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

Under the guidance of

Dr. Jyoti Srivastava

Department of Computer Science & Engineering
National Institute of Technology Hamirpur
Hamirpur, India-177005

Analysis & Comparison of Text Summarization Approaches

Candidate's 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** from January 2023 to May 2023 under the guidance of **Dr. Jyoti Srivastava** , Assistant Professor, DoCSE, NIT Hamirpur.

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
Assistant Professor

Date: May 18, 2023

Convener, DBPC

Head, DoCSE

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. For evaluation purposes, ROUGE metrics have been used.

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.

Furthermore, the project report highlights the utilisation of various extractive techniques alongside a few abstractive methods for text summarization on the DeepMind CNN Stories Dataset. A comprehensive account of the dataset and the procedures employed to generate a corpus suitable for the purpose of summary evaluation is presented. The algorithms' performance has been assessed through the utilisation of ROUGE metrics, in conjunction with a comparative analysis of the outcomes. The present study has concluded by addressing the constraints and potential possibilities for future research in the domain of text summarization.

This project's contributions are twofold.:

1. On the DeepMind Q&A 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, DeepMind, CNN, ROUGE

Table of Contents

Candidate's Declaration **2**

Acknowledgement **3**

Abstract **4**

1 Introduction **9**
- 1.1 Why Text Summarization? 9
- 1.2 Importance and Applications 9
- 1.3 Approaches of Text Summarization 10
  - 1.3.1 Extractive Text Summarization 10
  - 1.3.2 Abstractive Text Summarization 10
  - 1.3.3 Example 10
  - 1.3.4 Differences and Use cases 11

2 Dataset **12**
- 2.1 Gold Summaries 12
- 2.2 DeepMind CNN Stories Corpus 12
  - 2.2.1 State of Art Summarization Scores 12

3 Methodology **14**
- 3.1 Testing Corpus Generation 14
  - 3.1.1 Bogus Article Cleaning 15
  - 3.1.2 Exploratory Data Analysis 16
- 3.2 Preprocessing 19
  - 3.2.1 Tokenization 19
  - 3.2.2 Removing Punctuations 19
  - 3.2.3 Removing Stopwords 19
  - 3.2.4 Stemming vs Lemmatization 20
- 3.3 Extractive Text Summarization 20
  - 3.3.1 Introduction 20
  - 3.3.2 Frequency Based Approach 20
  - 3.3.3 Feature based approach 22
  - 3.3.4 Graph based approach 27
  - 3.3.5 Semantic based approach - LSA 29
- 3.4 Abstractive Text Summarization 30
  - 3.4.1 Introduction 30
  - 3.4.2 Encoder-Decoder Model 31
  - 3.4.3 Training the Model 32
  - 3.4.4 SimpleT5 Model 32
  - 3.4.5 Pegasus Model 34

4 Evaluation metric and Result **35**
4.1 Evaluation Metric 35
4.1.1 Precision, Recall, and F-Scores 35
4.1.2 ROUGE - N 35
4.2 Results 36
4.2.1 Extractive Text Summarization 36

5 Frontend/GUI for User Inputs **40**
5.1 Features 40
5.2 Screenshots 40

6 Errors and Future Work **43**
6.1 Errors and Caveats 43
6.2 Future Work 43

7 Conclusion **44**

References **45**

List of Figures

1.1 Original Text 10
1.2 Extractive Summarized Text 11
1.3 Abstractive Summarized Text 11

2.1 Raw Corpus 13
2.2 Raw Dataset Statistics 13
2.3 Dataset Statistics After Cleaning 13

3.1 Example story present in the Dataset 14
3.2 Conversations Patterns to remove from dataset 15
3.3 Histogram of Characters 16
3.4 Histogram of Words 16
3.5 Histogram of Sentences 17
3.6 Top most frequent words 17
3.7 Top most frequent stemed words 18
3.8 Top most frequent lemmatized words 18
3.9 Counts of POS Tags 19
3.10 Implementation - Term Frequency Based Summarization 22
3.11 Implementation - Sentence Scoring using Term Frequency 22
3.12 Implementation - Feature Term Frequeny 23
3.13 Implementation - Feature Title Word Similarity 24
3.14 Implementation - Feature Sentence Length 24
3.15 Implementation - Feature Sentence Position 24
3.16 Implementation - Feature Numerical Data 25
3.17 Implementation - Feature No. of ProperNouns 25
3.18 Implementation - Feature No. of Pronouns 25
3.19 Implementation - Feature Word Co-occurance 26
3.20 Implementation - Defuzzification of Similarity Matrix 26
3.21 Implementation - Feature Thematic Words 26
3.22 Implementation - Feature Cue Phrases 26
3.23 Implementation - Glove Word Embeddings 27
3.24 Implementation - Vectorization and Sentence Similarity 28
3.25 Implementation - Similarity Matrix 28
3.26 Implementation -TextRank Algorithm 28
3.27 Implementation - LSA Sentence Normalization 30
3.28 Implementation - SVD Calculation 30
3.29 Implementation - Main LSA 30
3.30 Code Snippet - Simple T5 Model 34

4.1 Code for ROUGE metrics 36
4.2 Results of Term Frequency Method 36
4.3 F1 Score Graph for Term Frequency Method 37

4.4 Term Frequency - Difference of sentences in Candidate and Reference Summary 37
4.5 F1 Score Graph for Feature Scoring Method 38
4.6 Feature Scoring - Difference of sentences in Candidate and Reference Summary 38

5.1 GUI Screenshot 40
5.2 GUI Screenshot 41
5.3 GUI Screenshot 42

7.1 Benchmark Studies 44

List of Tables

2.1 Benchmarks of academic studies using the DeepMind CNN Dataset 12

4.1 Results - Extractive Text Summarization on 1000 articles 39
4.2 F1 Score - Extractive Text Summarization on random article 39
4.3 Result for Encoder-Decoder Model 39
4.4 Result for Simple T5 Model 39
4.5 Result for Pegasus Model 39

5.1 Extractive Text Summarization on user input article 42

Introduction

1.1 Why Text Summarization?

The topic of Text Summarization remains a subject of ongoing research in the domains of Information Retrieval as well as Natural Language Processing. Usage of Text Summarization is on the rise in the commercial sector, through applications in sectors such as Telephone communication, text database data extraction, internet-based knowledge retrieval, and language processing. Various methodologies differ based on the way associated with their the corresponding problem formulations. The process of automatically generating a condensed version of a given text is considered to be a pivotal aspect in the effective management of information. This addresses the problem of identifying the most crucial segments of a given text. The production of high-quality summaries requires the utilisation of sophisticated natural language processing techniques.

1.2 Importance and Applications

In light of cutting-edge technological advancements, data has become the equivalent of oil in the previous century. The modern day society is moulded by the accumulation and distribution of extensive amounts of data.
According to the International Data Corporation (IDC), there is a projected increase in the global circulation of digital data from a whopping 4.4 zettabytes in 2013 to 180 zettabytes by 2025. That's a lot of information!
Given the vast amount of data present in the digital domain, it is imperative to devise machine learning procedures that can effectively condense lengthy texts and generate precise summaries that effectively express the desired messages.

- Summarising reduces the amount of time required to read.
- In research, summaries facilitate the selection of relevant documents.
- Automatic summarization improves indexing efficiency.
- Machine-generated summarization algorithms are less prejudiced than human summarizers.
- In question-answering systems, personalised summaries are advantageous because they provide customised information.
- The utilisation of autonomous or partially autonomous summarization systems facilitates the capacity of commercial abstract services to handle a larger quantity of textual data.

Furthermore, the implementation of text summarization results in a reduction of the time required for reading, an acceleration of the research process, and an augmentation of the amount of information that can be accommodated within a specified space..

1.3 Approaches of Text Summarization

1.3.1 Extractive Text Summarization

The technique of extractive text summarization involves evaluating the significance of each sentence within a given text, with consideration given to its level of comprehension and relevance to the overall content. The outcome of this process is the identification and presentation of the most significant sentences.

This approach does not produce new words and phrases or idiomatic expressions; rather, it merely showcases pre-existing sentences.

1.3.2 Abstractive Text Summarization

The technique of abstractive text summarization entails the act of rephrasing and briefly summarising certain segments of the primary text. The utilisation of abstraction in deep learning-based text summarization can overcome the grammatical inconsistencies associated with the extractive approach.
Similar to human beings, abstractive text summarization algorithms produce new phrases and sentences that communicate the essential information from the source text.
Therefore, the concept of abstraction is deemed to be more advantageous than that of extraction. The development of text summarization algorithms required for abstraction is comparatively challenging, hence the prevalent usage of extraction.

1.3.3 Example

Original text:[1]

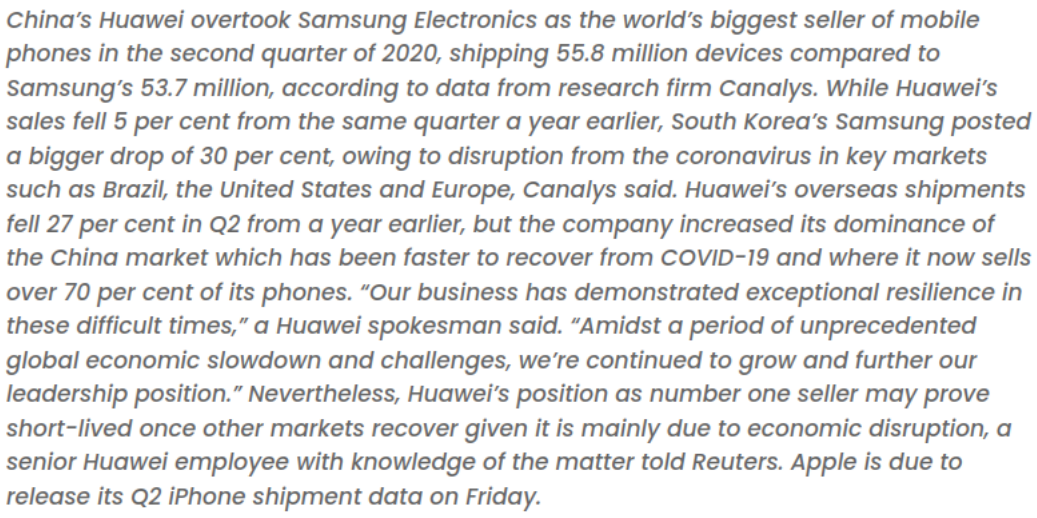
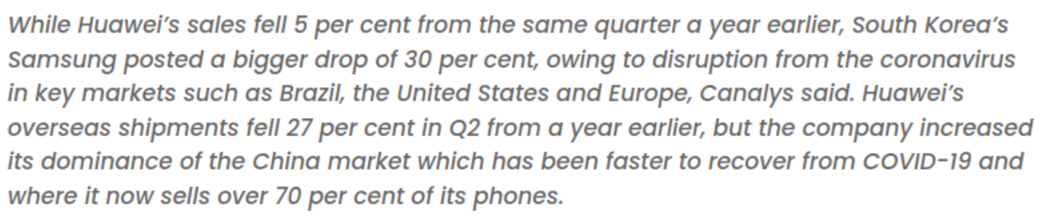

China's Huawei overtook Samsung Electronics as the world's biggest seller of mobile phones in the second quarter of 2020, shipping 55.8 million devices compared to Samsung's 53.7 million, according to data from research firm Canalys. While Huawei's sales fell 5 per cent from the same quarter a year earlier, South Korea's Samsung posted a bigger drop of 30 per cent, owing to disruption from the coronavirus in key markets such as Brazil, the United States and Europe, Canalys said. Huawei's overseas shipments fell 27 per cent in Q2 from a year earlier, but the company increased its dominance of the China market which has been faster to recover from COVID-19 and where it now sells over 70 per cent of its phones. "Our business has demonstrated exceptional resilience in these difficult times," a Huawei spokesman said. "Amidst a period of unprecedented global economic slowdown and challenges, we're continued to grow and further our leadership position." Nevertheless, Huawei's position as number one seller may prove short-lived once other markets recover given it is mainly due to economic disruption, a senior Huawei employee with knowledge of the matter told Reuters. Apple is due to release its Q2 iPhone shipment data on Friday.

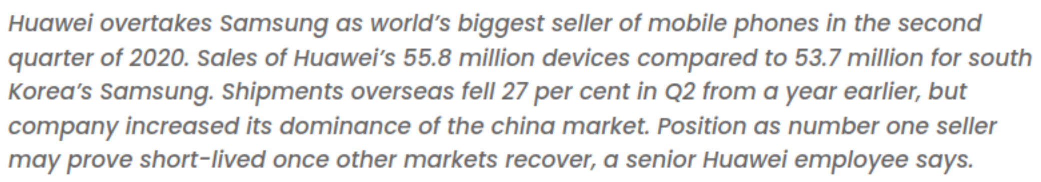
Figure 1.1: Original Text

[1]https://turbolab.in/types-of-text-summarization-extractive-and-abstractive-summarization-basics/



While Huawei's sales fell 5 per cent from the same quarter a year earlier, South Korea's Samsung posted a bigger drop of 30 per cent, owing to disruption from the coronavirus in key markets such as Brazil, the United States and Europe, Canalys said. Huawei's overseas shipments fell 27 per cent in Q2 from a year earlier, but the company increased its dominance of the China market which has been faster to recover from COVID-19 and where it now sells over 70 per cent of its phones.

