University of Hertfordshire UH

School of Physics, Engineering and Computer Science

MSc Data Science Project 7PAM2002-0901-2024

Department of Physics, Astronomy and Mathematics

Data Science FINAL PROJECT REPORT

Project Title:

Traditional Vs Transformer Models in NLP: A Comparative Study on Text Summarization

Student Name and SRN:

Sandra Binu - 22029960

Supervisor: Dr. Alyssa Drake

Date Submitted: 06 January 2025

Word Count: 6,830

GitHub address: **Text_Summarization_using_NLP-Project**

DECLARATION STATEMENT

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in Data Science at the University of Hertfordshire.

I have read the guidance to students on academic integrity, misconduct and plagiarism information at Assessment Offences and Academic Misconduct and understand the University process of dealing with suspected cases of academic misconduct and the possible penalties, which could include failing the project module or course.

I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged. (Ref. UPR AS/C/6.1, section 7 and UPR AS/C/5, section 3.6).

I have not used chatGPT, or any other generative AI tool, to write the report or code (other than where declared or referenced).

I did not use human participants or undertake a survey in my MSc Project.

I hereby give permission for the report to be made available on module websites provided the source is acknowledged.

Student SRN number: 22029960

Student Name printed: Sandra Binu

Student signature:

[illegible]

UNIVERSITY OF HERTFORDSHIRE
SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

Acknowledgment

With deep appreciation, I would like to thank everyone who helped me finish this project, ”Traditional Vs. Transformer Models in NLP: A Comparative Study on Text Summarisation.”

Firstly, I extend my deepest appreciation to my supervisor Dr.Alyssa Drake for her continuous guidance, insightful feedback, and invaluable support throughout this project. Her expertise and encouragement have been instrumental in refining the methodology and achieving meaningful results.

I am also grateful to the University of Hertfordshire for providing access to essential resources, research materials, and computational tools that enabled the smooth execution of the experiments involved in this study.

Special thanks to my peers and group members, whose discussions and feedback helped shape the direction of this project.

Finally, I would like to acknowledge the support of my family and friends for their constant encouragement and understanding throughout the course of this research. Their unwavering belief in my abilities motivated me to persevere and complete this work.

Thank you to everyone who played a part, directly or indirectly, in the completion of this project.

Abstract

In this study, the effectiveness of transformer-based and conventional models in text summarization tasks is compared. The CNN/Daily Mail dataset is used to test three models: TextRank, Word2Vec, and T5. T5 is an example of the state-of-the-art transformer architecture intended for abstractive summarization, whereas TextRank, a graph-based extractive method, and Word2Vec, a word embedding model, represent conventional approaches. ROUGE metrics are used to evaluate the models to determine recall, precision, and F1 scores. Based on all metrics, the results show that T5 performs noticeably better than traditional methods, indicating superior summarization quality. The study does, however, draw attention to the trade-off between the enhanced computational requirements of T5 and the effectiveness of conventional models.

Link to Google Colab: Colab Notebook Link

Link to GitHub: Text Summarization Code

Link to the Dataset: CNN/Daily Mail Dataset

Contents

1 Introduction

In today's big data-driven world, vast amounts of information are generated daily on platforms such as news outlets, social media, and industries. This overwhelming influx of data has increased the need for effective summarization techniques that distil complex information into concise and digestible formats. Text summarization, a critical task in natural language processing (NLP), addresses this challenge by generating concise summaries while retaining essential meaning.

It is broadly categorized into two approaches: extractive and abstractive summarization.

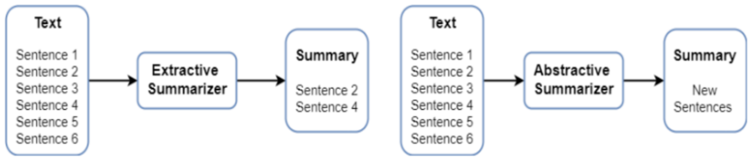

Figure 1: Extractive and Abstractive text summarization
(Yadav et al. 2022)

Figure 1 illustrates the fundamental difference between extractive and abstractive summarization approaches. Extractive methods select the most informative sentences directly from the source text. In contrast, abstractive methods generate new phrases to encapsulate the core information, often achieving more human-like summaries (Karjule et al. 2023).

Traditional methods for extractive summarization rely heavily on statistical techniques and feature engineering, offering simplicity and interpretability. However, contemporary advances in NLP leverage deep learning-based approaches to improve accuracy and generalization. Models like Word2Vec, a traditional word embedding-based method, have served as foundational tools in extractive summarization. However, transformer-based models such as T5 (Text-to-Text Transfer Transformer) have redefined the landscape with their ability to understand and generate natural language more effectively, particularly excelling in abstractive tasks.

This study aims to evaluate and compare the performance of three prominent models for text summarization. TextRank (a graph-based extractive method), Word2Vec (a word embedding-based extractive approach), and T5 (a cutting-edge transformer model for abstractive summarization). By analyzing these models using metrics such as ROUGE scores, computational efficiency, and scalability, the study seeks to illuminate their relative strengths and weaknesses.

The objective is to evaluate these models using performance measures such as ROUGE scores, training data, and architecture. This study aims to improve how we process and extract information from enormous datasets by examining the advantages and disadvantages of these three models to help create more effective and efficient text summarizing methods.

By bridging the gap between traditional and modern NLP paradigms, this research not only highlights the evolution of summarization techniques but also aims to identify opportunities for improvement in handling massive datasets. The findings will guide researchers and practitioners in developing more efficient and effective methods for text summarization, tailored to diverse real-world applications.

1.1 Research Questions

1. How does the performance of traditional word embedding models like TextRank and Word2Vec compare to state-of-the-art transformer-based models like T5 for text summarization?
2. What are the computational trade-offs between traditional (Textrank and Word2Vec) and transformer-based (T5) summarization models in terms of efficiency and complexity on the CNN/Daily Mail dataset?

1.2 Objectives

- Text Summarization using Textrank (traditional): Implement extractive summarization using the graph-based model Textrank on the CNN/Daily Mail dataset
- Text Summarization using Word2Vec (traditional): Implement and train extractive summarization using word embedding model Word2Vec, on the same dataset to facilitate direct comparison.
- Text Summarization using T5: Implement and train extractive summarization using transformer-based model T5, on the same dataset to facilitate direct comparison.
- Evaluate and Compare Model Performance: Compare the accuracy, relevance, and coherence of summaries generated by Textrank, Word2Vec, and T5 using evaluation metrics such as ROUGE.
- Analyze Trade-offs in Model Efficiency and Complexity: Analyze the computational trade-offs between traditional and transformer-based models regarding model efficiency and computational complexity.

2 Literature Review and Background

Text summarization is crucial in natural language processing (NLP), with the aim of condensing large textual data into concise and information-rich summaries. Automatic text summarization is a very challenging task in Natural Language Processing (NLP), and several methods address it. (Karjule et al. 2023).

The text summarization techniques are broadly categorized into extractive and abstractive approaches. Extractive summarization selects key sentences or phrases directly from the source text, forming a summary with original words and syntax from the input. In contrast, abstractive summarization rephrases and condenses the text, imitating human summarization. Extractive text summarization remains highly popular due to its simplicity and efficiency, especially in handling large datasets like the CNN/Daily Mail corpus (Karjule et al. 2023). Consequently, it has found applications in areas ranging from news articles to customer reviews, as noted by (Yadav et al. 2022).

Extractive summarization is widely used in various application areas, including news articles to provide concise headlines, medical records for summarizing patient information, books for generating synopses, legal documents for case summaries, and customer reviews to extract key opinions. It is also applied in blogs, tweets, and other social media posts to capture essential content quickly and efficiently (Yadav et al. 2022).

TextRank,(Mihalcea & Tarau 2004) is a widely used unsupervised graph-based ranking algorithm for extractive summarization. It creates a similarity graph with sentences serving as nodes and edges indicating semantic similarity based on content overlap. The most crucial sentences to include in the summary are determined by TextRank using an iterative scoring procedure similar to PageRank. Because it is unsupervised and does not require training data, it can be applied to a variety of domains and is therefore useful for summarization tasks.

