stem(word) for word in filtered_tokens]
    doc = ' '.join(filtered_tokens)
    return doc

normalize_corpus = np.vectorize(normalize_document)
```

Figure 3.27: Code - LSA Sentence Normalization

```
def low_rank_svd(matrix, singular_count=10):
    u, s, vt = svds(matrix, k=singular_count)
    return u, s, vt
```

Figure 3.28: Code - SVD Calculation

```
def lsa(text,threshold=0.5):
    text = re.sub(r'\n|\r',' ', text)
    text = re.sub(r' +',' ', text)
    text = text.strip()
    sentences = nltk.sent_tokenize(text)

    norm_sentences = normalize_corpus(sentences)
    
    tv = TfidfVectorizer(min_df=0., max_df=1., use_idf=True)
    dt_matrix = tv.fit_transform(norm_sentences)
    dt_matrix = dt_matrix.toarray()

    vocab = tv.get_feature_names_out()
    td_matrix = dt_matrix.T
    num_sentences = int(len(nltk.sent_tokenize(text)) * threshold)
    num_topics = 1

    u, s, vt = low_rank_svd(td_matrix, singular_count=num_topics)
    term_topic_mat, singular_values, topic_document_mat = u, s, vt

    sv_threshold = threshold
    salience_scores = np.sqrt(np.dot(np.square(singular_values), np.square(topic_document_mat)))
    min_sigma_value = sv_threshold * mean(salience_scores)
    salience_scores[salience_scores ≤ min_sigma_value]=0
    top_sentence_indices = (-salience_scores).argsort()[salience_scores≠0]
    top_sentence_indices.sort()
    summary='\n'.join(np.array(sentences)[top_sentence_indices])

    return summary
```

Figure 3.29: Code - LSA Main

3.4 Abstractive Text Summarization

3.4.1 Introduction

Abstractive text summarising is a method for creating an easily understood and logical summary of a lengthy text that captures the key concepts and pertinent details. Abstractive summarization includes creating new sentences that carry the same meaning as the original text, as opposed to extractive summarization, which chooses and combines sentences from the original text.

For abstractive summarization to be successful, natural language generation (NLG) methods must be able to comprehend the content of the source text and produce grammatically accurate and fluid summaries. These methods might include computational linguistics approaches, neural networks, and machine learning algorithms.

Abstractive summarization is a difficult undertaking because it demands the model to comprehend the intricacies of the language and the context of the text in order to produce summaries that are comparable to those produced by humans. Since there may be numerous legitimate ways to summarise a text, it is equally challenging to determine the quality of abstractive summaries.

The general steps followed by abstractive text summarization:

- Preprocessing: The input text is preprocessed by removing stop words, punctuations, and other irrelevant information to reduce noise and improve the quality of the summary.
- Representation: The input text is represented in a machine-readable format, such as vectors or matrices, that can be processed by the model.
- Model Training: A model is trained on a large corpus of text using deep learning algorithms and natural language processing techniques to learn the patterns and structures of the language.
- Decoding: The model generates a summary by decoding the input representation and generating new sentences that capture the main ideas and important information of the input text.
- Evaluation: The generated summary is evaluated based on metrics such as ROUGE, BLEU, and other measures of summary quality to ensure that it is coherent, relevant, and accurate. Refinement: The model is refined and fine-tuned based on the evaluation results to improve the quality of the generated summaries.

3.4.2 Encoder-Decoder Model

The encoder-decoder model is a neural network architecture used in abstractive text summarization. It consists of two main components: an encoder and a decoder. The encoder processes the input sequence and generates a fixed-length vector representation of its meaning. The decoder generates an output sequence based on this representation, word by word.

Recurrent neural networks, such as LSTMs or GRUs, are commonly used as building blocks for both the encoder and decoder. The encoder's final hidden state is used as the fixed-length representation of the input sequence. The decoder generates output words by sampling from a probability distribution conditioned on the previous hidden state and generated words.

Attention mechanisms and beam search are techniques used to improve the performance of the model. Attention mechanisms allow the decoder to focus on different parts of the input sequence, while beam search explores multiple possible output sequences in parallel. The encoder-decoder model for abstractive text summarization can generate summaries that are both concise and informative. However, it requires large amounts of training data and careful hyperparameter tuning.

Evaluation metrics such as ROUGE are used to measure the quality of generated summaries. The encoder-decoder model has also been applied to other tasks such as machine translation and speech recognition. Future research directions include exploring more sophisticated architectures and training strategies, as well as developing better evaluation metrics for abstractive summarization.

Terminologies used:

RNN

RNN stands for Recurrent Neural Network, which is a type of neural network that is designed to process sequential data, such as time series or natural language. It has a memory element that allows it to maintain information about previous inputs, which enables it to

capture patterns and dependencies in sequential data. RNNs are widely used in various natural language processing tasks, such as language modelling, machine translation, and text classification. The most common types of RNNs are the Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks, which address the issue of vanishing gradients and improve the performance of RNNs on long sequences.