Figure 1.2: Extractive Summarized Text



Huawei overtakes Samsung as world's biggest seller of mobile phones in the second quarter of 2020. Sales of Huawei's 55.8 million devices compared to 53.7 million for south Korea's Samsung. Shipments overseas fell 27 per cent in Q2 from a year earlier, but company increased its dominance of the china market. Position as number one seller may prove short-lived once other markets recover, a senior Huawei employee says.

Figure 1.3: Abstractive Summarized Text

1.3.4 Differences and Use cases

The process of extractive summarization entails the identification of significant segments from a given text and the subsequent generation of these segments in an unaltered form, resulting in a subset of sentences derived from the original text.

Abstractive summarization relies on natural language processing techniques to comprehend and extract the salient features of a given text, thereby producing a summary that is more amenable to human interpretation.

To provide an analogy, the process of extractive summarization can be compared to the function of a highlighter, whereas abstractive summarization can be compared to that of a pen.
Although both methods have their respective advantages and suitable applications, abstractive summarization tends to yield superior outcomes in dialogues characterised by complex and disorganised information. whereas extractive summarization is better suited for minimising texts that contain a significant amount of factual information.

Dataset

2.1 Gold Summaries

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. The evaluation metric The evaluation metric ROUGE quantifies the degree of similarity between the n-grams present in the predicted summary and those found in the gold standard summary.

2.2 DeepMind CNN Stories Corpus

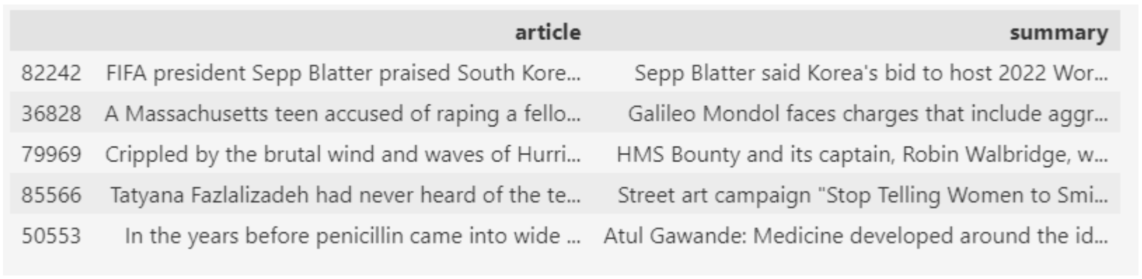
The CNN news stories dataset contains 92,579 news articles and their respective summaries termed as highlights in form of seperate paragraphs with @highlight tag.
The dataset was made available by from New York University and can be found here.

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

2.2.1 State of Art Summarization Scores

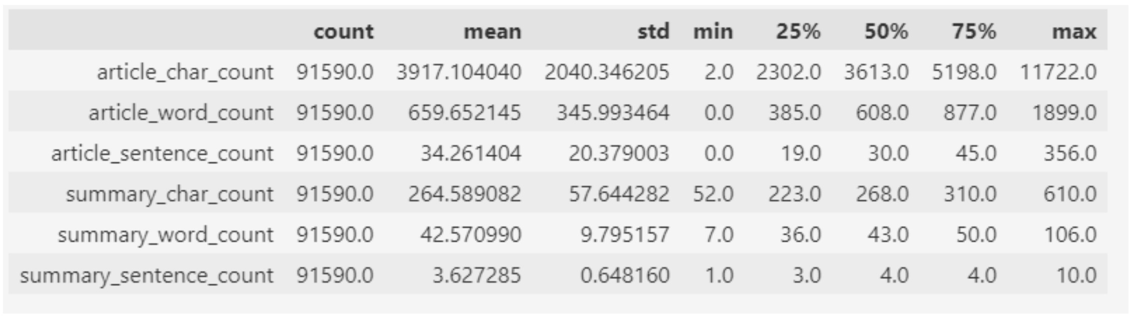
Benchmarks of academic studies using the CNN dataset			
Study	**Date**	**Authors**	**F1 score**
Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond	August/2016	Ramesh Nallapati	35.5
Get To The Point: Summarization with Pointer-Generator Networks	April/2017	Abigail See	39 -40
A Hierarchical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

Table 2.1: Benchmarks of academic studies using the DeepMind CNN Dataset



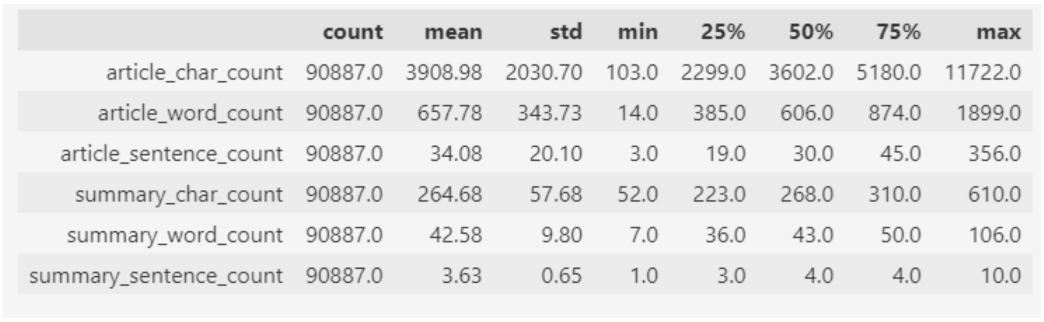
	article	summary
82242	FIFA president Sepp Blatter praised South Kore...	Sepp Blatter said Korea's bid to host 2022 Wor...
36828	A Massachusetts teen accused of raping a fello...	Galileo Mondol faces charges that include aggr...
79969	Crippled by the brutal wind and waves of Hurri...	HMS Bounty and its 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



	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 2.3: Dataset Statistics After Cleaning

Methodology

3.1 Testing Corpus Generation

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.

```
(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
```

Figure 3.1: Example story present in the Dataset

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.