(Mihalcea & Tarau 2004) explains in the paper that Google's PageRank algorithm, which was first used to rank web pages according to hyperlink connectivity, serves as the model's inspiration.

Extractive summarization on large text was difficult and previously used traditional methods were supervised learning techniques that need training data with annotations, which can be costly and domain-specific. The authors suggest TextRank, an unsupervised, domain-independent algorithm that successfully ranks and identifies the most important components in a text, as a solution to this drawback (Mihalcea & Tarau 2004).

(Gulati et al. 2023) comprehends that the primary goal of the similarity function is to produce a metric for gauging how similar two sentences in a document are.

The choice of similarity function significantly affects the performance of TextRank by influencing the structure of the similarity graph, ultimately determining the relevance of sentences in the summary. The research by (Gulati et al. 2023) explains various similarity functions, as shown in Fig.2, which explains the TF-IDF cosine similarity function. TF-IDF calculates the importance of words in a document by considering their frequency relative to the entire corpus. Using these values, sentences are vectorized, and cosine similarity is computed to measure the similarity between sentence pairs, forming a similarity matrix used in TextRank for summarization (Gulati et al. 2023).

(Wang et al. 2016) explained that Word2Vec is a model that uses a three-layer neural network (input, projection, and output layers) to map words into a low-dimensional space while leveraging contextual information to establish relationships and meanings between words (e.g., "king - man + woman = queen"). This paper proposes a novel optimization

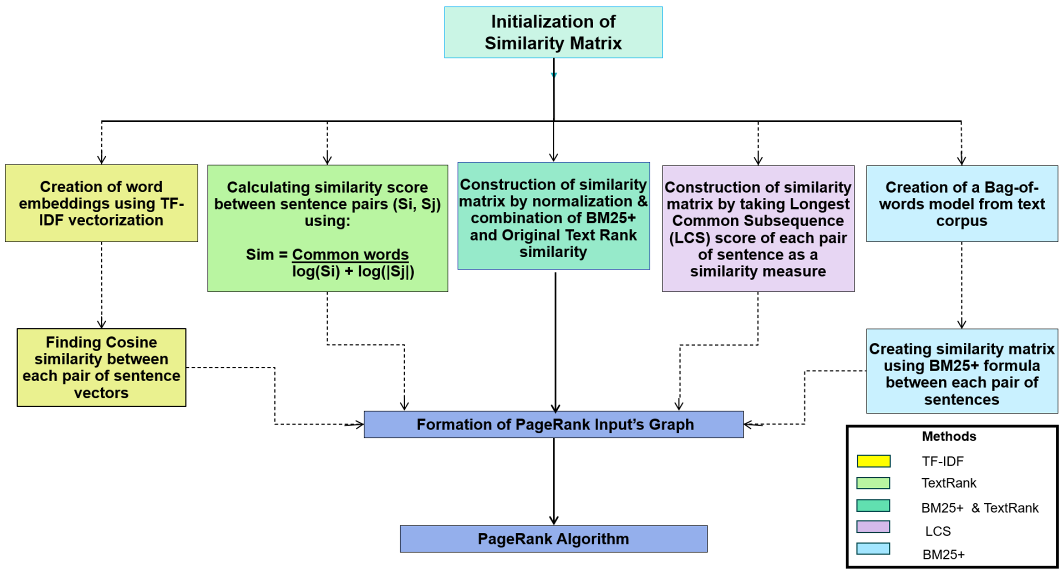
Initialization of Similarity Matrix
Creation of word embeddings using TF-IDF vectorization
Calculating similarity score between sentence pairs (Si, Sj) using:
Sim = Common words / log(Si) + log(|Sj|)
Construction of similarity matrix by normalization & combination of BM25+ and Original Text Rank similarity
Construction of similarity matrix by taking Longest Common Subsequence (LCS) score of each pair of sentence as a similarity measure
Creation of a Bag-of-words model from text corpus
Finding Cosine similarity between each pair of sentence vectors
Creating similarity matrix using BM25+ formula between each pair of sentences
Formation of PageRank Input's Graph
PageRank Algorithm
Methods
TF-IDF
TextRank
BM25+ & TextRank
LCS
BM25+

Figure 2: Various Similarity functions of TextRank algorithm. Adapted from (Gulati et al. 2023).

method for document summarization that extends Word2Vec by clustering word vectors using K-means, evaluating the semantic correlation of document partitions, and retaining only topic-related sections to reduce noise and improve classification accuracy.

Word2Vec model operates using two architectures: CBoW (Continuous Bag of Words) and Skip-gram as illustrated in Fig.3.

(Wang et al. 2016) specifically adopts the CBOW scheme with the Hierarchical Softmax framework to demonstrate the algorithm showing how Word2Vec captures relationships between words in a semantic space.

In the "Attention is All You Need" paper, the authors introduce the Transformer model, which relies solely on attention mechanisms to process input sequences in parallel, eliminating the need for recurrent and convolutional networks (Vaswani 2017).

The Transformer model helps in text summarization by utilizing self-attention mechanisms that allow the model to weigh the importance of different words in a sentence, capturing long-range dependencies and enabling more effective representation of the input text for tasks like summarization (Vaswani 2017).

(Ravichandran et al. 2023) discuss how the T5 transformer model uses advanced language modeling techniques, specifically the transformer architecture, to capture contextual relationships and generate coherent, concise summaries.

Evaluating the generated summaries is important to ensure that the meaning of the corpus is preserved in the summary produced. (Barbella & Tortora 2022) explain that the ROUGE metric is commonly used to evaluate text summarization techniques by measuring the overlap of n-grams and the longest common subsequence between machine-generated summaries and reference summaries.

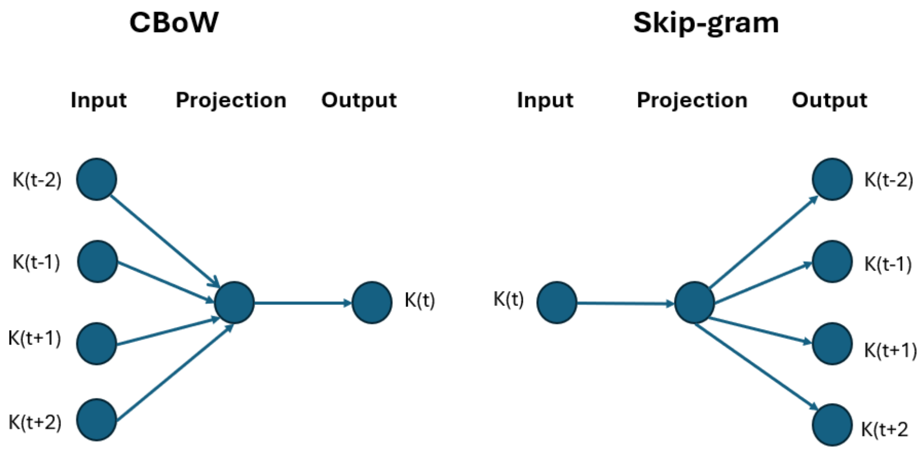

Figure 3: CBoW and Skip-gram architectures in Word2Vec

3 Ethical Requirements

This research work aligns with the legal, professional, cultural and personal obligations and follows all the ethical principles in accordance with the UH rules (Academic Integrity and Academic Misconduct UPR AS14).

The CNN/Daily Mail dataset includes anonymization measures that align with UK GDPR and UK Data Protection Act 2018 requirements, ensuring ethical use in research. The dataset and the research work clearly align with the UH ethical requirements. Proper licensing and adherence to institutional policies further enhance its suitability for academic studies.

The dataset is released under the **Apache-2.0 License** permitting academic and commercial use.

The link to the license: http://www.apache.org/licenses/LICENSE-2.0

Licensing Information

The CNN / Daily Mail dataset version 1.0.0 is released under the Apache-2.0 License.