LSTM

LSTM stands for Long Short-Term Memory, which is a type of recurrent neural network (RNN) architecture that is designed to overcome the vanishing gradient problem in traditional RNNs. LSTMs have a memory cell that can maintain information over long periods of time and three gates (input, forget, and output) that regulate the flow of information into and out of the cell. These gates use sigmoid and element-wise multiplication operations to control the amount of information that is kept or discarded, allowing the LSTM to selectively remember or forget past inputs. LSTMs are widely used in various natural language processing tasks, such as language modelling, machine translation, and text classification, where they have demonstrated superior performance compared to traditional RNNs.

GRU

GRU stands for Gated Recurrent Unit, which is a type of recurrent neural network (RNN) architecture that is designed to address the issue of vanishing gradients in traditional RNNs. GRUs have a simpler structure than Long Short-Term Memory (LSTM) networks, with only two gates (reset and update) that regulate the flow of information into and out of the hidden state. These gates use sigmoid and element-wise multiplication operations to control the amount of information that is kept or discarded, allowing the GRU to selectively update its hidden state based on the current input and the previous hidden state. GRUs are widely used in various natural language processing tasks, such as language modelling, machine translation, and text classification, where they have demonstrated similar or slightly inferior performance compared to LSTMs but with lower computational cost.

Steps involved in Encoder-Decoder Model :

- The encoder takes an input sequence and transforms it into a fixed-length vector that represents the meaning of the input sequence.
- The encoder typically uses a recurrent neural network, such as LSTM or GRU, to process the input sequence and generate a hidden state at each time step.
- The final hidden state of the RNN is used as the fixed-length representation of the input sequence.
- The decoder generates the summary using another RNN that takes the fixed-length representation as input and generates the output sequence word by word.
- At each time step, the decoder generates a probability distribution over the vocabulary of possible output words, conditioned on the previous hidden state and the previously generated words.
- The next word in the summary is then sampled from this distribution.

- The decoder continues to generate words until it produces an end-of-sequence token or reaches a maximum length.
- Attention mechanisms allow the decoder to focus on different parts of the input sequence at each time step.
- Beam search is a search algorithm that explores multiple possible output sequences in parallel.
- The encoder-decoder model requires large amounts of training data and careful tuning of various hyperparameters to achieve good performance.
- The model can generate concise and informative summaries by understanding the input text and generating the summary through the decoder.

3.4.3 BART Model or Sequence-to-Sequence Model

BART stands for Bidirectional and Auto-Regressive Transformer. Its primary features are a bidirectional encoder from BERT and an autoregressive decoder from GPT. The encoder and decoder are united using the seq2seq model to form the BART algorithm. Let us look at it in more detail.

Architecture

BART consists of a stack of transformer encoders and decoders. Each transformer encoder takes as input a sequence of tokens and produces a hidden representation of that sequence. Similarly, each transformer decoder takes as input a sequence of tokens and produces a sequence of hidden representations that correspond to the target sequence. BART uses a bidirectional encoder, which means that it can use both the left and right contexts of a sequence to generate the hidden representations. This is in contrast to other autoregressive models like GPT-2, which use a unidirectional left-to-right encoder.

Pretraining

BART is pretrained on a combination of denoising autoencoding and masked language modeling objectives. In the denoising autoencoding objective, the model is trained to reconstruct a corrupted version of the original input sequence. The corruption can be done in various ways, such as replacing tokens with a special [MASK] token, shuffling the order of tokens, or deleting tokens at random. In the masked language modeling objective, the model is trained to predict the masked tokens in a sequence, where a certain percentage of tokens are randomly masked during training. BART is pretrained on a large corpus of text data, such as Wikipedia and BookCorpus, and is fine-tuned on downstream NLP tasks.

Steps involved in BART Model

- Load the BART model and tokenizer from the Hugging Face Transformers library. The BART model is a transformer-based neural network architecture that is specifically designed for natural language processing tasks like text summarization. The tokenizer is used to convert text into a sequence of numerical tokens that can be fed into the model.

- Load the input text that you want to summarize. This text should be a long piece of text, such as an article or a research paper.
- Preprocess the input text by performing any necessary cleaning and formatting steps. This might include removing punctuation or converting the text to lowercase.
- Tokenize the preprocessed input text using the BART tokenizer. This will convert the text into a sequence of numerical tokens that can be fed into the BART model.
- Feed the tokenized input text into the BART model to generate a summary. The BART model will use its pre-trained knowledge to generate a concise and informative summary of the input text.
- Postprocess the summary by converting the numerical tokens back into human-readable text. This might involve using the BART tokenizer to decode the summary sequence or performing other formatting steps.
- Output the summary of the input text. This summary should capture the main points and key ideas of the input text in a concise and informative manner.