3.1.1 Bogus Article 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?
```

Figure 3.2: Conversations Patterns to remove from dataset

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.

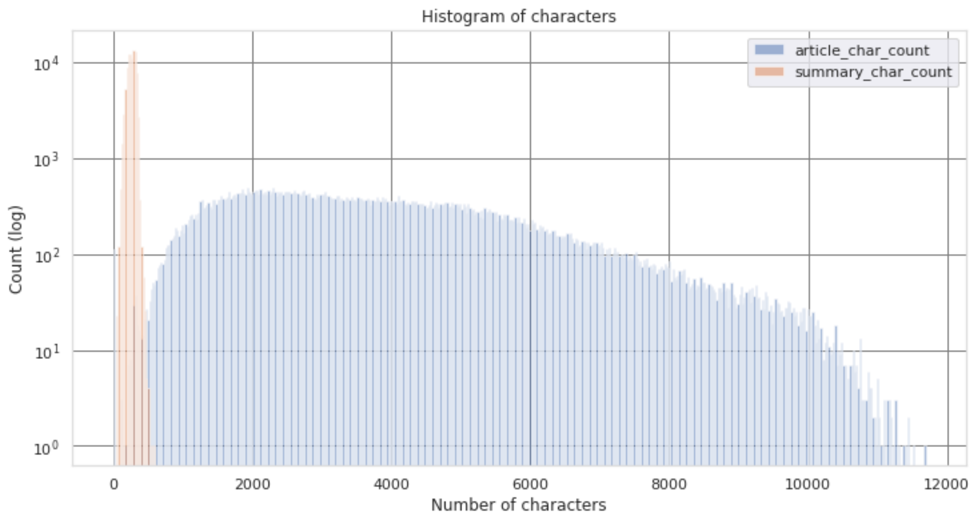


Figure 3.3: Histogram of Characters

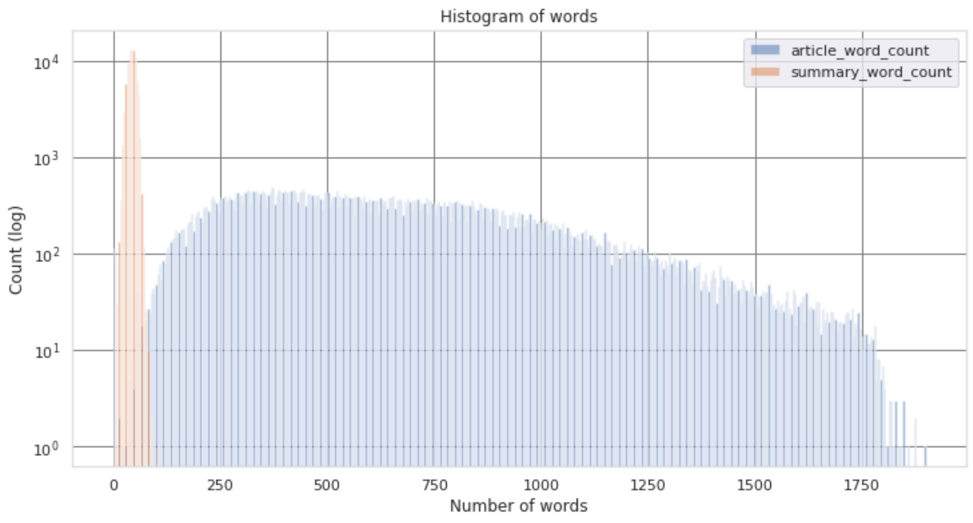


Figure 3.4: Histogram of Words

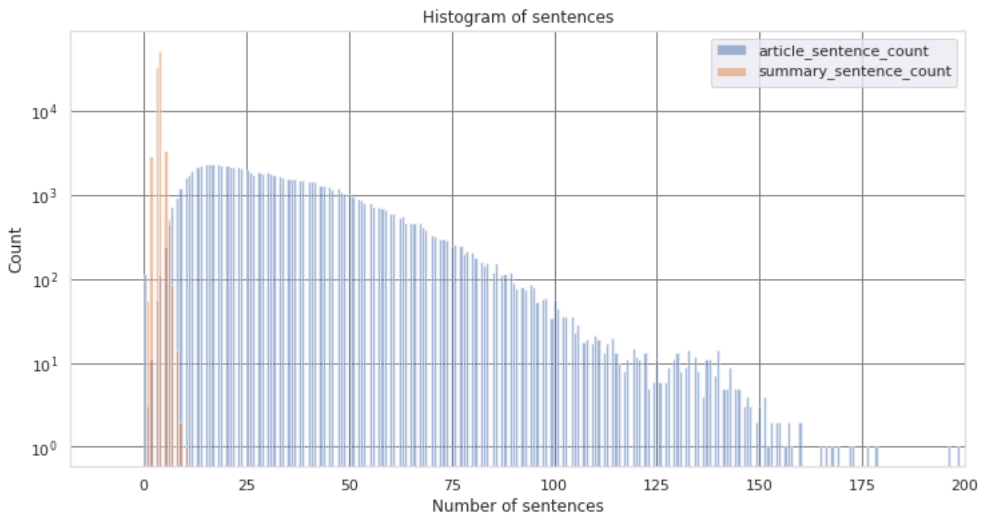


Figure 3.5: Histogram of Sentences

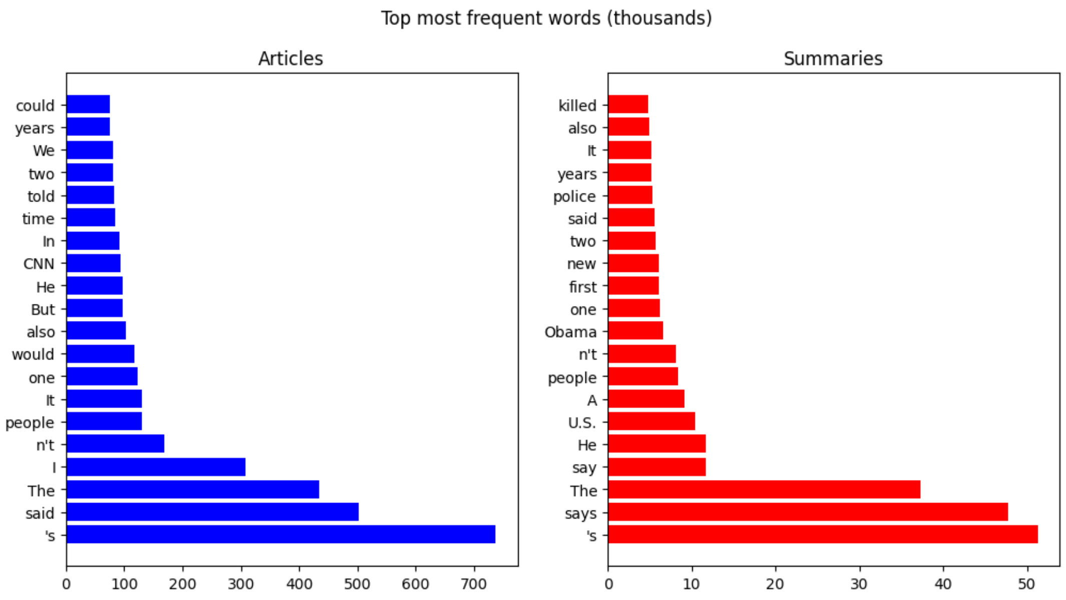


Figure 3.6: Top most frequent words

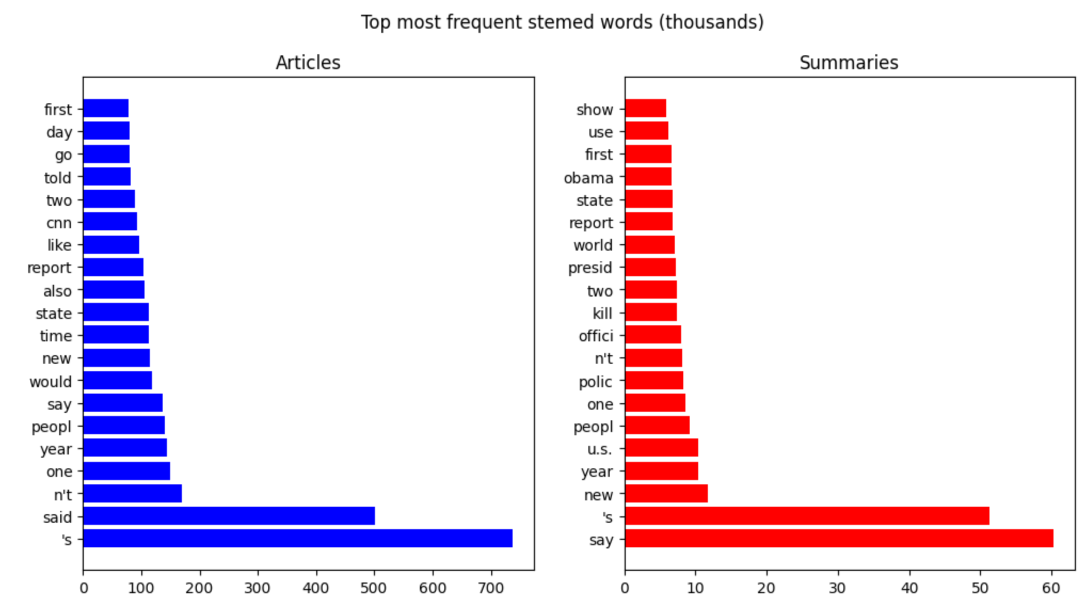


Figure 3.7: Top most frequent stemed words

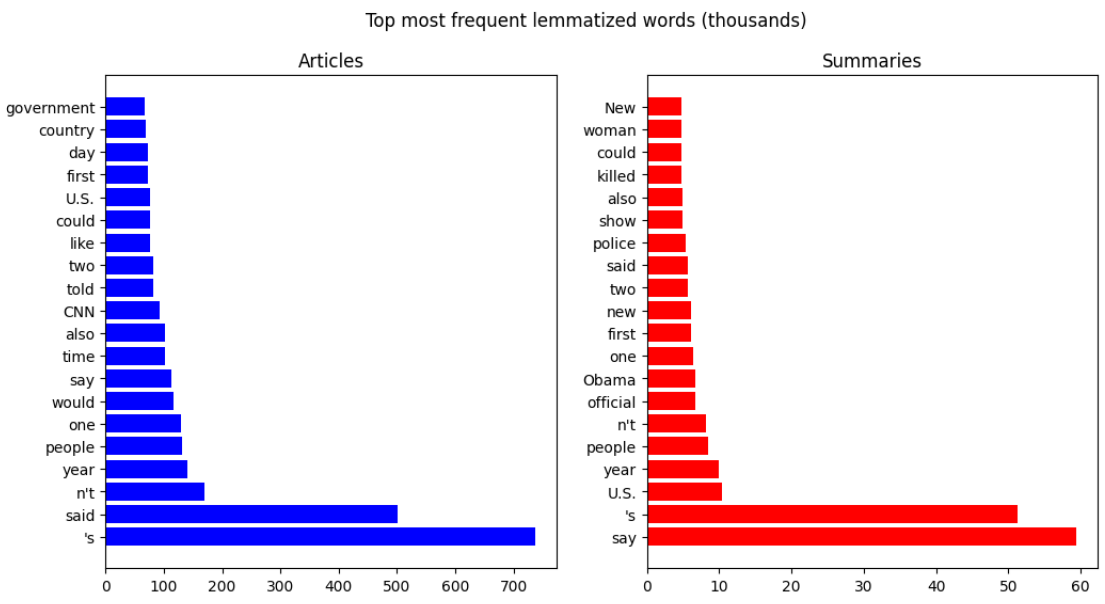


Figure 3.8: Top most frequent lemmatized words

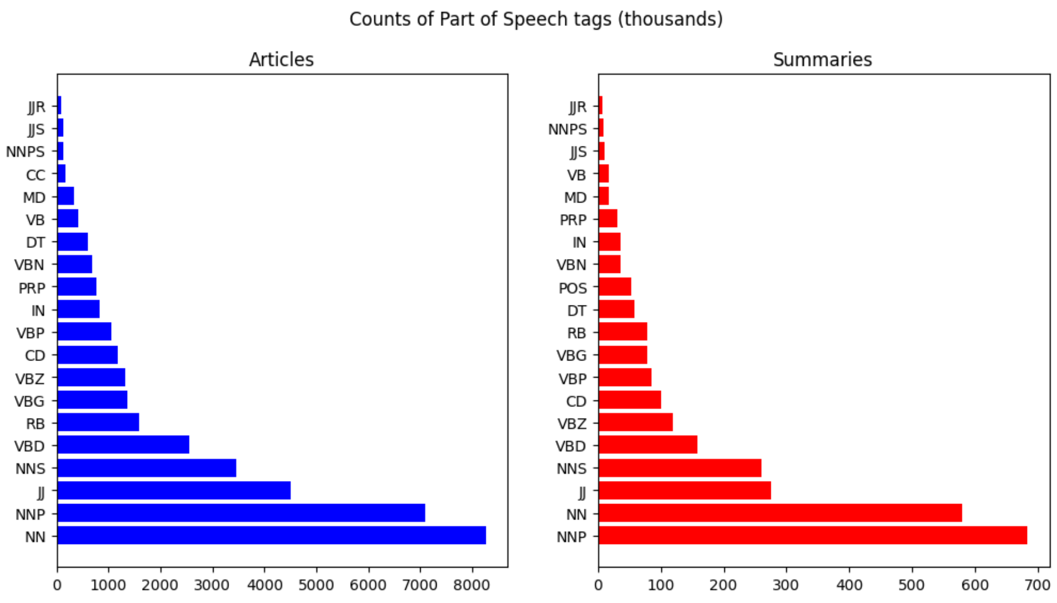


Figure 3.9: Counts of POS Tags

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.
The decision to not convert all text to lowercase is based on the potential issues that may arise, such as the conversion of "U.S" 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 Removing Punctuations

Punctuation characters are obsolete symbols present in our corpus documents.We are just going to remove these -

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

3.2.3 Removing Stopwords

Stop words refer to frequently appearing words that do not contribute any significant meaning to the document. The nltk library can be utilised to iterate through all words and eliminate stop words.

3.2.4 Stemming vs Lemmatization

This stage represents the conclusive and pivotal step of the preprocessing phase.Stemming is the process of reducing words to their stem, while lemmatization involves reducing words to their root form.
The Porter-Stemmer from the nltk library was utilised for the purpose of stemming. Occasionally, the output of the stemmer may not possess semantic meaning, yet it will be recognised as a solitary token.

Lemmatisation is a morphological analysis technique that involves reducing a given word to its base or root form, which is synonymous with the original word. In contrast to stemming, lemmatization ensures that the resulting word is a valid entry in a dictionary. For Lemmatisation we have used nltk library's WordNetLemmatizer

Stemming - need not be a dictionary word
Lemmatization - reduced to a root synonym and will be a dictionary word

3.3 Extractive Text Summarization

3.3.1 Introduction

Extractive Text summarization is the process of generating significant summaries from large documents by preserving only the most relevant sentences from the text and the sentences are not changed at all.

Sometimes extractive summaries perform better compared to their abstractive counterparts. This is due to the fact that abstractive methods exploit the concept of semantics and infer new information that may or may not make sense, in addition to being more difficult than the extraction for extractive-based approaches.

Methods based on Extractive Summarization include a series of steps:

- 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 fundamental method for summarising articles is the frequency-based approach.The utilisation of frequency analysis allows for the identification of relevant and non-relevant elements within a given article through the assessment of word significance. Typically, the topic conveyed in an article is represented by the words that occur most frequently.

Word Probability Method

The Word Probability Score is determined by dividing the frequency of a given word in a sentence by the total number of words in that sentence.

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

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

Upon conducting a word frequency analysis of the document,the frequency of each individual word was determined. This methodology involves assigning a score to each word in a given sentence, which will then be summed to obtain the overall Word Probability Score for the sentence.
In order to prevent long sentences from being unfairly favoured over shorter ones, the score assigned to each sentence is normalised by dividing it by the number of words contained within that sentence.
Now we select the required number of sentences by sorting this score.

Term Frequency – Inverse Document Frequency (TF-IDF)

The TF-IDF methodology prioritises uniqueness in generating the feature vector, thereby addressing the challenge of rarity of a given term.

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

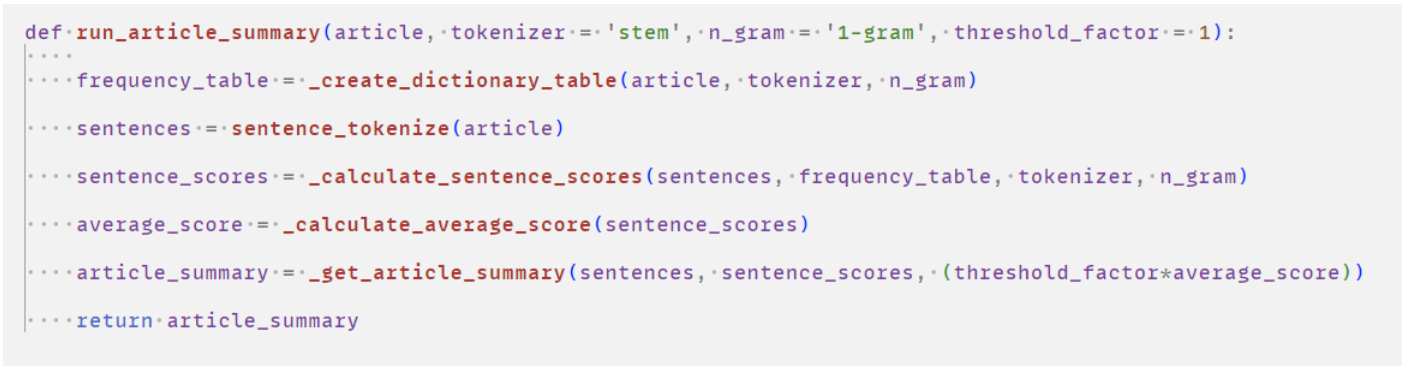
$$TF(t) = \frac{Frequency(t)}{Total\ number\ of\ terms\ in\ the\ document}$$

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

The TF-IDF method mitigates the influence of frequently appearing words that possess a higher frequency in the corpus by associating them with their respective occurrence count in the set of documents.

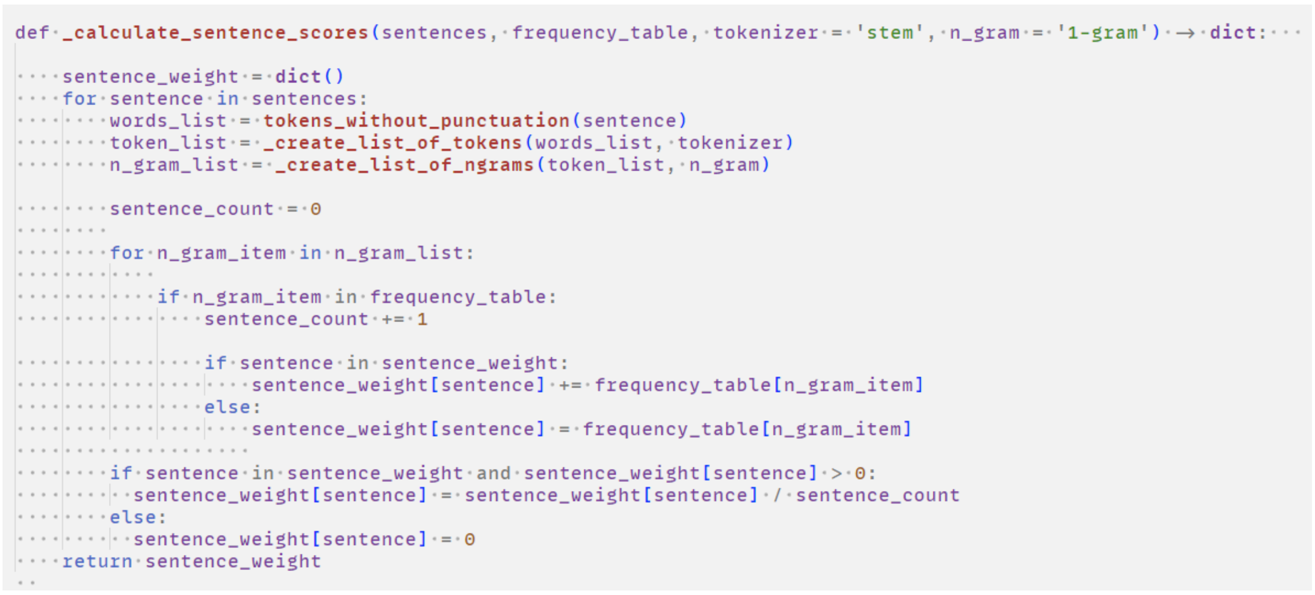
Steps involved in TF-IDF approach :

- Calculate Term Frequency for the words present in each sentence
- Calculate Document Frequency for the words present 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.