4 Methodology

4.1 Project Overview

This project was developed in Google Colaboratory using Python and a wide range of libraries to implement and evaluate text summarization techniques. The libraries utilized include NumPy, Pandas, Matplotlib, and Seaborn for data manipulation, exploration, and visualization. Libraries such as Scikit-learn, Gensim, and NLTK were employed for preprocessing and implementing traditional NLP models while Hugging Face Transformer and PyTorch were leveraged for modern transformer-based techniques.

In the first steps, the CNN/Dailymail dataset is loaded into the Google colab as training, testing and validation csv files, and a subset from each was randomly taken for further model training and analysis. Then the data was explored for checking the missing values and vocab word frequency. The preprocessing script was implemented using Python with libraries such as re for regular expressions, NLTK for tokenization and lemmatization, and stopwords to filter irrelevant words.The preprocessing function was designed to ensure compatibility with both the extractive summarization (TextRank) and embedding-based approaches (Word2Vec). Next, each of the three models (TextRank, Word2Vec, T5) are implemented and for evaluation, ROUGE metrics is used.

4.2 Data Overview

The dataset used for this project is CNN/Daily Mail dataset. The CNN/Daily Mail Dataset is an extensive collection of over 300,000 unique English-language news articles sourced from CNN and the Daily Mail. The dataset was originally created between April 2007 and April 2015, with articles collected through automated web scraping techniques. The dataset was curated by researchers from Google DeepMind and modified for summarization by teams from IBM Watson, Université de Montréal, and Stanford University. Researchers aimed to support machine reading comprehension and question-answering tasks, but subsequent versions have shifted focus to text summarization tasks, allowing for both extractive and abstractive summarization applications.

4.2.1 Data Collection

The data was originally collected by Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom of Google DeepMind, with Tomáš Kočiský and Phil Blunsom also affiliated with the University of Oxford. For this project, the dataset with version 1.0.0 was collected from Hugging Face.

4.2.2 Exploratory Data Analysis

The purpose of this EDA is to understand the characteristics of the text data, including its structure, length distributions, and potential preprocessing needs for downstream tasks like text summarization.

The dataset is already present in the HuggingFace as three subsets: training (95,705 entries), testing (11,490 entries), and validation (13,368 entries). Each entry contains three fields: article (the full text), highlights (summaries), and id (a unique identifier). The training subset was further sampled into a subset of 5,000 entries *(train_subset)*, the testing into

1000 samples *(test_subset)*, and the validation data into 100 *(val_subset)*, for easy training purposes.

All fields in the dataset are non-null and of type object. This confirms the data is complete, requiring no handling of missing values.

- Text Length Analysis: The word count of articles and summaries was analyzed using histograms. Articles tend to have a broader length distribution compared to summaries.
  - Article length: The histogram of article lengths, as shown in Figure 4, indicates that the dataset contains a mix of short and long articles, with the majority being relatively short. The right-skewed distribution suggests that longer articles are less common but still present.

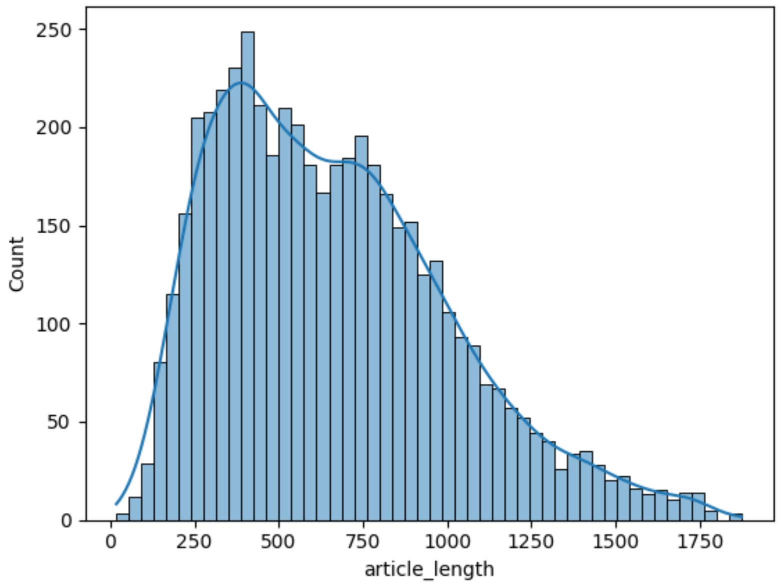

Figure 4: Article Length Histogram

    For the text summarization task in this project, normalizing text lengths is necessary. This involves techniques like truncation or padding.

  - Summary length: The histogram of summary lengths, as shown in Figure 5, indicates that most summaries in this dataset are relatively short, with a few longer ones. This also can be normalized using padding or truncating.

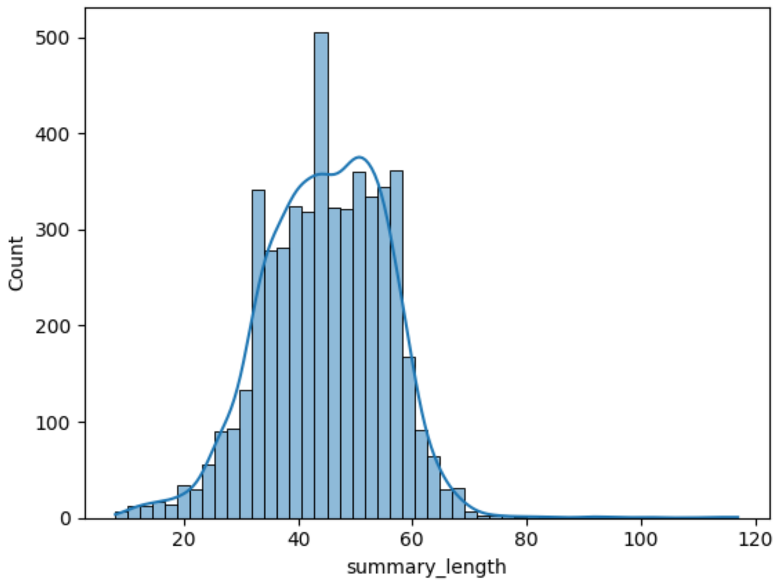

Figure 5: Summary Length Histogram

Summary lengths exhibit a mean of 44.8 words with a standard deviation of 10.68, indicating some variation but generally consistent lengths.

- Word Cloud: A word cloud was generated, as shown in Figure 6, to examine the most frequent terms in the dataset. Common words in the articles (excluding stop words) include 'said', 'people', 'one' etc. This highlights dominant themes in the text data.

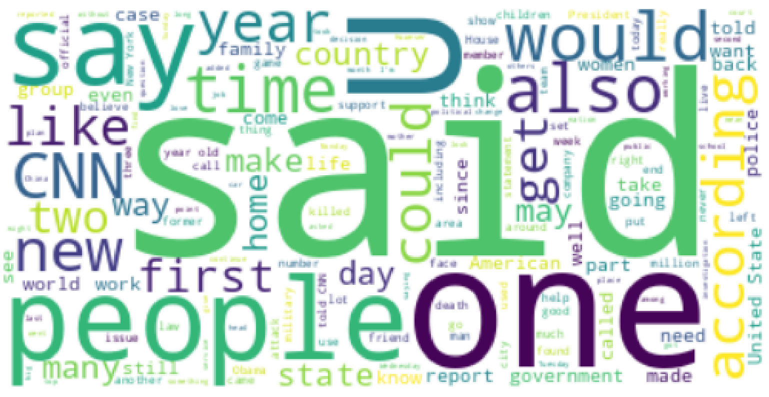

Figure 6: Word Cloud of *train_subset*

- **Stop words** from the NLTK library were identified and removed to focus on meaningful words. **Lemmatization** was applied using WordNetLemmatizer to reduce words to their base forms, aiding in consistency across text entries.