Evaluation metric and Result

4.1 Evaluation Metric

4.1.1 Precision, Recall, and F-Scores

$$Precision = \frac{S_{reference} \bigcap S_{candidate}}{S_{candidate}}$$

$$Recall = \frac{S_{reference} \bigcap S_{candidate}}{S_{reference}}$$

$$F - Score = \frac{2 * Precision * Recall}{Precision + Recall}$$

4.1.2 ROUGE - N

Recall-Oriented Understudy for Gisting Evaluation (ROUGE) is the most commonly used tool for automatic evaluation of the automatically generated summaries. ROUGE is a set of metrics and a software package used for evaluating automatic summarization and machine translation software in NLP . It compares the computer-generated summaries against several human-created reference summaries.

The basic idea of ROUGE is to count the number of overlapping units between candidate (or system) summaries and the reference summaries, such as overlapped n-grams. ROUGE has been proven to be effective in measuring qualities of summaries and correlates well to human judgments.

There are various ROUGE metrics as follows:

- ROUGE-1 (R1): it is based on the uni-gram measure between a candidate summary and a reference summary. ROUGE-N is an n-gram recall between a candidate summary and reference summaries.
- ROUGE-L (R-L): it is based on the longest common subsequences between a candidate summary and a reference summary.
- ROUGE-S* (R-S*): it measures the overlap ratio of skip-bigrams between a candidate summary and reference summaries.
- ROUGE-SU* (R-SU*): it extends ROUGE-S* by using skipbigrams and using unigram as a counting unit. The * refers to the number of skip words. For example, ROUGE-SU4 permits bi-grams to consist of non-adjacent words with a maximum of four words between the two.

```
import rouge
def rouge_scoring(hypothesis, reference, max_n = 1, alpha = 0.5, score = 'F1'):
    
    evaluator = rouge.Rouge(metrics = ['rouge-n'],
                            max_n = max_n,
                            limit_length = False,
                            alpha = alpha,
                            stemming = True)
    if score == 'F1':
        score_entry = 'f'
    elif score == 'Precision':
        score_entry = 'p'
    else:
        score = 'Recall'
        score_entry = 'r'

    rouge_score = evaluator.get_scores(hypothesis, reference)['rouge-' + str(max_n)]

    return rouge_score
```

Figure 4.1: Code for ROUGE metrics

4.2 Results

4.2.1 Extractive Text Summarization

For **Term Frequency Method** - 23.69 for 3-gram lemma at threshold of 1.1 on testing on 1000 articles

		Mean				
		1	1.1	1.2	1.3	1.4
1-gram	stem	20.26	22.49	23.51	22.94	20.93
	lemma	20.31	22.46	23.51	22.79	21.02
2-gram	stem	20.84	23.13	23.48	21.27	18.64
	lemma	20.83	23.15	23.32	21.41	18.64
3-gram	stem	21.39	23.54	22.57	19.97	16.67