```python
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: Implementation - Term Frequency Based Summarization



```python
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: Implementation - Sentence Scoring using Term Frequency

3.3.3 Feature based approach

Various scholars have put forth a range of methods for evaluating sentence scores on a sentence and word level.

The calculation of the sentence score involves performing a simple linear combination of the values of the features and weights associated with the feature.

$$Score = \sum w_i * F_i$$

Following are the features that we have implemented for scoring :

1. Term Frequency F1

$$F1 = \frac{Frequency(term)}{Number\ of\ terms\ present\ in\ the\ article}$$

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

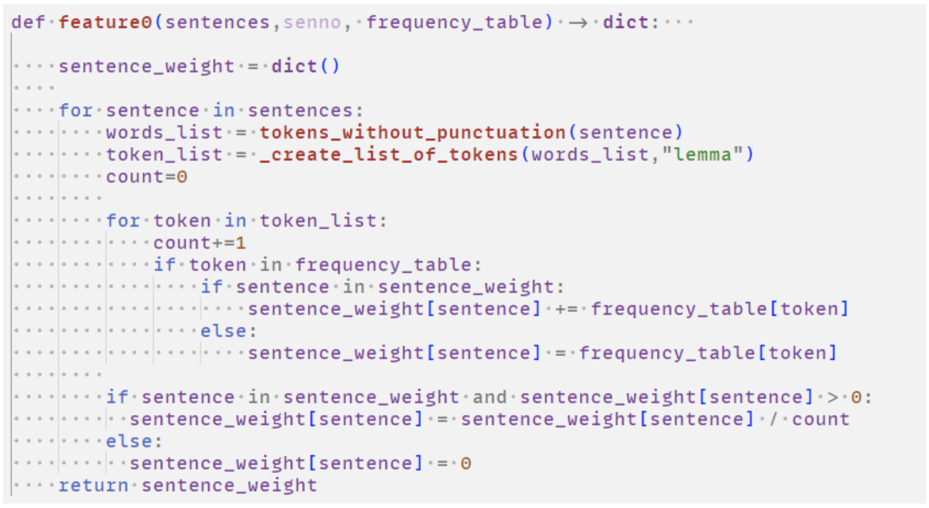
6. Proper Noun F6
$$F6 = \frac{No\ of\ Proper\ Nouns\ present\ in\ S_i}{|S_i|}$$

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

8. Numerical Value in Sentence F8
$$F8 = \frac{No\ of\ Numerical\ data\ present\ in\ S_i}{|S_i|}$$

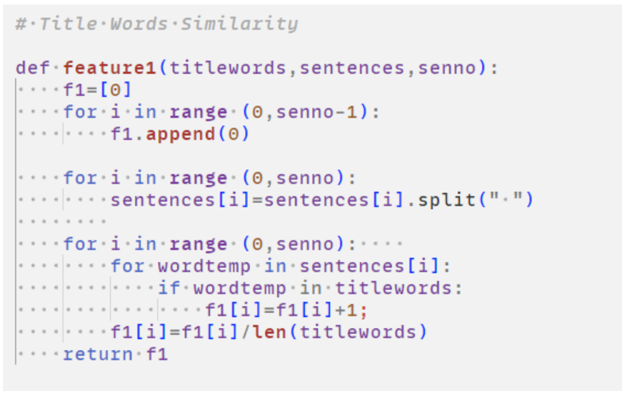
9. Thematic Features F9: Thematic words were identified by selecting the most frequent words.
$$F9 = No\ of\ thematic\ words\ in\ S_i$$

10. Word Co-occurrence matrix Defuzzification F10 : There is a possibility that certain terms may co-occur in the same manner and position within sentences. Assigning greater weight to co-occurring words can enhance the prominence of significant information in the ultimate summary.



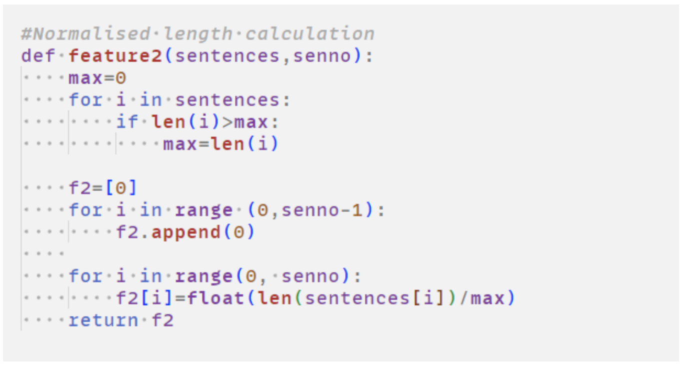
```python
def feature0(sentences,senno, frequency_table) → dict: ···

    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: Implementation - Feature Term Frequeny



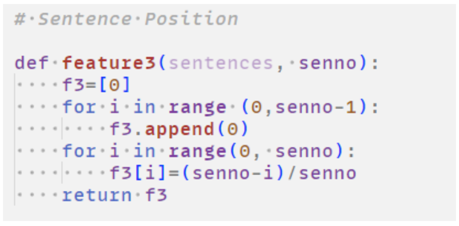
```
# Title Words Similarity

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: Implementation - Feature Title Word Similarity



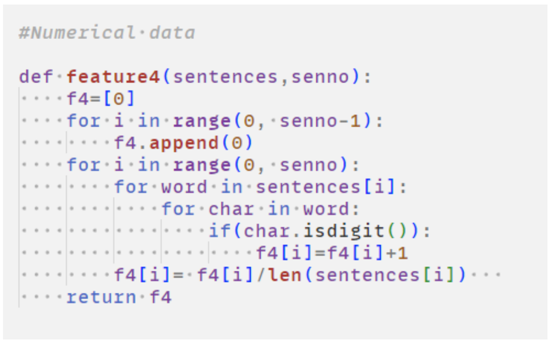
```
#Normalised length calculation
def feature2(sentences,senno):
    max=0
    for i in sentences:
        if len(i)>max:
            max=len(i)

    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: Implementation - Feature Sentence Length



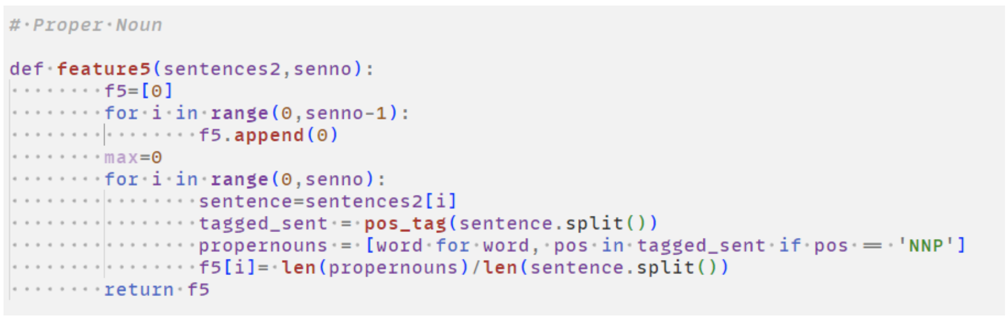
```
# Sentence Position

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: Implementation - Feature Sentence Position



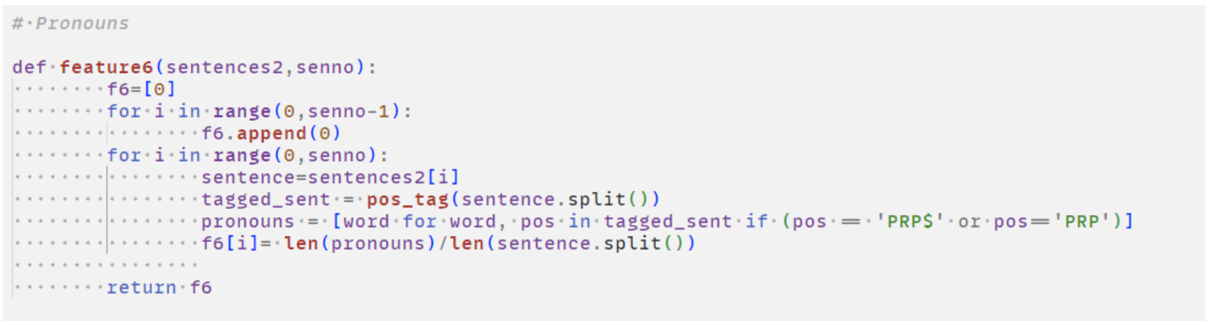
```
#Numerical data

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: Implementation - Feature Numerical Data



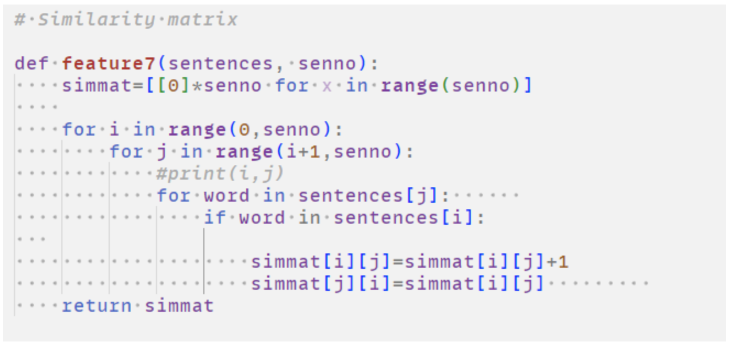
```
# Proper Noun

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: Implementation - Feature No. of ProperNouns



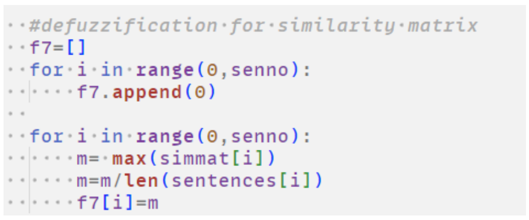
```
# Pronouns

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: Implementation - Feature No. of Pronouns



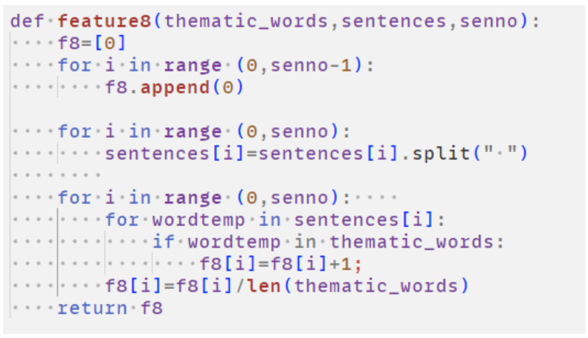
```python
# Similarity matrix

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: Implementation - Feature Word Co-occurance



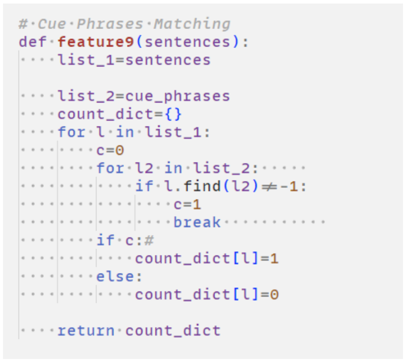
```python
  #defuzzification for similarity matrix
  f7=[]
  for i in range(0,senno):
      f7.append(0)
  
  for i in range(0,senno):
      m= max(simmat[i])
      m=m/len(sentences[i])
      f7[i]=m
```

Figure 3.20: Implementation - Defuzzification of Similarity Matrix