4.3 TextRank

4.3.1 Data Preprocessing for TextRank

The TextRank algorithm was implemented to perform extractive text summarization. Initially, the subset of CNN/Daily mail dataset chosen is preprocessed for TextRank summarisation. The text data is first cleaned to remove unwanted characters and standardize its format. For example, "I am going to the store."

1. **Lowercasing:** All text is converted to lowercase to ensure uniformity and eliminate case sensitivity issues. This simplifies the text and avoids treating words like "The" and "the" as different terms during vectorization. After lowercasing the above example becomes "i am going to the store."

2. **Removal of Special Characters, Numbers, and Extra Spaces:**Special characters (e.g., punctuation marks), numbers, and extra whitespace are removed from the text using a regular expression (). This step helps to clean the text and retain only alphabetic characters, which are most relevant for text summarization. The baove eg., becomes "i am going to the store".

3. The cleaned text is tokenized into words using the `word_tokenize` function from NLTK:

   - **Sentence Tokenization**: The text is first split into individual sentences using `sent_tokenize`. These sentences form the nodes of the similarity graph in the TextRank algorithm. The above example becomes ["i am going to the store"].
   - **Word Tokenization**: Each sentence is further split into individual words for additional processing steps. The above example becomes ["i", "am", "going", "to", "the", "store"].

4. **Stopword Removal:** Commonly used words with little semantic meaning (e.g., "and", "the", "is") are removed using a predefined stopword list. This helps in focusing on more meaningful words that contribute to the content of the text. The above example becomes ["going", "store"].

5. **Lemmatization:** Each word is reduced to its base form using lemmatization. For example, "running" is reduced to "run", and "better" is reduced to "good".This ensures consistency and reduces variations of the same word, improving the quality of sentence vectorization in later steps. The above example becomes ["go", "store"].

6. **Reconstructing Cleaned Sentences:** After stopword removal and lemmatization, the processed words of each sentence are joined back into a single string. This cleaned version of the sentences is used for further processing. The above example becomes "go store".

Lowercasing, lemmatization, and removal of noise ensure that the text is consistent and free from irrelevant elements. Stopword removal focuses the analysis on important words that contribute to the meaning of the sentences. Cleaning and tokenizing the sentences ensures that the term-document matrix created during TF-IDF vectorization is accurate and meaningful. This preprocessing pipeline is integral to preparing the text for TextRank's graph-based approach, as it ensures that the sentences and their relationships are represented effectively for ranking.

4.3.2 TF-IDF Vectorization

To quantify the semantic content of each sentence, the cleaned sentences were transformed into numerical representations using the Term Frequency-Inverse Document Frequency (TF-IDF) vectorizer.

$$\text{TF}(t, d) = \frac{\text{Number of times term } t \text{ appears in sentence } d}{\text{Total number of words in sentence } d} \tag{1}$$

$$\text{IDF}(t, D) = \log\left(\frac{\text{Total number of sentences } N}{\text{Number of sentences containing term } t}\right) \tag{2}$$

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D) \tag{3}$$

As shown in Equation 1, the Term Frequency (TF) measures the number of times the term occurs in the corpus(Zaware et al. 2021) and as shown in Equation 2, the Inverse Document Frequency measures the importance of a term within a document (Zaware et al. 2021)

Next, a pairwise cosine similarity matrix was computed between all sentence vectors. This matrix represents the degree of similarity between each pair of sentences. Sentences with high similarity scores are more likely to share common information or themes.

4.3.3 Similarity matrix

Cosine similarity is a metric commonly used in NLP to quantify the similarity between two textual documents by evaluating their orientation in a multi-dimensional vector space, irrespective of their size. Each word is transformed into a vector representation, and the documents are embedded in an n-dimensional space. The cosine similarity value ranges from 0 to 1, where 0 indicates no similarity, and 1 indicates complete similarity.(Zaware et al. 2021)

$$\text{Cosine similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} \tag{4}$$

4.3.4 Graph Construction and Sentence Scoring

The cosine similarity matrix was converted into a graph structure using NetworkX. In this graph nodes represent sentences and edges between nodes are weighted by the corresponding similarity scores from the matrix. This graph captures the relationships between sentences in terms of their semantic similarity.

The **PageRank** algorithm was applied to the graph to calculate a score for each sentence. Sentences with more connections to other highly similar sentences (strong relationships) received higher scores. This scoring mechanism effectively identifies sentences that are central to the text's overall meaning.

The sentences were ranked in descending order of their scores. The top n sentences (where n is a user-specified parameter, default set to 2) were selected to form the summary. The selected sentences were then combined in their original order to preserve the context and coherence of the text.

The final summary was constructed as a concatenation of the selected sentences. This extractive approach ensures that the output consists of actual sentences from the input text, making it both coherent and faithful to the original content.

4.4 Word2Vec

4.4.1 Data Preprocessing for Word2Vec

The Word2Vec approach was implemented to generate extractive summaries by leveraging word embeddings and sentence similarity. The preprocessing pipeline ensures that the text is appropriately cleaned and prepared for embedding-based analysis. The following steps were performed:

1. **Text Cleaning**: Non-alphabetic characters were removed using regular expressions (`\W`), leaving only meaningful text.
2. **Sentence Tokenization**: The cleaned text was split into individual sentences using NLTK's `sent_tokenize`, as Word2Vec operates on sentence-level tokenized data.
3. **Word Tokenization and Processing**:
   - Each sentence was split into individual words using `word_tokenize`.
   - Stop words (e.g., "the", "is", "and") were removed using a predefined stop-word list.
   - Lemmatization was applied to reduce words to their root forms (e.g., "running" $\rightarrow$ "run") using NLTK's `WordNetLemmatizer`.
4. **Output**: Each sentence was represented as a list of processed tokens used as input for training the Word2Vec model.

This preprocessing pipeline ensures that the data is consistent, free from noise, and ready for embedding generation.

4.4.2 Word2Vec Model Training

Tokenized sentences from the preprocessing step are provided as input to the Word2Vec model. The preprocessed data was used to train a Word2Vec model to generate dense vector representations for words. The following parameters were set during training:

- **Vector Size**: The dimensionality of the word vectors was set to 100.
- **Window Size**: The context window size was set to 5 words i.e., 5 words to the left and right of a target word should be considered when learning the word embeddings.
- **Minimum Count**: Words that appeared fewer than 5 times were ignored. This helps to reduce noise from infrequent or irrelevant words.
- **Architecture**: The Skip-Gram (`sg=1`) model was used to predict the context of a word. The sg parameter specifies the training algorithm. sg=1 uses the Skip-Gram model, which predicts the context words given a target word. sg=0 would use the CBOW (Continuous Bag of Words) model, which predicts a target word based on its context words.For the CNN/Daily Mail dataset, Skip-Gram (sg=1) is often preferred for its ability to capture nuanced relationships, particularly for rare words, which is advantageous for summarization tasks, whereas CBOW (sg=0) is faster and suited for small, homogeneous datasets.

- **Workers**: 4 CPU threads were utilized for parallel processing, speeding up the process.

The model learned semantic relationships between words, enabling meaningful comparisons between them. That is, words with similar meaning tend have vectors closer in the vector space.

4.4.3 Sentence Vectorization

To compare sentences, each sentence was converted into a fixed-size vector representation:

1. **Word Vectors**: Each word in a sentence was matched with its vector representation from the Word2Vec vocabulary.
2. **Sentence Vector**: The sentence vector was computed as the average of the word vectors in the sentence. This approach ensures that the sentence is represented as a single dense vector while capturing the semantic essence of its words.
3. **Handling Unknown Words**: For sentences containing words not present in the vocabulary, a zero vector was assigned.

4.4.4 Similarity Matrix

The cosine similarity between sentence vectors was calculated to measure semantic similarity:

- A pairwise cosine similarity matrix was created, where each entry represents the similarity between two sentences.
- The cosine similarity values ranged from 0 (completely dissimilar) to 1 (identical). (Explained in Section 4.3.3)

This matrix was critical in identifying sentences that were central to the document's meaning.