	lemma	21.43	23.69	22.54	19.74	16.54

Figure 4.2: Results of Term Frequency Method

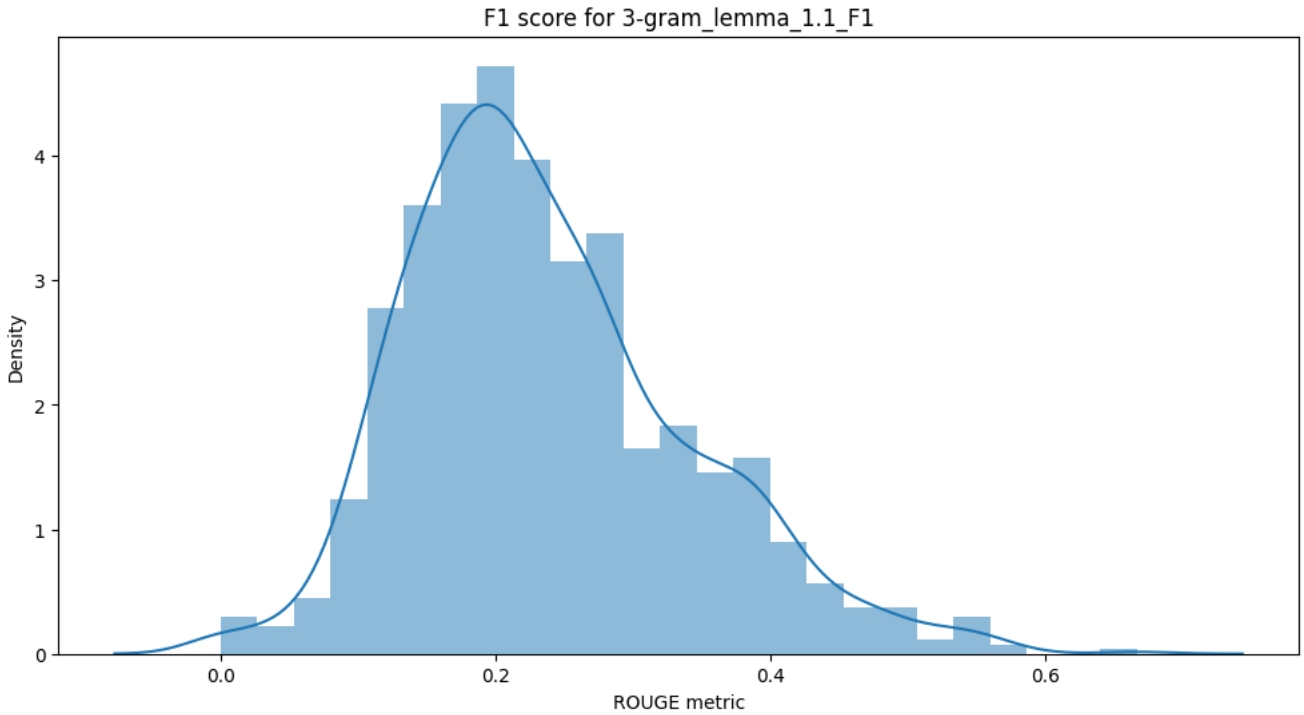

Figure 4.3: F1 Score Graph for Term Frequency Method

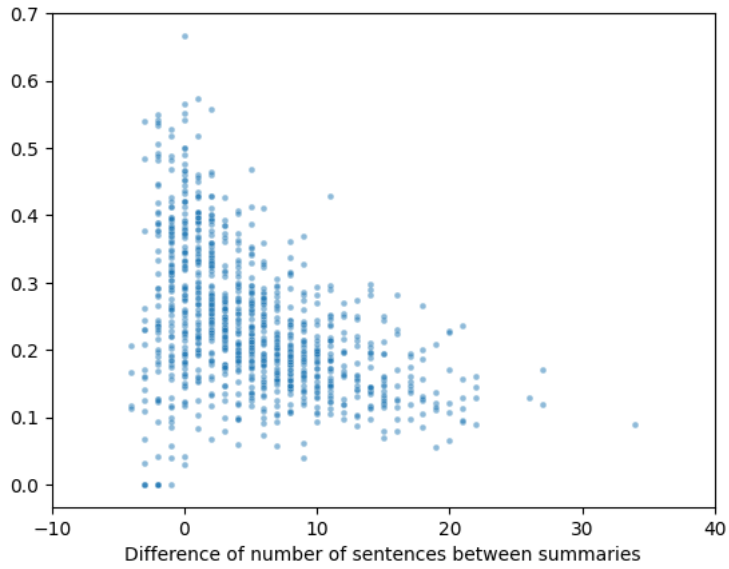

Figure 4.4: Term Frequency - Difference of sentences in Candidate and Reference Summary

For **Feature Scoring Method** with 10 features - 30.18 for stem at threshold of 1.3 at a similarity tolerance of 0.7 on testing on 1000 articles.

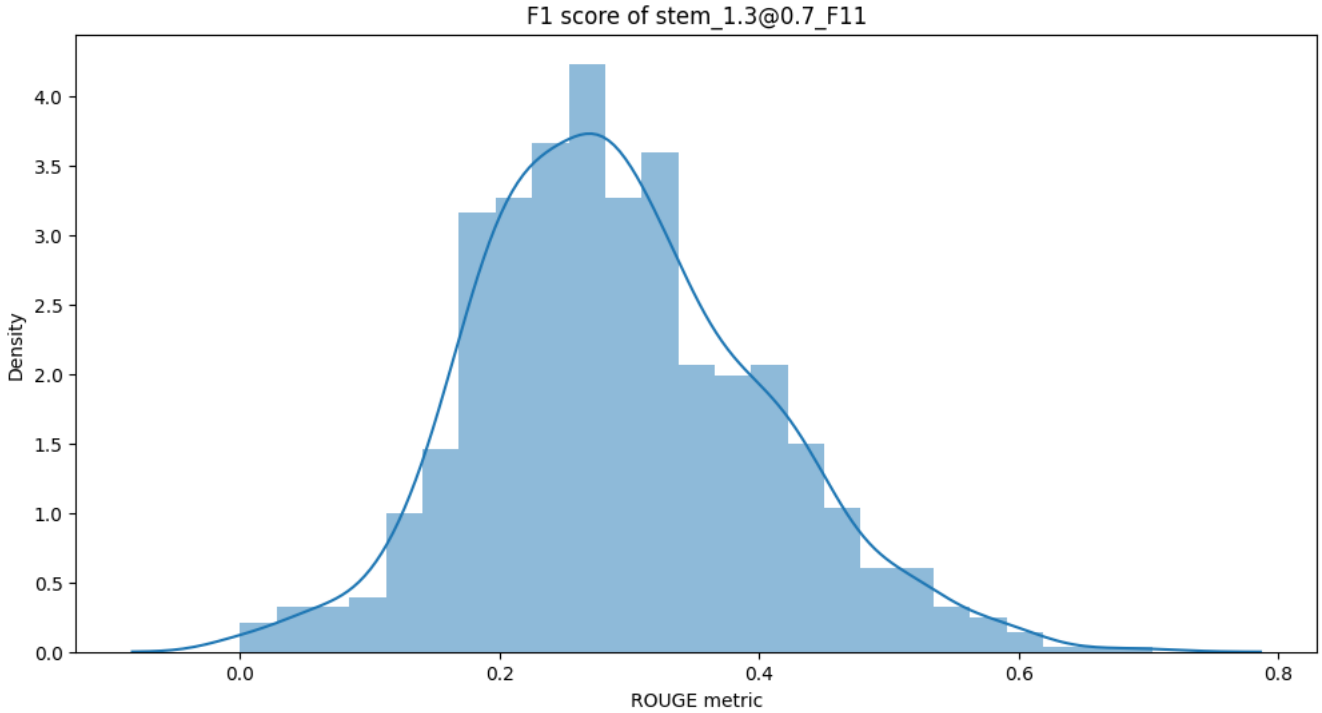

Figure 4.5: F1 Score Graph for Feature Scoring Method

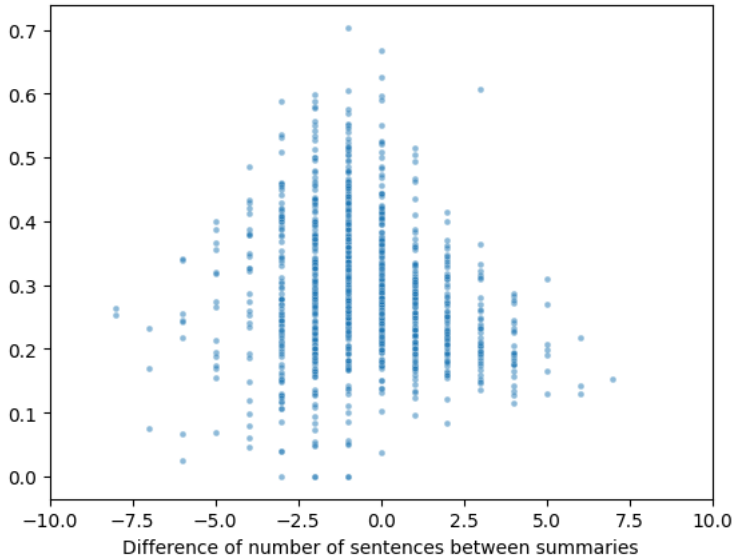

Figure 4.6: Feature Scoring - Difference of sentences in Candidate and Reference Summary

Approach	**F Score**
Word Probability	23.69
TF-IDF	24.30
TextRank Approach-Normal	25.26
TextRank Approach-Glove	25.76
Feature Based Approach	30.18
Semantic Based Approach- LSA	18.56

Table 4.1: Results - Extractive Text Summarization on 1000 articles

	Index						
	12168	34861	87066	13950	40082	84878	87026
Term Frequency	25.21	43.33	42	51.42	24.39	25.64	22.22
Feature Based	43.03	25.73	37.58	51.48	29.72	26.1	26.49
Glove Text Rank	24.48	19	32	46.66	11.37	13.93	13.61
Normal Text Rank	24.09	18.18	34	46.66	10.57	13.56	14.93
LSA	17.64	34.61	36.36	35.89	15.30	12.88	19.52

Table 4.2: F1 Score - Extractive Text Summarization on random article

Frontend/GUI for User Inputs

5.1 Features