```python
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: Implementation - Feature Thematic Words



```python
# 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: Implementation - 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.

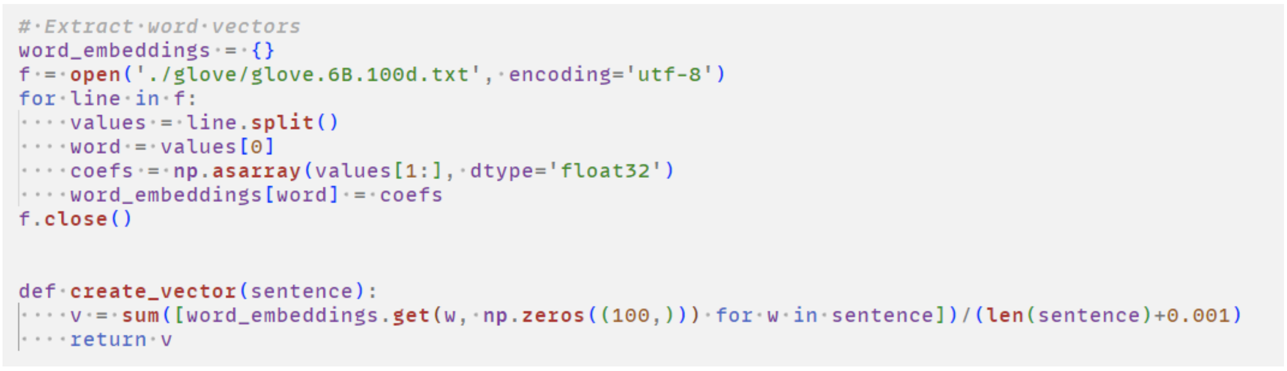
Our focus for text summarisation is to rank the sentences according to their score. This score is calculated using the PageRank Algorithm using the **sentence_similarity_graph** created.
Once the sentences have been ranked and sorted based on their ranking, the summary is generated by including the most significant sentences. This can be achieved through the application of either threshold values or selection of the top n sentences.

PageRank Algorithm at a glance :
The article is laid out in the structure of an undirected graph, wherein the nodes that are incoming and outgoing are regarded as identical. Sentences are regarded as vertices, while edges represent the connections between distinct 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 refer to its degree of similarity with each sentence present 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

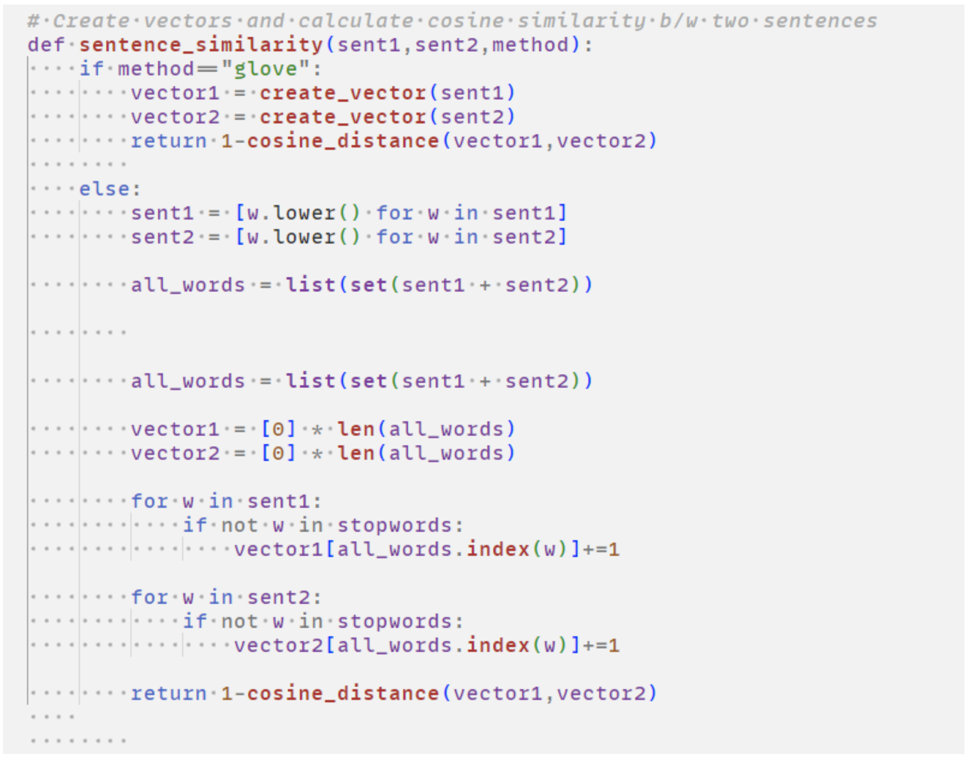
To calculate sentence similarity , we will be vectorizing both the sentences and then calculation the **cosine distance between both vectors as a similarity score**.

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



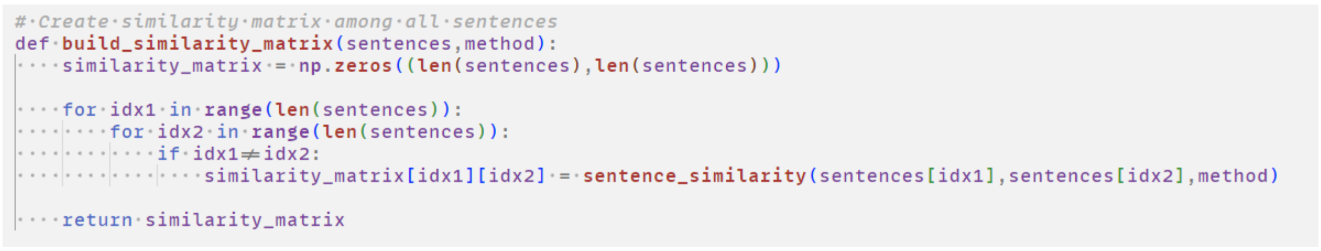
```
# Extract word vectors
word_embeddings = {}
f = open('./glove/glove.6B.100d.txt', encoding='utf-8')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    word_embeddings[word] = coefs
f.close()


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: Implementation - Glove Word Embeddings



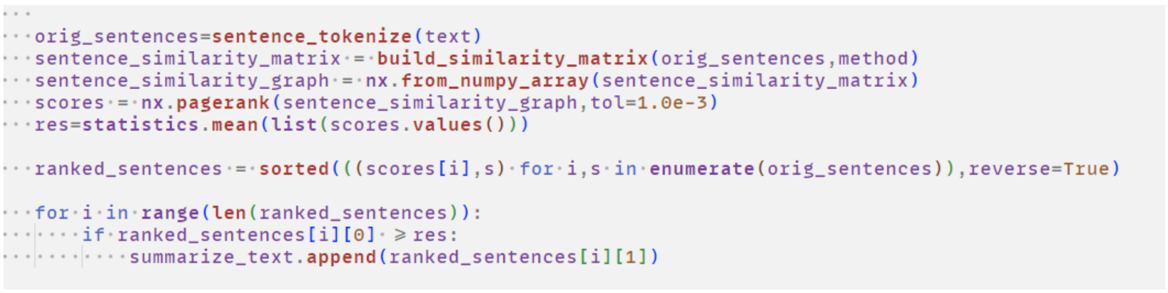
```python
# Create vectors and calculate cosine similarity b/w two sentences
def sentence_similarity(sent1,sent2,method):
    if method=="glove":
        vector1 = create_vector(sent1)
        vector2 = create_vector(sent2)
        return 1-cosine_distance(vector1,vector2)
        
    else:
        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: Implementation - Vectorization and Sentence Similarity



```python
# 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: Implementation - Similarity Matrix



```python
   
   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: Implementation -TextRank Algorithm

3.3.5 Semantic based approach - LSA

The fundamental concept underlying the utilisation of Latent Semantic Analysis (LSA) in the process of text summarization is that words that tend to appear in similar contexts are likewise associated within the same singular unified space.
The Latent Semantic Analysis (LSA) technique is utilised to convert sentence vectors from a term-space that comprises non-orthogonal features to a concept-space that has a lower dimensionality and an orthogonal basis. This is done by performing singular value decomposition of term sentence matrix A. The outcome of the process yields three matrices, namely U, V, and $\sum$:

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

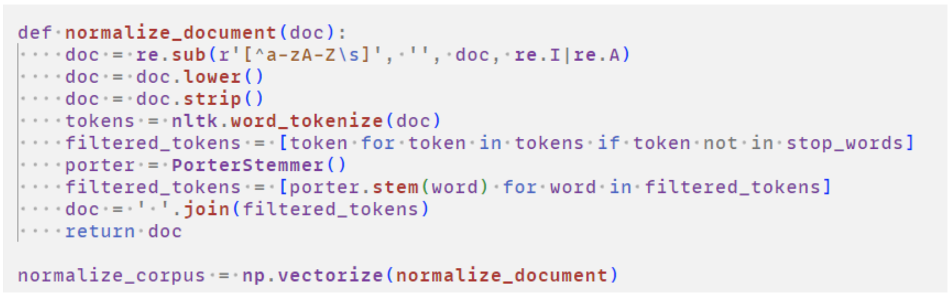
The matrices U and V are characterised as orthogonal matrices, while is a diagonal matrix. The diagonal elements of signify the respective significance of each concept dimension in the basis of the concept space. The vectors that correspond to the terms and sentences in the concept space are respectively represented by the columns of matrix U and the rows of matrix V^T. By selecting the top k concepts based on their eigenvalues, we can obtain the optimal k-rank approximation of matrix A through the least squares method.

Mathmatically, the score of a sentence, if V is the sentence vector and σ is the eigen value of the i^{th} concept in the concept space, is calculated as

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

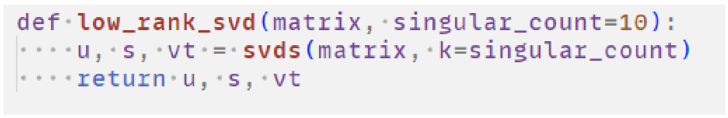
Steps followed :

- 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
- Obtain the k highest singular values S utilising the SVD.
- Utilise thresholding to eliminate singular values. This is a heuristic, and we can choose values according to our preference or take it as mean of all values.
- To obtain the weights of sentences per topic, it is necessary to multiply each column of the term sentence matrix V by its corresponding squared singular value from matrix S.
- To obtain the salience scores for each sentence in a given document, one should first calculate the sum of the weights of the sentences across the topics. Subsequently, the square root of the resulting score should be computed.



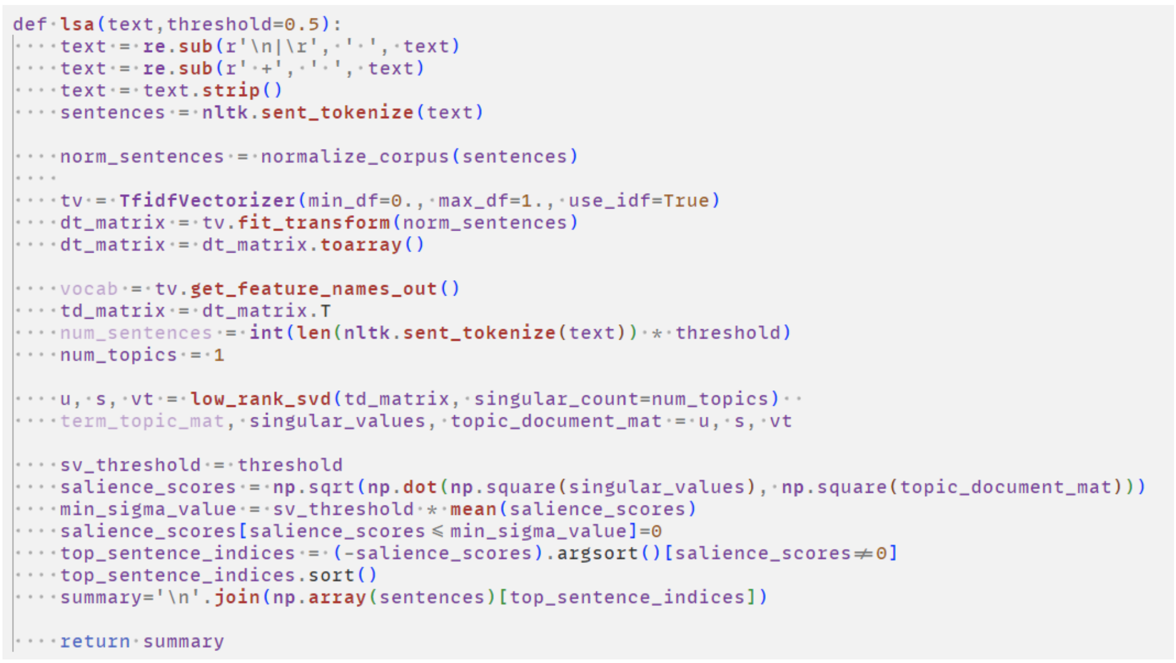
```python
def normalize_document(doc):
    doc = re.sub(r'[^a-zA-Z\s]', '', doc, re.I|re.A)
    doc = doc.lower()
    doc = doc.strip()
    tokens = nltk.word_tokenize(doc)
    filtered_tokens = [token for token in tokens if token not in stop_words]
    porter = PorterStemmer()
    filtered_tokens =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

normalize_corpus = np.vectorize(normalize_document)
```

Figure 3.27: Implementation - LSA Sentence Normalization



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

Figure 3.28: Implementation - SVD Calculation



```python
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: Implementation - Main LSA

3.4 Abstractive Text Summarization

3.4.1 Introduction

Abstractive text summarization is a technique for producing an easily comprehensible and logical summary of a lengthy text that captures key concepts and relevant details. In contrast to extractive summary, which selects and combines sentences from the original text, abstractive summarization involves the creation of new phrases that have the same meaning as the original text. Natural language generation (NLG) algorithms must be able to comprehend the content of the source text and produce grammatically correct and flowing summaries for abstractive summarization to be successful. Computational linguistics methodologies, neural networks, and machine learning algorithms are examples of these methods. To provide similar summaries, abstractive summarization requires the model to grasp language and context.