4.4.5 Extractive Summarization using Word2Vec

The extractive summary was generated by selecting the most relevant sentences based on their similarity scores:

1. **Scoring Sentences**: Each sentence was assigned a similarity score based on its relationships with other sentences.
2. **Ranking Sentences**: Sentences were ranked in descending order of their similarity scores.
3. **Summary Construction**: The top n sentences (where n is a user-defined parameter, default = 2) were selected and concatenated to form the summary.

This approach ensures that the summary retains the most semantically significant sentences while maintaining coherence.

The Word2Vec methodology enables the generation of concise, contextually relevant summaries, making it suitable for extractive summarization tasks across diverse text datasets.

4.5 T5 (Text-to-Text Transfer Transformer)

The T5 model is an advanced language model designed for NLP tasks, beneficial for text summarization. T5 follows a sequence-to-sequence framework, converting input text into the target text, making it versatile for tasks like summarization, translation, and question-answering (Ravichandran et al. 2023).

It is particularly good at identifying contextual relationships in text, which enables it to produce summaries that are human-like and coherent while preserving the main ideas of the original material. T5 streamlines training and application across various domains by treating each NLP task as a text-to-text problem (Ravichandran et al. 2023).

4.5.1 Transformer Model Architecture

The transformer model was introduced in the 2017 paper "Attention Is All You Need" by a team of researchers at Google. By showcasing the strength of attention mechanisms and opening the door for a new generation of potent and effective neural network architectures, the "Attention Is All You Need" paper transformed sequence-to-sequence modelling (Vaswani 2017).

The attention mechanism serves as the sole foundation for the novel neural network architecture that is proposed in this paper for sequence-to-sequence tasks. This is different from the widely used recurrent neural network (RNN)-based encoder-decoder architectures. With the Transformer, a purely attention-based mechanism takes the place of recurrence and convolutions. More parallelization during training is made possible by this, which significantly increases speed (Vaswani 2017).

The architencture of a transforner model is given in Figure 7 whcih shows the flow through encoder-decoder architecture

A summary of what encoder, decoder, and self-attention mechanism from the paper "*Attention is all you Need*" is given below:

Encoder: The encoder attends to every position in the input sequence and transforms it into a continuous representation. Each of its several layers includes a self-attention mechanism and a feed-forward neural network. For every token, the encoder creates contextual embeddings by identifying connections throughout the input (Vaswani 2017).

Decoder: Using the encoder's output and previously created tokens, the decoder creates the output sequence. It has several layers, each of which has masked self-attention, cross-attention to the encoder's output, and a feed-forward neural network. Using historical outputs and input context, the decoder makes incremental token predictions (Vaswani 2017).

Self-attention Mechanism: By focusing on every other token in the input sequence and assessing their relative importance, self-attention enables each token to calculate a contextual representation. It calculates attention scores using queries, keys, and values, and applies softmax to highlight relevant tokens. Through this process, the model is able to identify long-term relationships and dependencies in the input (Vaswani 2017).

The Transformer's self-attention mechanism is its central component. When calculating a representation for a specific position, this enables the model to directly attend to every position in the input sequence. By using multiple attention heads, the model can concurrently attend to data from various representation subspaces. Positional information is added to the input embeddings using sinusoidal functions of various frequencies because the Transformer does not rely on recurrence (Vaswani 2017).

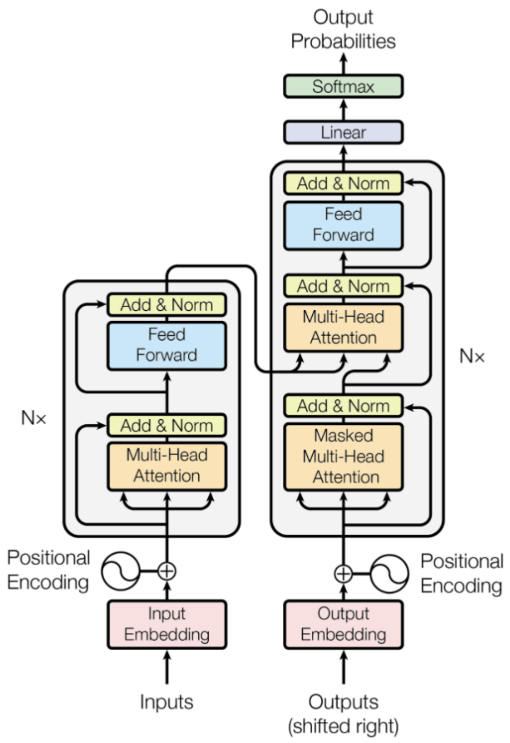

Figure 7: Encoder-Decoder structure of a transformer architecture(Vaswani 2017)

4.5.2 Data Preprocessing for T5

The preprocessing pipeline ensures the dataset is prepared in a format compatible with the T5 model while maintaining its contextual integrity. The following steps were performed:

- The *train_subset* and *test_subset* are converted into Hugging Face `Dataset` objects for compatibility with the training pipeline.
- Task-Specific Prefix: Each article is prefixed with the word "*summarize:*" to explicitly instruct the T5 model about the summarization task, as T5 is a multitask model.
- Tokenization: The input articles and target summaries are tokenized using the `AutoTokenizer` from the Hugging Face library.
  - Input Truncation: Articles are truncated to a maximum length of 512 tokens to manage computational constraints.
  - Output Truncation: Summaries are truncated to a maximum length of 128 tokens to maintain focus on concise summaries.
- Label Assignment: The tokenized summaries are assigned as labels for the model, ensuring it learns to generate corresponding outputs.

Both the datasets are tokenized and preprocessed in batches. The tokenized datasets are further transformed into a PyTorch-compatible format with selected columns (*input_ids*, *attention_mask*, and *labels*).

4.5.3 Model Initialization and Fine-Tuning

The pre-trained T5 model (*t5-small*) is loaded using the `AutoModelForSeq2SeqLM` class from Hugging Face. During batch training, padding is handled dynamically using a `DataCollatorForSeq2Seq`. This reduces computational overhead by guaranteeing that input sequences in a batch are padded to the batch's maximum length.

Fine-tuning is performed using the Hugging Face `Seq2SeqTrainer`, which simplifies the training process for sequence-to-sequence tasks. The training configuration is specified using `Seq2SeqTrainingArguments`, which include:

- **Output Directory and Evaluation Strategy:** Former specifies where model checkpoints and results are saved and latter Performs evaluation at the end of each training epoch.
- **Learning Rate:** Set to 2e-5 for gradual weight updates during training.
- **Batch Sizes and Number of Epochs:** batch size configured for both training and evaluation, set to 18 per device. Training is run for 3 epochs.
- **Mixed Precision Training and Weight Decay:** Former enabled with `fp16` to reduce memory usage and accelerate training. Weight decay is applied with a value of 0.01 to prevent overfitting.
- **Checkpointing:** Saves model checkpoints every 500 steps, with a limit of 2 checkpoints to manage storage.
- **Evaluation Metric:** The model's performance is monitored using the training loss.

4.5.4 T5 text summarization

After training, the model is evaluated on a validation subset. The tokenizer and model from the best checkpoint are reloaded, ensuring the most accurate parameters are used for evaluation.The validation articles are tokenized using the trained tokenizer.

The trained model generates summaries using the `generate` method, which implements beam search with parameters to control the randomness and quality of output:

- **Beam Width:** Set to 5 to consider multiple potential sequences.
- **Temperature:** Adjusted to 1.2 to encourage some randomness.
- **Top-k and Top-p Sampling:** Limits the probability space for token sampling.
- **Repetition Penalty:** Applied to discourage repetitive sequences.
- **Early Stopping:** Ensures sequences are terminated once the model predicts an end token.

5 Justification of Model Selection

1. TextRank with TF-IDF :

   - Simplicity and Efiiciency: TextRank is a great starting point for extractive summarisation tasks because it is computationally efficient and doesn't require labelled data.
   - TF-IDF Integration: By giving significant words a higher weight, TF-IDF (Term Frequency-Inverse Document Frequency) improves the model and raises the score for sentence relevance. The frequency and uniqueness of terms are both well captured by this hybrid approach.
   - Baseline Model: TextRank with TF-IDF provides a straightforward and interpretable baseline against which to compare the performance of more complex models.