- Able to select the approach to use for Extractive Text Summarization
- Able to enter user defined values to parameters of the method to use
- Shows the number of sentences for both the text and summary

5.2 Screenshots

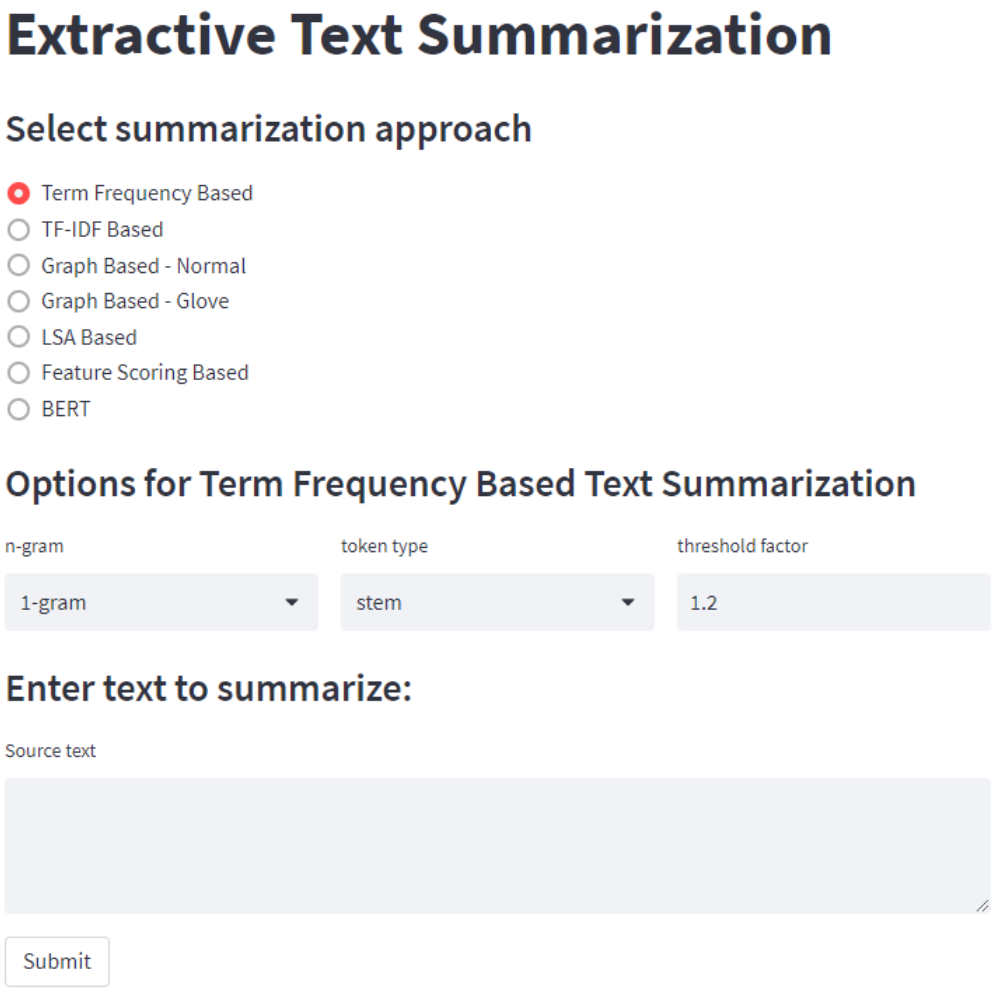

Figure 5.1: GUI Screenshot

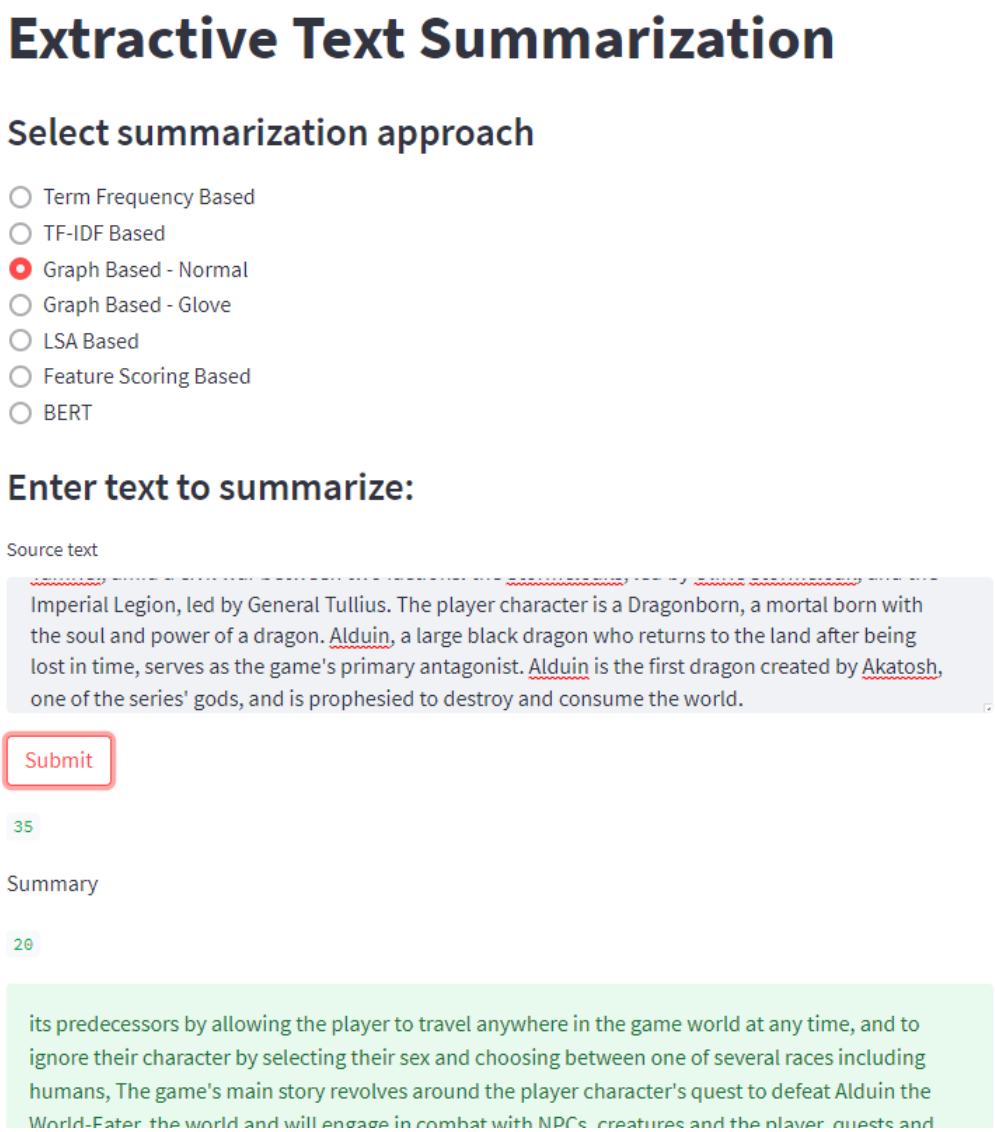

Figure 5.2: GUI Screenshot

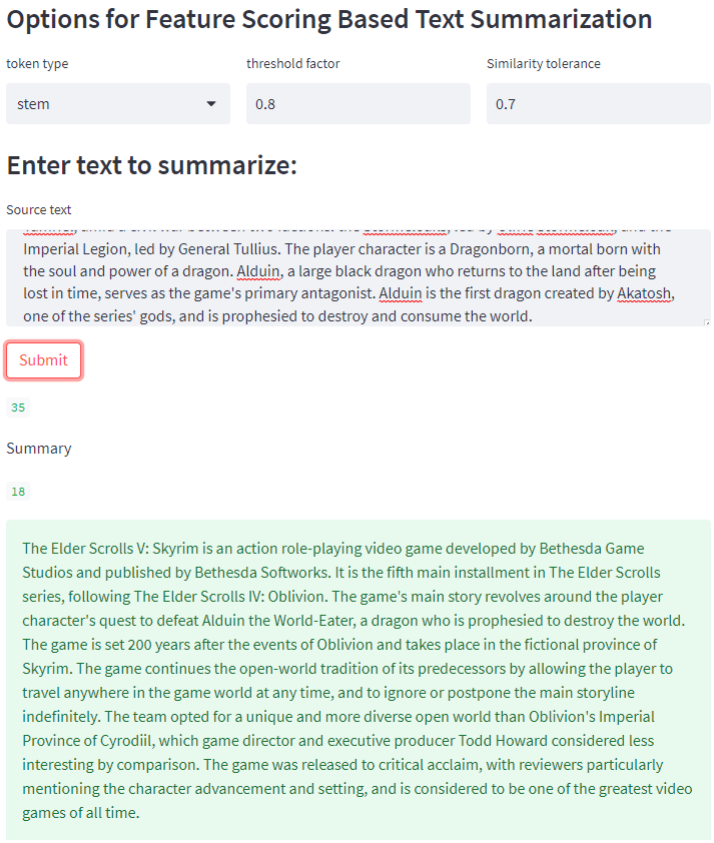

Figure 5.3: GUI Screenshot

		No of Sentences		**Ratio**
Method	**Parameters**	**Article**	**Summary**	
Term Frequency	1-gram Threshold 0.6	35	17	0.48
	2-gram Threshold 0.6	35	20	0.57
	3-gram Threshold 0.6	35	22	0.62
Text Rank	Normal	35	20	0.57
	Glove Embedding	35	23	0.65
Feature Based	Threshold 1 Tolerance 0.6	35	16	0.45
Latent Semantic Analysis(LSA)	Threshold 0.6	35	18	0.51

Table 5.1: Extractive Text Summarization on same article

Errors and Future Work

6.1 Errors and Caveats

Text summarization is a challenging task, and there have been various errors associated with both extractive and abstractive approaches to summarization. Some common errors in text summarization techniques include:

- Incoherence: In extractive summarization, the summary may contain disjointed or unrelated sentences that do not flow well. In abstractive summarization, the generated summary may contain language errors or inconsistencies.
- Bias: The summarization process can introduce bias by over-emphasizing certain aspects of the text or ignoring others.
- Over-reliance on surface-level features: Extractive summarization algorithms may be biased towards selecting sentences with high frequency or importance scores based on surface-level features such as sentence length or word frequency, which may not accurately reflect the content of the text.
- Limited scope: Text summarization algorithms may not always capture the full scope of a text, particularly when dealing with longer documents or complex topics.
- Disadvantage of ROUGE is that it does not look for a sentence-level mapping between the generated and model summaries. To address this, we also considered the following evaluation scheme.

6.2 Future Work

Our future research will deal with

- using more than one distributional semantic models for capturing semantics in the text so that semantics are captured at the fine grain level;
- Improvement of ranking algorithms by exploring more semantic features to be incorporated into our ranking algorithms, as semantic features tend to improve overall system performance;
- testing the technique on more than one dataset;
- using BLEU (bilingual evaluation understudy) for evaluation along with the ROUGE.The paper presents a summarization technique based on the distributional hypothesis to capture the semantics of the text for producing better summaries using text summarization process.