The general steps followed by abstractive text summarization:

- Preprocessing stages remove stop words, stemming/lemmatizing, and other text cleaning activities.
- Sentence Splitting: This technique is used to divide material into sentences.
- Text Representation: To be processed by the model, text is represented in a machine-readable format, such as vectors or matrices.
- Model Training: To learn linguistic patterns and structures, deep learning models are trained on a huge corpus of text using natural language processing techniques.
- Summarization: To construct a summary, an abstractive summarization model is used to the input text. Encoding the text into a fixed-length form and decoding it to generate the summary is involved.
- Evaluation: To guarantee that the generated summary is cohesive, relevant, and correct, it is reviewed using metrics such as ROUGE, BLEU, and others.
- Refinement: Based on the evaluation findings, the model is refined and fine-tuned to increase the quality of generated summaries.

3.4.2 Encoder-Decoder Model

Encoder: The encoder is made up of N = 6 identical layers stacked together. Each layer is composed of two sublayers. The first is a multi-head self-attention mechanism, whereas the second is a simple feed-forward network that is totally connected positionally. A residual connection is used around each of the two sub-layers, followed by layer normalisation . In other words, the output of each sub-layer is LayerNorm(x + Sublayer(x)), where Sublayer(x) is the function that the sub-layer itself implements. To accommodate these residual connections, all model sub-layers and embedding layers create outputs with size dmodel = 512.

The decoder is also made up of a stack of N = 6 identical layers. The decoder adds a third sub-layer to each encoder layer, on top of the two already there, to execute multi-head attention over the encoder stack's output. We employ residual connections surrounding each sub-layer, similar to the encoder, followed by layer normalisation. We also adjust the self-attention sub-layer in the decoder stack to prevent positions from paying attention to succeeding positions. Because of the masking and the fact that the output embeddings are offset by one place, the predictions for location i can only rely on the known outputs at positions less than i.

Scaled Dot-Product is essential for Transformer attention computation which is given by :- Attention(Q, K, V) = softmax(QKT/ dk)V

Point-wise Feed-Forward Network is used after every sub-layer and is identical.

Multi-Head Attention splits inputs into multiple heads and concatenates output.

Encoder and Decoder Blocks are fundamental units of encoder and decoder.

In the **Actual Encoder and Decoder** Nx Encoder and Decoder interact with inputs and outputs.

Stack intermediate layers in Custom Model class for inference or loss calculation.

3.4.3 Training the Model

We had trained the model with Sparse Categorical Cross Entropy loss as regular Categorical Cross Entropy loss that would require the target as a one-hot encoding of each word in the target sequence which would be a huge array.
Moreover, we mask the loss incurred by the padded tokens to 0 so that they do not contribute to the mean loss.
Following are the hyper-parameter values that we have used while training:-

hyper-params
num_layers = 4
d_model = 128
dff = 512
num_heads = 8
EPOCHS = 20
BUFFER_SIZE = 20000
BATCH_SIZE = 64
Encoder_maxlen = 400
Decoder_maxlen =75
encoder_vocab_size, decoder_vocab_size = 76362, 29661

3.4.4 SimpleT5 Model

In the realm of natural language processing (NLP), the SimpleT5 (Text-To-Text Transfer Transformer) model is a potent transformer-based neural network architecture. We give a thorough explanation of the T5 model and its associated processes in this research report. We talk about T5's influence on the NLP field and examine its fundamental ideas, training methods, and applications. For researchers and professionals interested in comprehending and utilizing the T5 model for diverse NLP tasks, this paper is an invaluable resource.

Introduction

The T5 model, introduced by Colin Raffel et al. in their paper "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," has emerged as a groundbreaking approach in NLP. It addresses the limitations of traditional task-specific models by adopting a text-to-text framework, enabling a unified architecture for a wide range of NLP tasks.

Text-to-Text Framework

The text-to-text framework employed by T5 converts different NLP tasks into a text generation format. Input text is transformed into a text prompt that describes the task, and the model generates the desired text response as the output. This approach provides a unified and consistent way of formulating various NLP tasks.

Transformer Architecture

T5 utilizes the Transformer architecture, which incorporates self-attention mechanisms to capture global dependencies within the input sequence. The self-attention mechanism enables the model to efficiently process and understand the contextual relationships between words, resulting in improved performance on complex NLP tasks.

Training Process

The T5 model undergoes a two-step training process: pretraining and fine-tuning. In the pre-training phase, the model is trained on a large corpus of diverse unlabeled text data, learning general language representations. Fine-tuning follows, where the pretrained model is further trained on task-specific datasets to adapt its knowledge to the downstream tasks

Transfer Learning in T5

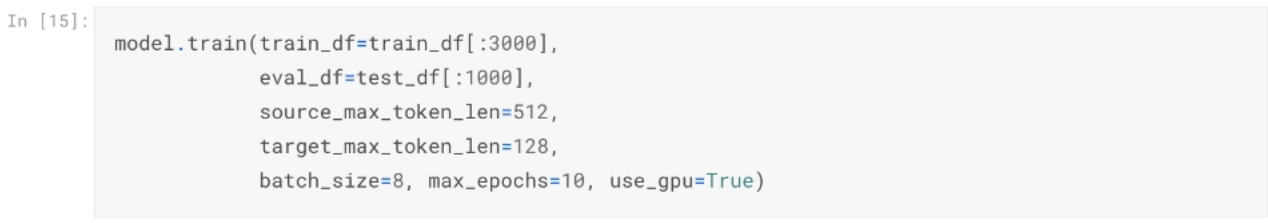
Transfer learning plays a vital role in the effectiveness of T5. By leveraging pretrained language representations, T5 benefits from the rich knowledge learned during pretraining and transfers it to various downstream tasks. This approach significantly reduces the need for task-specific labeled data, improves efficiency, and enables the model to achieve state-of-the-art performance across multiple NLP benchmarks.

By introducing the text-to-text transfer learning paradigm, the T5 model transformed the area of NLP. T5 has established itself as a leader in the NLP research community thanks to its unified design, transfer learning capabilities, and cutting-edge performance on several NLP tasks. This article offers a thorough explanation of the T5 model and its associated procedures, making it an invaluable resource for academics and industry professionals. Here for training our T-5 model we have used DeepMind CNN Stories dataset

Steps involved in model training after preprocessing of dataset

- Loss Function Choice Define a loss function that is suitable for the particular job (for example, mean square error for regression, or cross-entropy loss for classification). The type of job at hand and the intended result will determine which loss function is best.
- Training Cycle Repeat the training data iterations in groups. The expected output may be obtained by running the input through the T5 model. Calculate the difference in value between the expected result and the actual result using the chosen loss function.To update the parameters, backpropagate the gradients through the model using an optimisation technique (such as Adam, SGD, etc.).Repeat the procedure over several epochs while modifying the model's parameters to reduce loss and enhance performance.

- Assessment and Validation: Throughout training, periodically assess how well the model performed on the validation set.Assess the model's quality by computing assessment metrics (such accuracy, F1 score, or ROUGE scores) and making any necessary corrections.To prevent overfitting (excessive optimisation on the training set) and to guarantee generalization to new data, keep an eye on the training process.
- Deployment and Testing: When training is finished, test the final trained model to get objective performance measures. The trained T5 model should be used for inference so that it can produce text summaries and carry out additional operations on unobserved data.These phases provide a broad overview of the T5 model training procedure. Depending on how the work is implemented and its particular needs, the details may change.



```
In [15]:
model.train(train_df=train_df[:3000],
            eval_df=test_df[:1000],
            source_max_token_len=512,
            target_max_token_len=128,
            batch_size=8, max_epochs=10, use_gpu=True)
```

Figure 3.30: Code Snippet - Simple T5 Model

3.4.5 Pegasus Model

PEGASUS (Pre-training with Extracted Gap-sentences for Abstractive Summarization) is a pre-trained sequence-to-sequence model for abstractive text summarization.PEGASUS is based on the Transformer architecture. However, PEGASUS uses a specific pre-training objective that involves generating summaries from randomly selected passages of text, which are called "gap-sentences." The model is then fine-tuned on a specific summarization task using supervised learning.The gap-sentence approach allows PEGASUS to pre-train on large amounts of text data and learn to generate high-quality summaries that capture the essential information in the original text. The model can be fine-tuned on a wide range of summarization tasks, including news articles, scientific papers, and long-form content. The authors (Jingqing Zhang et. al.) hypothesizes that pre-training the model to output important sentences is suitable as it closely resembles what abstractive summarization needs to do. Using a metric called ROUGE1-F1, the authors were able to automate the selection of important sentences and perform pre-training of the model on a large corpus, i.e., 350 million web pages and 1.5 billion news articles.

With the pre-trained model, we can then perform fine-tuning of the model on the actual data which is of a much smaller quantity. In fact, evaluation results on various datasets showed that with just 1,000 training data, the model achieved comparable results to previous SOTA models. This has important practical implications as most of us will not have the resources to collect tens of thousands of document-summary pairs.

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

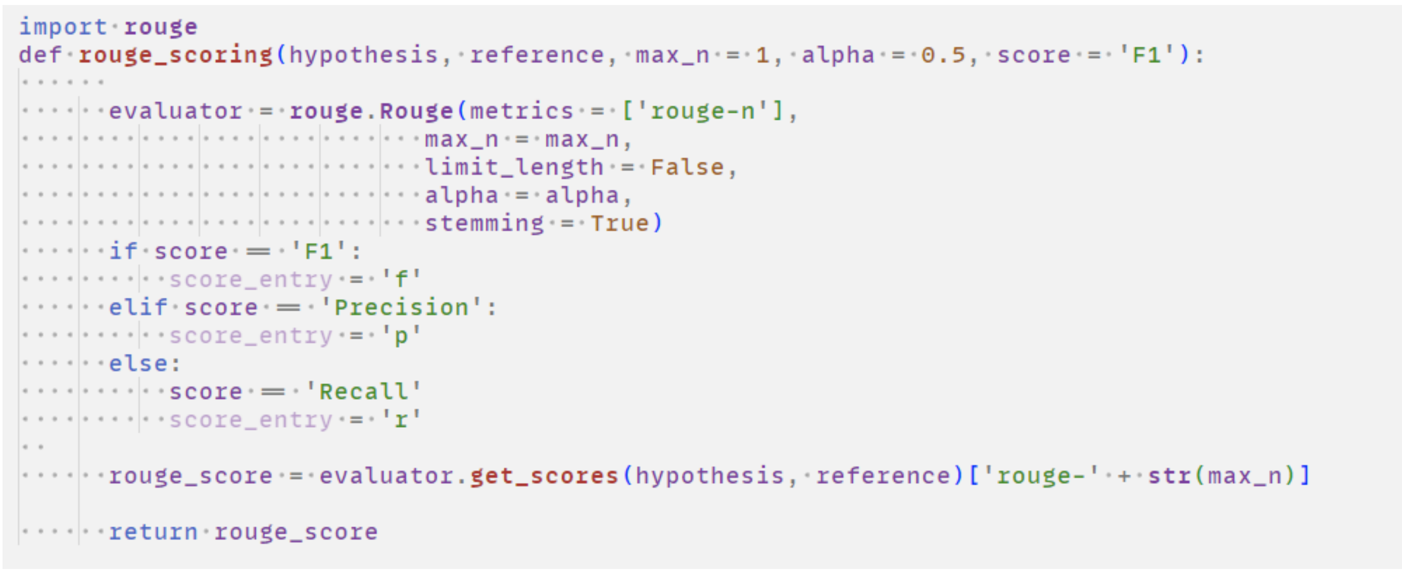
The tool most frequently employed for the automated assessment of generated summaries is Recall-Oriented Understudy for Gisting Evaluation (ROUGE). The ROUGE system comprises a suite of metrics and a software package that is commonly employed in the field of natural language processing (NLP) to assess the performance of automatic summarization and machine translation software. The study involves a comparison between summaries generated by computer and reference summaries created by human beings.