2. Word2Vec with Sentence Scoring

   - Semantic Understanding: The model is able to comprehend contextual meaning more effectively than basic statistical techniques by creating semantic relationships between the words using dense vector representations and averaging or aggregating word vectors for sentences.
   - Sentence Scoring: The model can rank sentences according to how closely they match the text's overall semantic representation by employing cosine similarity.
   - Increased Relevance: Word2Vec captures more complex relationships between words than TF-IDF, which is only statistical. This makes extractive summaries more contextually relevant and coherent.
   - Traditional yet Effective: This method bridges the gap between basic statistical techniques and more complex deep learning models, and it is more complex than TF-IDF alone.

3. T5-Small (Transformer Model):

   - Scalability and Flexibility: The T5-small variant strikes a balance between computational efficiency and performance, making it feasible to train on constrained hardware while still being appropriate for large datasets like CNN/Daily Mail.
   - Contextual Understanding: In contrast to conventional models, T5-small is able to synthesise information across sentences, create original phrases, and rephrase content to produce summaries that are more human-like.
   - Benchmark Comparison: By incorporating T5, it is possible to assess how transformer-based models outperform conventional summarisation methods, showcasing the effectiveness of deep learning in summarisation tasks.

6 Results and Analysis

6.1 ROUGE

Recall-Oriented Understudy of Gisting Evaluation (ROUGE) is a metric used for comparing the generated summary with the reference (human-generated summary). The ROUGE metric scores range from 0 to 1, with 1 showing the highest similarity and 0 with no similarity.

Different ROUGE metrics used in this study are:

- ROUGE-N: measures the overlap of n-grams (sequence of words or symbols) between the produced and reference summaries.
  - ROUGE-1: measures the overlap of unigrams
  - ROUGE-2: measures the overlap of bigrams
- ROUGE-L: measures the Longest Common Subsequence (LCS) ie., looks into the structure similarity of the sentences.

In this study we look into the Recall(r), Precision(p) and F1-score from the ROUGE scores.

Recall measures how much the generated text is captured from the reference summary.

$$\text{Recall} = \frac{\text{Number of overlapping words}}{\text{Total words in the reference}} \tag{5}$$

Precision measures how accurate is the generated summary.

$$\text{Precision} = \frac{\text{Number of overlapping words}}{\text{Total words in the generated text}} \tag{6}$$

The **F1-score** is the harmonic mean of precision and recall which gives the balanced measure when precision and recall differ.

$$F_1 = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}} \tag{7}$$

6.2 TextRank Results

Metric	Recall	Precision	F1-Score
ROUGE-1	0.335	0.355	0.330
ROUGE-2	0.136	0.140	0.131
ROUGE-L	0.298	0.318	0.295

Table 1: ROUGE scores for TextRank Model.

The ROUGE scores (recall,precision, and F1 score) presented in Table 1 evaluate the performance of the TextRank model for text summarization across ROUGE-1, ROUGE-2, and ROUGE-L metrics.

Observations:

1. ROUGE-1:
    - The precision (0.355) indicates that approximately 35.5% of the words in the generated summaries are relevant to the reference summaries.
    - The recall (0.335) suggests that around 33.5% of the words from the reference summaries are captured by the generated summaries.
    - The F1-score (0.330) reflects a balance between precision and recall, indicating a moderate level of quality in unigram overlap.
2. ROUGE-2:
    - The lower values, particularly the F1-score (0.131), suggest that while the model performs decently at the word level, it struggles to capture larger, more complex patterns in the text.
    - This is to be expected, given that TextRank is extractive and might overlook subtle word combinations that are more prevalent in abstractive models.
3. ROUGE-L
    - Although the TextRank-generated summaries maintain some of the reference's structural integrity, the F1-score of 0.295 indicates that sequence consistency could be improved.

6.3 Word2Vec Results

Metric	Recall	Precision	F1-Score
ROUGE-1	0.353	0.361	0.345
ROUGE-2	0.148	0.149	0.143
ROUGE-L	0.298	0.306	0.292

Table 2: ROUGE scores for Word2Vec Model.

Table 2 shows the ROUGE scores of the text summarization using the Word2Vec model.

Observations:

1. ROUGE-1
    - The precision (0.361) indicates that 36.1% of the words in the generated summaries are relevant to the reference summaries.
    - The recall (0.353) suggests the model successfully retrieves 35.3% of the critical words from the reference summaries.
    - The F1-score (0.345) demonstrates a balanced performance, suggesting moderate improvement over TextRank's unigram results (0.330).

2. ROUGE-2
    - Although the scores are relatively low, they show a slight improvement over TextRank (F1-score of 0.131).
    - This suggests that the Word2Vec model better captures phrase-level information, though there is still room for significant improvement.
3. ROUGE-L
    - The F1-score (0.292) is comparable to TextRank (0.295), indicating that the structural coherence of the summaries is preserved at a similar level.
    - Recall (0.298) and precision (0.306) suggest that the model performs slightly better in structure retention compared to the TextRank model.

6.4 T5 Results

Metric	Recall	Precision	F1-Score
ROUGE-1	0.405	0.387	0.387
ROUGE-2	0.197	0.181	0.183
ROUGE-L	0.380	0.365	0.363

Table 3: ROUGE scores for T5 Model.

Table 3 shows the ROUGE scores of the text summarization using the T5 model.
Observations:

1. ROUGE-1
    - A recall of 0.405 indicates that 40.5% of the words in the reference text appear in the predicted output. This is an improvement over the Word2Vec (35.3%) and TextRank (33.5%) models, showing that T5 is more successful in retrieving relevant words from the reference summaries.
    - A precision of 0.387 means that 38.7% of the words in the predicted output exist in the reference. This is slightly lower than the recall, indicating that while the model captures a significant portion of relevant unigrams, it still generates some irrelevant words.
    - The F1-score is equal to the precision, which reflects a balanced trade-off between precision and recall. Since the recall is slightly higher than the precision, this is a good sign that T5 is retrieving more relevant content.
2. ROUGE-2
    - The recall is 19.7%, which is higher than Word2Vec's and TextRank's recall. This suggests that T5 is better at capturing relevant bigrams (two consecutive words) from the reference summaries.
    - The precision is 18.1%, which is slightly better than TextRank's and Word2Vec's. This indicates that T5 generates a decent proportion of relevant bigrams.

- The F1-score is 0.183, reflecting a small but noticeable improvement over both TextRank and Word2Vec. It suggests that T5 does a better job of generating relevant summaries.

3. ROUGE-L

   - The recall is 38.0%, which is better than both Word2Vec and TextRank. This shows that T5 is capturing a larger proportion of the longest common subsequences between the model-generated and reference summaries.
   - The precision is 36.5%, which is higher than the recall but still not optimal. This suggests that T5 is generating relatively relevant subsequences, but some of the sequences are not as relevant as they could be.
   - The F1-score is 0.363, indicating a balanced performance between recall and precision and thus T5 performs better than the other models in terms of ROUGE-L.

7 Comparison and Discussion

Comparing the three models by analysing their ROUGE scores and specifically looking into their recall precision and F1 scores helps to know how each model performed for the text summarization task and how well was the research question analyzed and dealt with.