Conclusion

Our work is producing summaries which ROUGE F1 score of **X**, which is a reasonable score compared with academic benchmarks of much more sophisticated summarization models using recurrent neural networks.

Study	Date	Authors	F1 score
Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond	Aug/2016	Ramesh Nallapati	35.5
Get To The Point: Summarization with Pointer-Generator Networks	Apr/2017	Abigail See	39 -40
A Hierarchical Structured Self-Attentive Model for Extractive Document Summarization (HSSAS)	Apr/2018	Kamal Al-Sabahi	42
Fine-tune BERT for Extractive Summarization	Mar/2019	Yang Liu	43

Figure 7.1: Benchmark Studies

References

[1] Wafaa S. El-Kassas and Cherif R. Salama and Ahmed A. Rafea and Hoda K. Mohamed. Automatic text summarization: A comprehensive survey . In Expert Systems with Applications , 2021, Volume 165

[2] A. Abuobieda, N. Salim, A. Albaham, A. Osman, and Y. Kumar. Text summarization features selection method using pseudo genetic-based model. In Information Retrieval Knowledge Management (CAMP), 2012 International Conference on, pages 193197, March 2012.

[3] M. Mendoza, S. Bonilla, C. Noguera, C. Cobos, and E. Leťon. Extractive single-document summarization based on genetic operators and guided local search. Expert Syst. Appl., 41(9):41584169, July 2014.

[4] R. Mihalcea and P. Tarau. Textrank: Bringing order into texts. In D. Lin and D. Wu, editors, Proceedings of EMNLP 2004, pages 404411, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[5] Gidiotis, A., Tsoumakas, G., A Divide-and-Conquer Approach to the Summarization of Long Documents, Speech, and Language Processing, 28, 30293040, 2020.

[6] D. Patel, S. Shah, and H. Chhinkaniwala, Fuzzy logic based multi document summarization with improved sentence scoring and redundancy removal technique, Expert Systems with Applications, 134:167-177, 2019.

[7] Widyassari, A. P., Rustad, S., Shidik, G. F., Noersasongko, E.,Syukur, A., Affandy, A., Review of automatic text summarization techniques methods, Journal of King Saud University-Computer and Information Sciences, 1-18, 2020.

[8] Liu, W., Gao, Y., Li, J., Yang, Y.:, A Combined Extractive with Abstractive Model for Summarization, IEEE Access 9, 4397043980, 2021.

[9] Chin-Yew Lin. Rouge, A package for automatic evaluation of summaries, In Text summarization branches out, pages 74-81, 2004.

[10] Saeed, M.Y., Awais, M., Talib, R., Younas, M., Unstructured Text Documents Summarization with Multi-Stage Clustering, IEEE Access, 8, 212838212854, 2020.

[11] Jang, M., Kang, P., Learning-Free Unsupervised Extractive Summarization Model, IEEE Access 9, 1435814368, 2021.

[12] Gao, Y., Xu, Y., Huang, H., Liu, Q., Wei, L., Liu, L., Jointly Learning Topics in Sentence Embedding for Document Summarization, IEEE Transactions on Knowledge and Data Engineering, 32(4), 688699, 2020

[13] Bhatia, S., A Comparative Study of Opinion Summarization Techniques, IEEE Transactions on Computational Social Systems, 8(1), 110117, 2021.

[14] Chin yew Lin. Rouge: a package for automatic evaluation of summaries. pages 2526, 2004.

[15] V. Gupta and G. S. Lehal, "A survey of text summarization extractive techniques," Journal of emerging technologies in web intelligence, vol. 2, no. 3, pp. 258-268, 2010

[16] Jain, A., et al. Extractive Text Summarization Using Word Vector Embedding. 2017 International Conference on Machine Learning and Data Science (MLDS), 2017, pp. 5155. IEEE Xplore

[17] Dalal, V., and L. Malik. A Survey of Extractive and Abstractive Text Summarization Techniques.2013 6th International Conference on Emerging Trends in Engineering and Technology, 2013,

[18] Akter, Sumya, et al. An Extractive Text Summarization Technique for Bengali Document(s) Using K-Means Clustering Algorithm. 2017 IEEE International Conference on Imaging, Vision Pattern Recognition (IcIVPR), 2017, pp. 16. IEEE Xplore,

[19] Padmakumar, Aishwarya, and Dhivya Eswaran. Extractive Text Summarization Using Latent Semantic Analysis. 2014, p. 10.

[20] Widjanarko, Agus, et al. Multi Document Summarization for the Indonesian Language Based on Latent Dirichlet Allocation and Significance Sentence. 2018 International Conference on Information and Communications Technology (ICOIACT), 2018, pp. 52024. IEEE Xplore