The fundamental concept underlying ROUGE involves quantifying the degree of overlap between candidate or system summaries and reference summaries, through the identification of shared units such as n-grams. The effectiveness of ROUGE in evaluating summary qualities has been established and it demonstrates a strong correlation with human assessments.

There exist multiple ROUGE metrics, which are as follows::

- The ROUGE-1 (R1): This metric utilises a uni-gram approach to evaluate the similarity between a candidate summary and a reference summary. ROUGE-N refers to the degree of n-gram recall that exists between a summary that is being considered as a candidate and the reference summaries.
- ROUGE-L (R-L): It is a metric that relies on identifying the longest common subsequences between a candidate summary and a reference summary.
- ROUGE-S* (R-S*): The evaluation metric known as ROUGE-S* (R-S*) quantifies the degree of overlap between skip-bigrams present in a candidate summary and those found in reference summaries.

- ROUGE-SU* (R-SU*): The ROUGE-SU* (R-SU*) method is an extension of the ROUGE-S* approach that incorporates skip-bigrams and utilises unigrams as the counting unit. The asterisk symbol denotes the quantity of words that are being skipped. As an illustration, the ROUGE-SU4 metric allows the inclusion of bi-grams that are composed of words that are not adjacent to each other, with a limit of four words between the two.



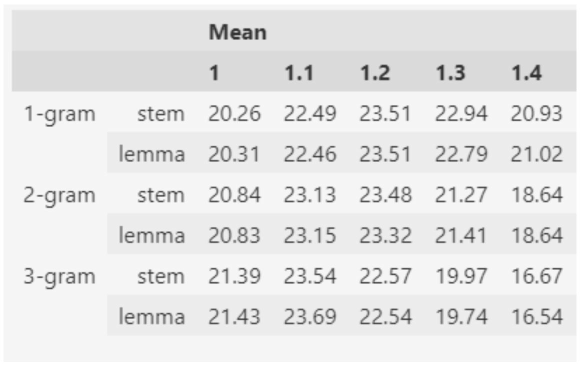
```
import rouge
def rouge_scoring(hypothesis, reference, max_n = 1, alpha = 0.5, score = 'F1'):
      
      evaluator = rouge.Rouge(metrics = ['rouge-n'],
                                max_n = max_n,
                                limit_length = False,
                                alpha = alpha,
                                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

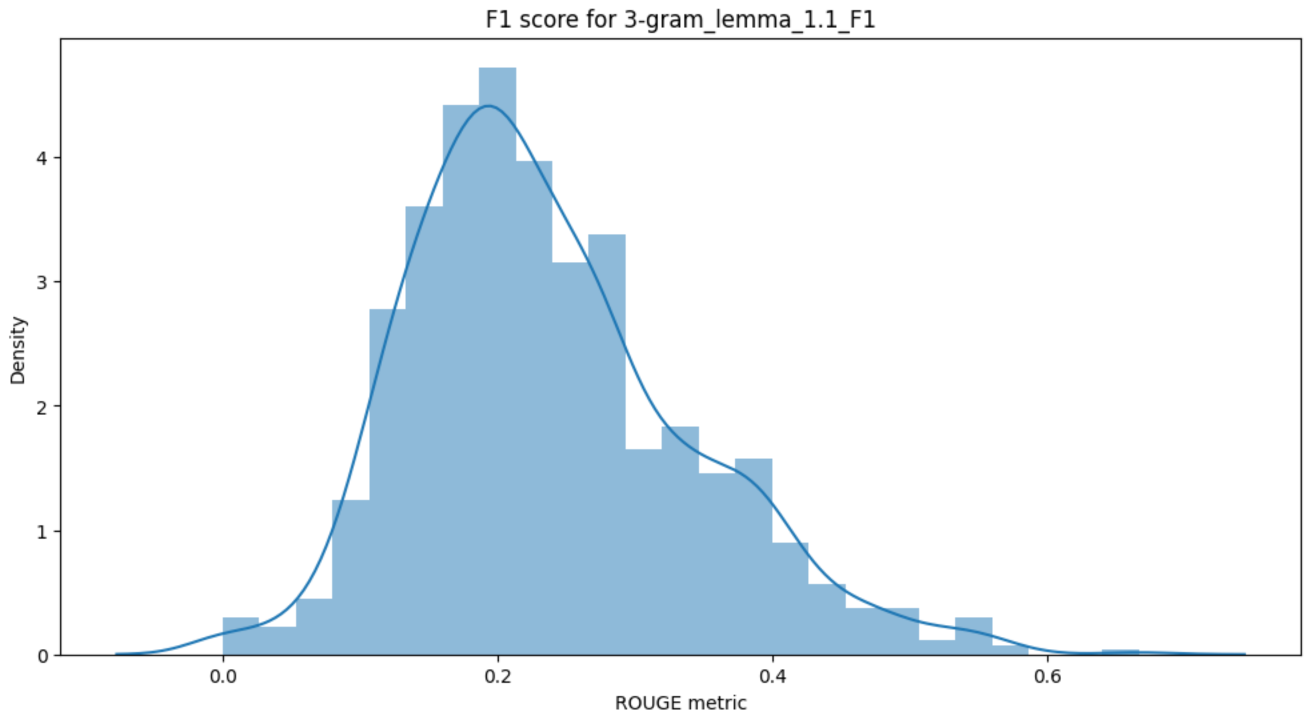


Figure 4.3: F1 Score Graph for Term Frequency Method

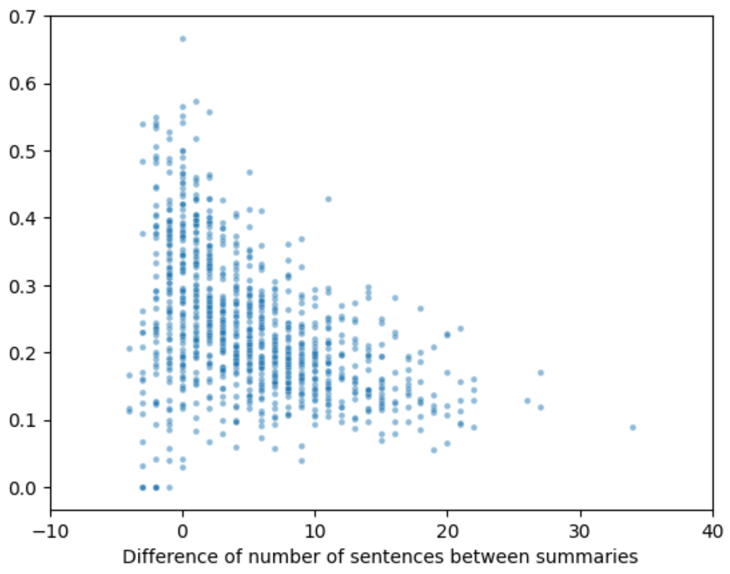


Figure 4.4: Term Frequency - Difference of sentences in Candidate and Reference Summary

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.

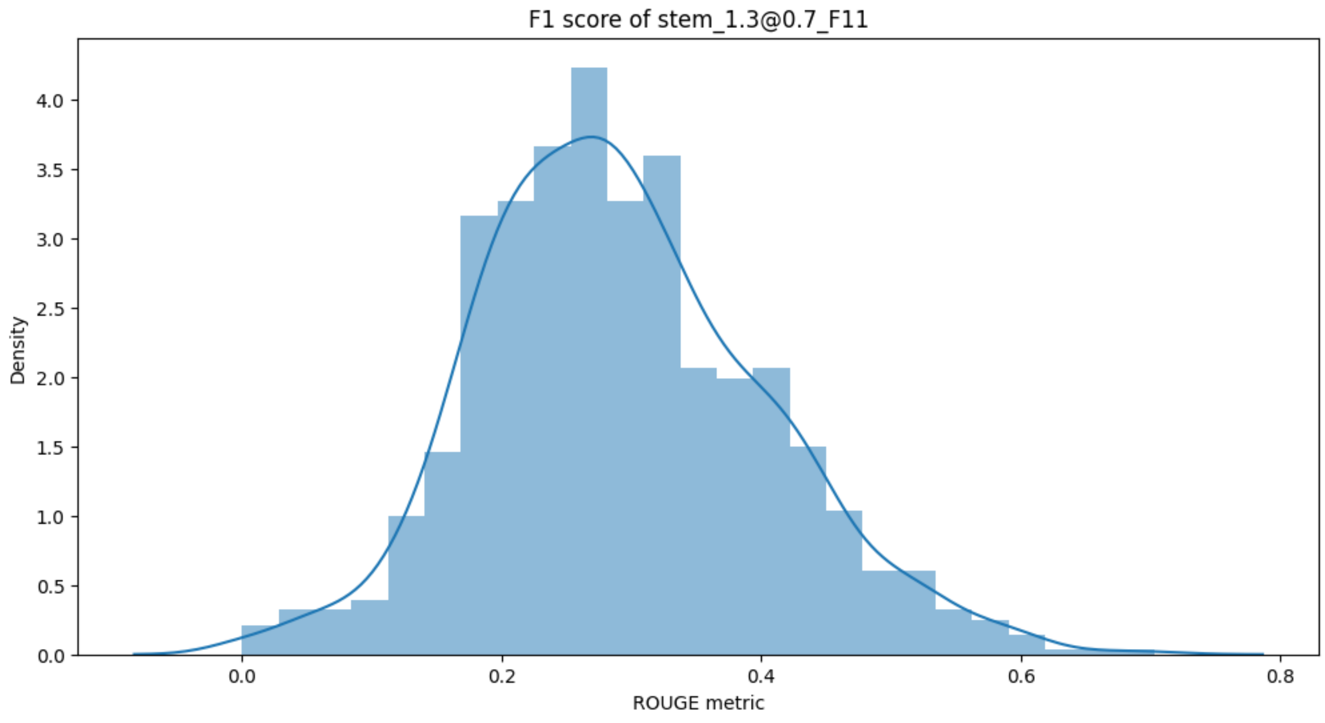


Figure 4.5: F1 Score Graph for Feature Scoring Method

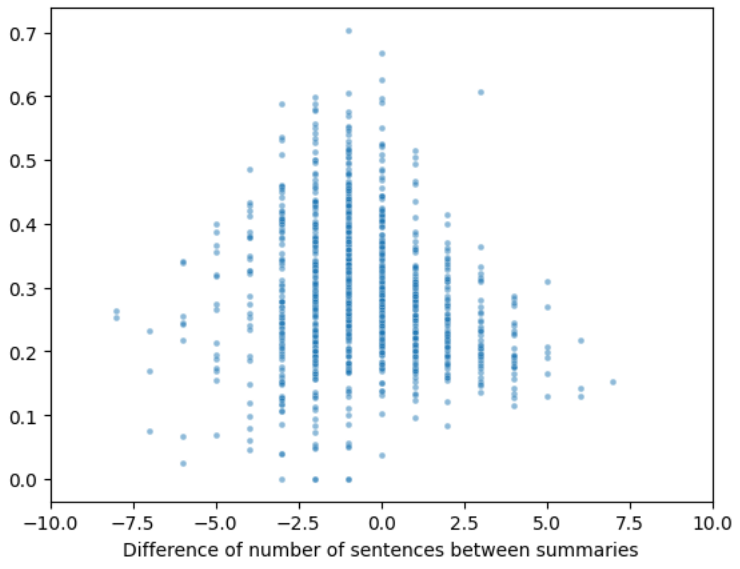


Figure 4.6: Feature Scoring - Difference of sentences in Candidate and Reference Summary

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

	ROUGE-1	ROUGE-2	ROUGE-3
F1	0.111	0.111	0.111
Precision	0.125	0.125	0.125
Recall	0.099	0.099	0.099

Table 4.3: Result for Encoder-Decoder Model

Index	F1 Score
3020	0.88
3022	0.30
3024	0.20
3026	0.30
3028	0.16

Table 4.4: Result for Simple T5 Model

	ROUGE-1	ROUGE-2	ROUGE-3
F1	0.222	0.222	0.222
Precision	0.225	0.225	0.225
Recall	0.214	0.215	0.259

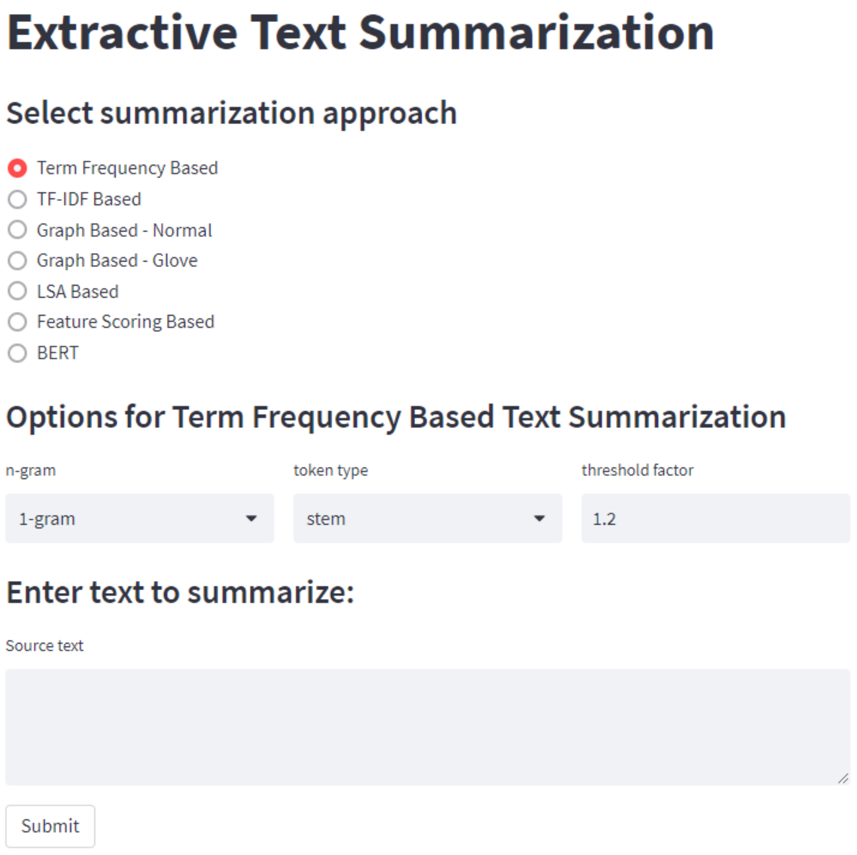
Table 4.5: Result for Pegasus Model

Frontend/GUI for User Inputs

5.1 Features

- Provides the option to choose the preferred method for Extractive Text Summarization
- Enables the input of user-specified values like threshold and tolerance as the parameters of the selected method
- Displays the sentence count for both the original text and the generated summary

5.2 Screenshots


Extractive Text Summarization

Select summarization approach

[✓ Term Frequency Based]
TF-IDF Based
Graph Based - Normal
Graph Based - Glove
LSA Based
Feature Scoring Based
BERT

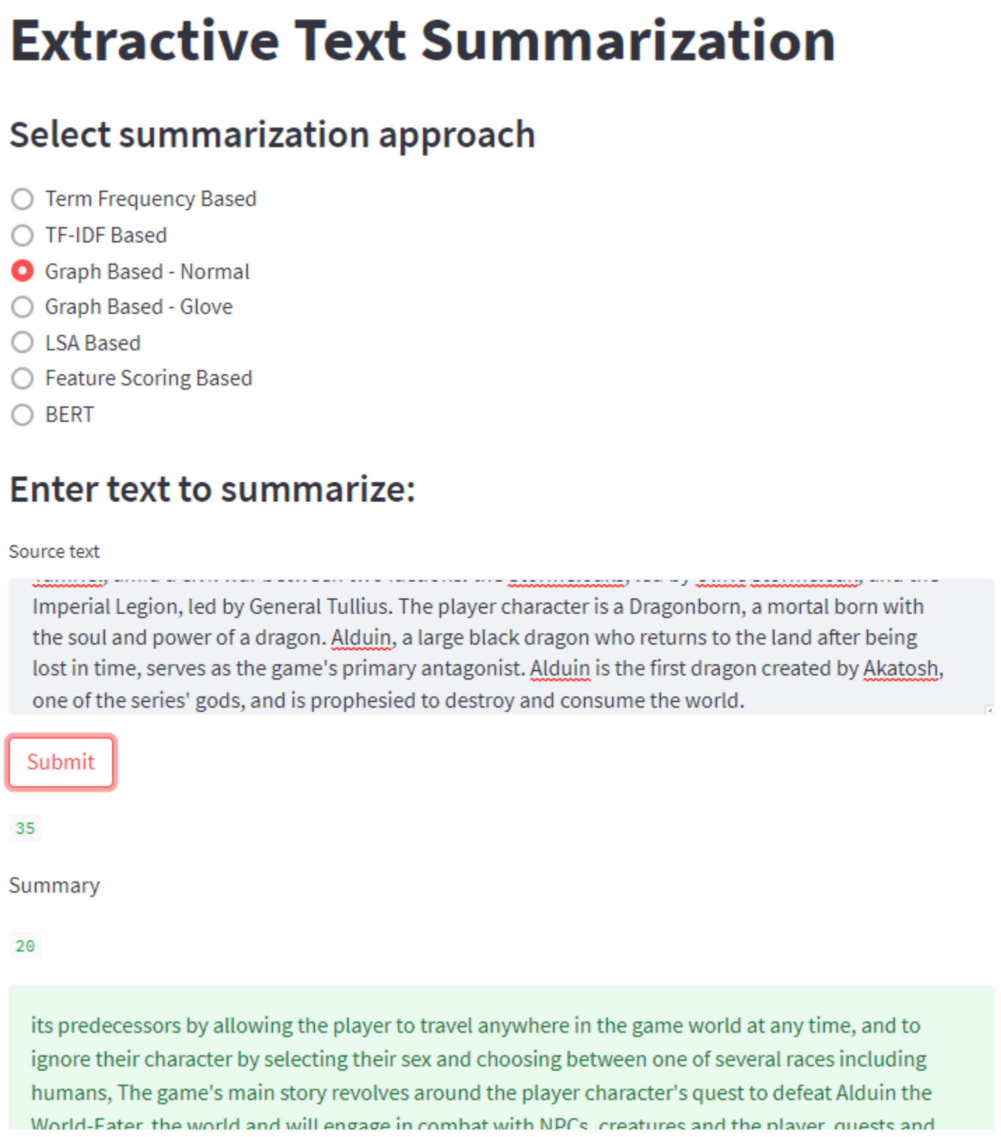
Options for Term Frequency Based Text Summarization

n-gram	token type	threshold factor
1-gram	stem	1.2

Enter text to summarize:

Source text

Figure 5.1: GUI Screenshot



Extractive Text Summarization

Select summarization approach

Term Frequency Based
TF-IDF Based
[✓ Graph Based - Normal]
Graph Based - Glove
LSA Based
Feature Scoring Based
BERT

Enter text to summarize:

Source text

[illegible]
Imperial Legion, led by General Tullius. The player character is a Dragonborn, a mortal born with the soul and power of a dragon. Alduin, a large black dragon who returns to the land after being lost in time, serves as the game's primary antagonist. Alduin is the first dragon created by Akatosh, one of the series' gods, and is prophesied to destroy and consume the world.

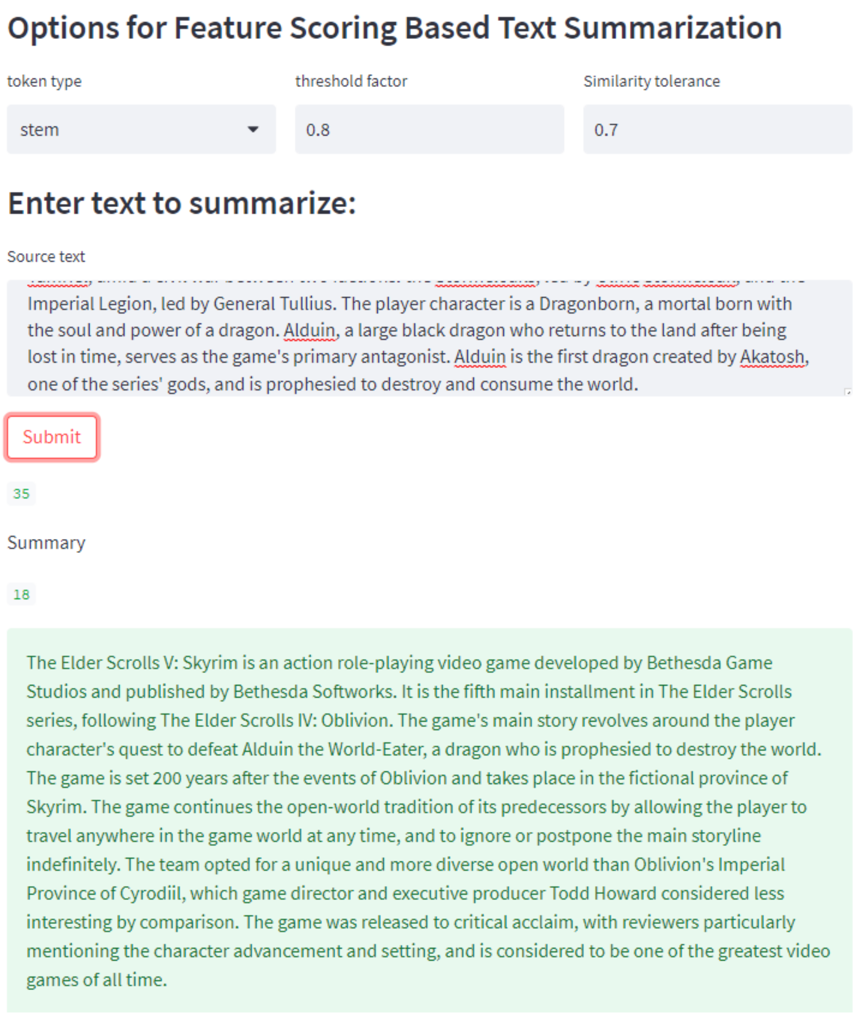
```
35
```

Summary