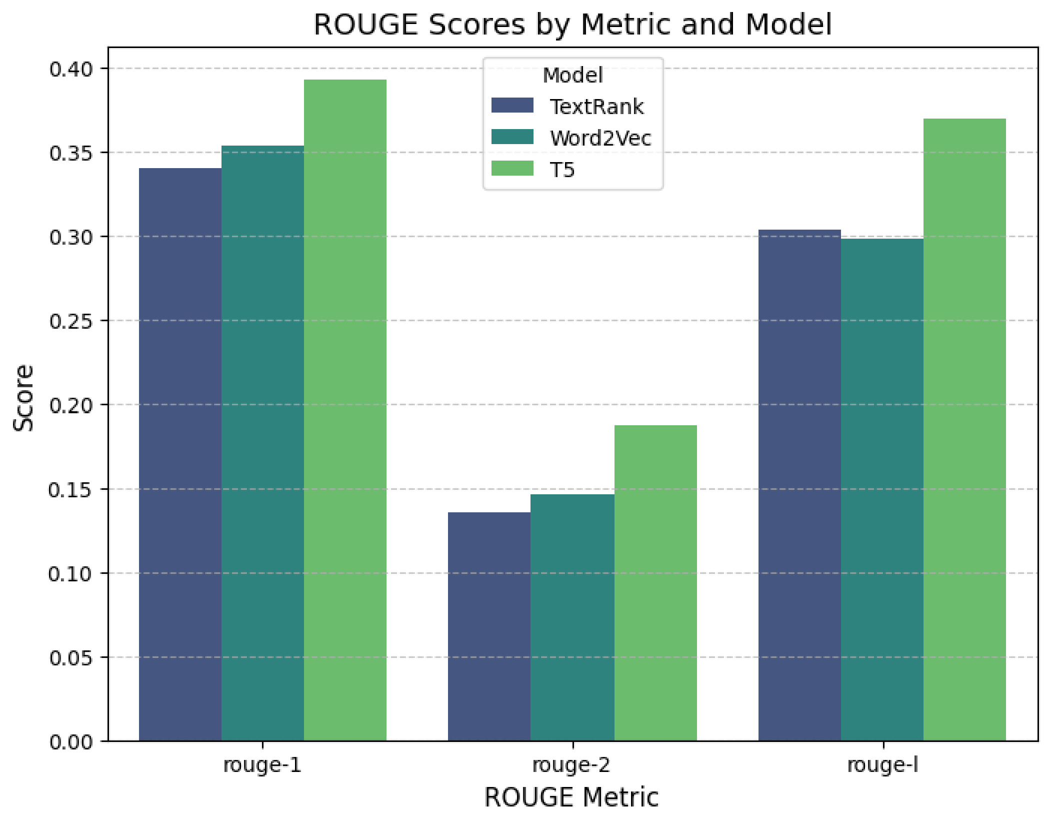

Figure 8: Model comparison against ROUGE metrics using ROUGE scores

As shown in the Figure 8, T5 outperforms in all ROUGE metrics ie, ROUGE-1, ROUGE-2, and ROUGE-L, compared to TextRank and Word2Vec. This implies that T5 is better at producing summaries that closely match the reference summaries regarding word overlap, n-gram overlap, and longest common subsequence.

Analyzing the ROUGE-1 scores, as shown in Figure 9, T5 model performs the best in recall, precision and F1 over the other models, indicating that it generates more comprehensive and accurate summaries.

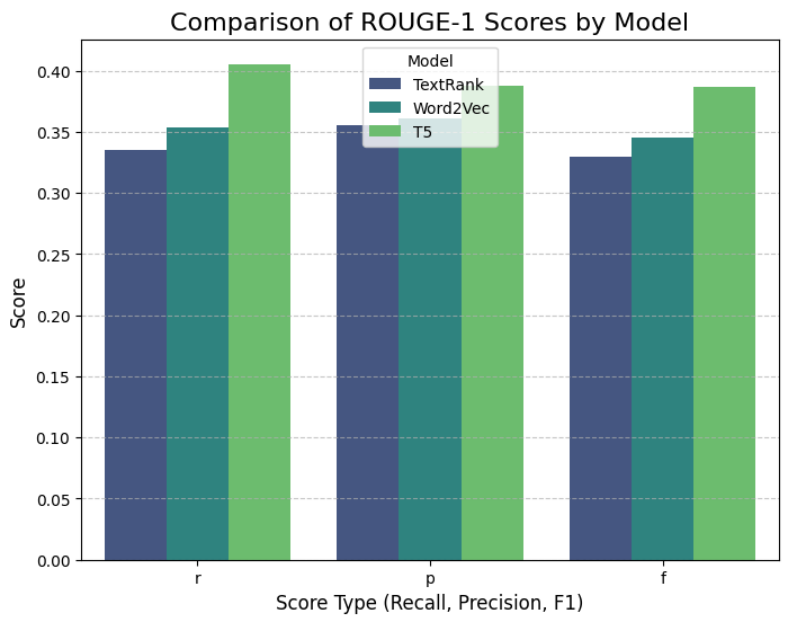

Figure 9: Comparison of ROUGE-1 scores (recall(r), precision(p) and F1-score by model

Looking into Figure 10, T5's ability to capture fine-grained details and phrase-level information is notably superior, with a significant gap over TextRank and Word2Vec.

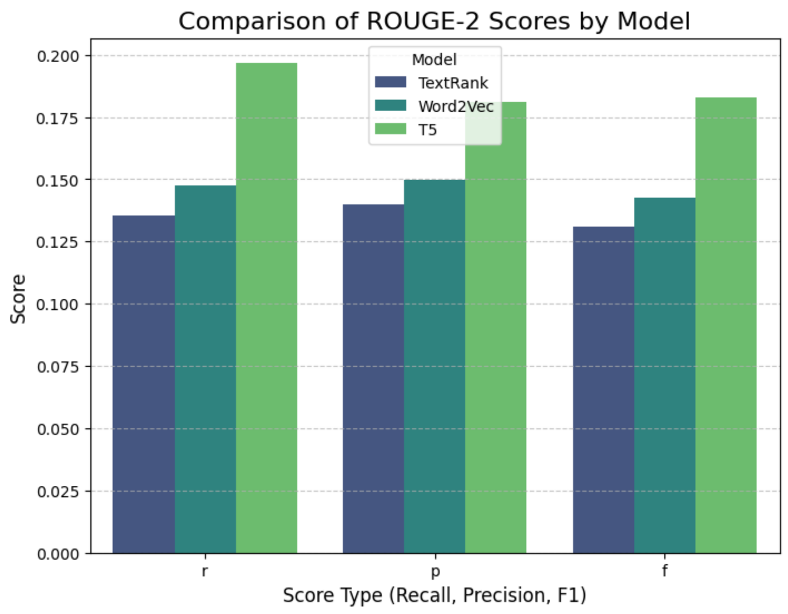

Figure 10: Comparison of ROUGE-2 scores (recall(r), precision(p) and F1-score by model

Finally analyzing Figure 11, it is clear that T5 excels at preserving the structure and coherence of the generated summaries, outperforming the other two models in ROUGE-L metrics.

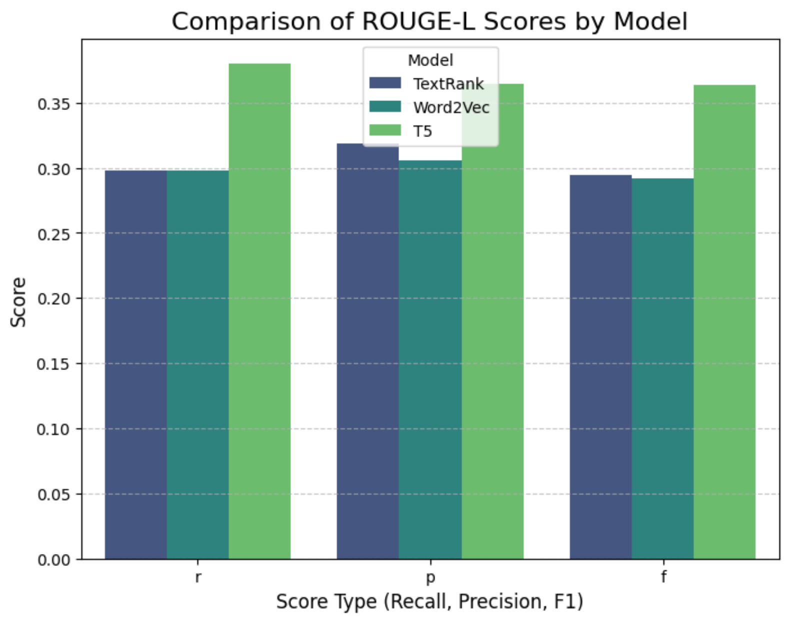

Figure 11: Comparison of ROUGE-L scores (recall(r), precision(p) and F1-score by model

From the comparative plots, it is clear and conclusive that T5's strong transformer architecture and extensive dataset pre-training are responsible for its exceptional performance on all metrics. T5 produces informative and fluid summaries that closely resemble summaries written by humans after learning complex language representations.