```
20
```

its predecessors by allowing the player to travel anywhere in the game world at any time, and to ignore their character by selecting their sex and choosing between one of several races including humans, The game's main story revolves around the player character's quest to defeat Alduin the World-Eater, the world and will engage in combat with NPCs, creatures and the player, quests and

Figure 5.2: GUI Screenshot



Options for Feature Scoring Based Text Summarization

token type: stem

threshold factor: 0.8

Similarity tolerance: 0.7

Enter text to summarize:

Source text

Imperial Legion, led by General Tullius. The player character is a Dragonborn, a mortal born with the soul and power of a dragon. Alduin, a large black dragon who returns to the land after being lost in time, serves as the game's primary antagonist. Alduin is the first dragon created by Akatosh, one of the series' gods, and is prophesied to destroy and consume the world.

35

Summary

18

The Elder Scrolls V: Skyrim is an action role-playing video game developed by Bethesda Game Studios and published by Bethesda Softworks. It is the fifth main installment in The Elder Scrolls series, following The Elder Scrolls IV: Oblivion. The game's main story revolves around the player character's quest to defeat Alduin the World-Eater, a dragon who is prophesied to destroy the world. The game is set 200 years after the events of Oblivion and takes place in the fictional province of Skyrim. The game continues the open-world tradition of its predecessors by allowing the player to travel anywhere in the game world at any time, and to ignore or postpone the main storyline indefinitely. The team opted for a unique and more diverse open world than Oblivion's Imperial Province of Cyrodiil, which game director and executive producer Todd Howard considered less interesting by comparison. The game was released to critical acclaim, with reviewers particularly mentioning the character advancement and setting, and is considered to be one of the greatest video games of all time.

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 user input article

Errors and Future Work

6.1 Errors and Caveats

The task of text summarization is a complex one, and there have been numerous errors observed in both extractive and abstractive methods employed for summarization. Some common errors in text summarization techniques include:

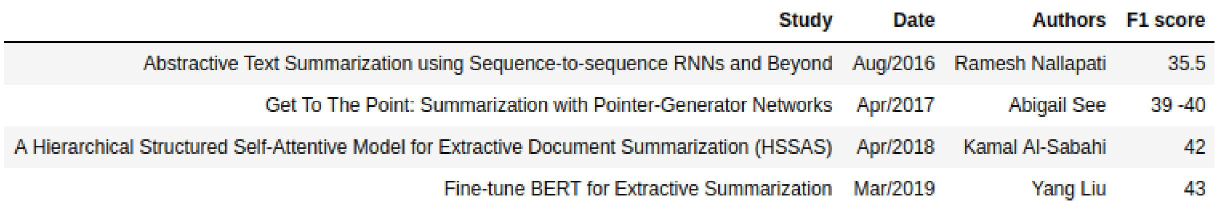
- Incoherence: In extractive summarization, the summary may contain disjointed or unrelated sentences that do not flow well. In abstractive summarization, the generated summary may contain language errors or inconsistencies.
- Bias: The summarization process can introduce bias by over-emphasizing certain 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.

6.2 Future Work

- The enhancement of ranking algorithms for extractive text summarization can be achieved by the exploration of additional semantic features that can be integrated into the ranking algorithms. The inclusion of semantic features has been observed to enhance the overall effectiveness of the algorithms.
- Applying Pronoun Augmentation on the articles
- Utilising multiple distributional semantic models to capture semantics in text, thereby enabling the capture of semantics at a more granular level for abstarctive text summarization.
- Validation of the technique on multiple datasets for more accurate results
- Utilisation of both BLEU (Bilingual Evaluation Understudy) and ROUGE metrics for evaluation of summaries.

Conclusion

Our work is producing summaries which ROUGE F1 score of **30.18**, 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. Le´on. 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