7.1 Time Consumption

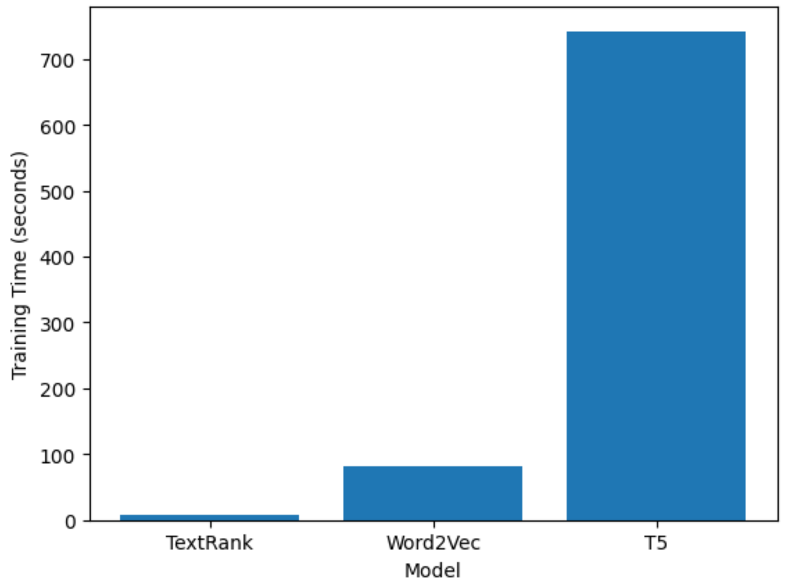

Figure 12: Comparison between models on training time

Comparing the computational efficiency and time for training, as shown in Figure12, TextRank is the most efficient model, requiring significantly less training time with minimal computational complexity.

Word2Vec has moderate complexity, requiring more computation than TextRank due to the training of word embeddings.T5 is the most complex model, with the transformer architecture and a large number of parameters, leading to high computational demands.

Model	TextRank	Word2Vec	T5
Time taken(sec)	7.57	81.69	742.45

Table 4: Computational time taken by each model in seconds

The computation trade-offs between TextRank, Word2Vec, and T5 are evident from the training times shown in Table 4. But T5, despite its high computational cost, often achieves superior summarization performance compared to TextRank and Word2Vec. This highlights the trade-off between efficiency and performance.

8 Limitations and Future work

8.1 Limitations

- A smaller subset of the CNN/Daily Mail dataset was used in the study because of hardware constraints, which may have limited how broadly the findings can be applied.
- T5's high computational cost restricted the use of larger models or larger datasets.
- Although T5 outperformed traditional models, it required significantly more time and resources to train, which may not be feasible in real-time or resource-constrained environments.
- The CNN/Daily Mail dataset mainly contains news articles, which may not generalize well to other domains such as medical, legal, or technical documents.

8.2 Future Work:

- In order to enhance model performance and generalisability, future research could make use of the entire CNN/Daily Mail dataset, which would require more computational resources.
- Larger T5 variations, like T5-base or T5-large, could be experimented with to produce more accurate and cohesive summaries.
- Better results may be obtained by combining the advantages of abstractive models (like T5) and extractive models (like Word2Vec) by first picking key sentences and then using transformers to refine them.
- To evaluate the model's performance and adaptability across domains, expand the study to include additional datasets (such as financial, medical, or legal documents).

9 Conclusion

According to this study, transformer-based models—more especially, T5—perform better on text summarisation tasks than conventional techniques like TextRank and Word2Vec, achieving higher recall, precision, and F1-scores across all ROUGE metrics. The revolutionary potential of NLP's attention mechanisms is highlighted by T5's capacity to comprehend contextual subtleties and produce logical, human-like summaries. However, T5's higher computational cost comes with a hefty price tag, which makes Word2Vec and TextRank good substitutes for resource-constrained environments.

In this study only a small subset of the whole training data available was used for the study due the computational limitations. T5 is the recommended choice for high-precision applications, while traditional models are workable options for situations with limited resources. In the end, the choice of model is determined by the trade-off between accuracy and computational efficiency.

References

Barbella, M. & Tortora, G. (2022), 'Rouge metric evaluation for text summarization techniques', *Available at SSRN 4120317* .
URL: *https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4120317*

Gulati, V., Kumar, D., Popescu, D. E. & Hemanth, J. D. (2023), 'Extractive article summarization using integrated textrank and bm25+ algorithm', *Electronics* **12**(2), 372.
URL: *https://www.researchgate.net/publication/367067790*

Karjule, V., Dange, J., Thange, S., Sase, J. & Kokate, P. (2023), 'A survey on text summarization techniques', *Journal of Natural Language Processing* .
URL: *https://www.ijsrp.org/research-paper-1123.php?rp=P14313170*

Mihalcea, R. & Tarau, P. (2004), Textrank: Bringing order into text, *in* 'Proceedings of the 2004 conference on empirical methods in natural language processing', pp. 404–411.
URL: *https://www.researchgate.net/publication/200042361*

Ravichandran, R., Sharma, S. B. P., Arkal, S. S., Das, S. & Nagarajan, S. (2023), 'Text summarization using the t5 transformer model', *International Research Journal of Engineering and Technology (IRJET)* **10**(08), 896. ISO 9001:2008 Certified Journal, Impact Factor: 8.226.
URL: *https://www.irjet.net/archives/V10/i8/IRJET-V10I8148.pdf*

Vaswani, A. (2017), 'Attention is all you need', *Advances in Neural Information Processing Systems* .
URL: *https://arxiv.org/abs/1706.03762*

Wang, Z., Ma, L. & Zhang, Y. (2016), A novel method for document summarization using word2vec, *in* '2016 IEEE 15th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)', IEEE, pp. 523–529.
URL: *https://www.researchgate.net/publication/313963415*

Yadav, D., Desai, J. & Yadav, A. K. (2022), 'Automatic text summarization methods: A comprehensive review'.
URL: *https://arxiv.org/abs/2204.01849*

Zaware, S., Patadiya, D., Gaikwad, A., Gulhane, S. & Thakare, A. (2021), Text summarization using tf-idf and textrank algorithm, *in* '2021 5th International conference on trends in electronics and informatics (ICOEI)', IEEE, pp. 1399–1407.
URL: *https://www.researchgate.net/publication/352600769_Text_Summarization_using_TF-IDF_and_Textrank_algorithm*

A Appendix

```
# Install necessary libraries
    !pip install rouge
    !pip install datasets

    # import necessary libraries
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
    from wordcloud import WordCloud
    import time
    import nltk
    from nltk.tokenize import sent_tokenize, word_tokenize
    from nltk.corpus import stopwords
    import re
    from sklearn.feature_extraction.text import TfidfVectorizer
    import networkx as nx
    from sklearn.metrics.pairwise import cosine_similarity
    from rouge import Rouge
    from nltk.stem import WordNetLemmatizer
    from gensim.models import Word2Vec
    from tqdm import tqdm
    from datasets import Dataset,load_dataset
    from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
    from transformers import DataCollatorForSeq2Seq
    from transformers import Seq2SeqTrainingArguments, Seq2SeqTrainer

    # download necessary packages
    nltk.download(’punkt’)
    nltk.download(’stopwords’)
    nltk.download(’punkt_tab’)
    nltk.download(’wordnet’)

    # Load the CSV files
    test_df = pd.read_csv(’/content/drive/MyDrive/Project/test_data.csv’)
    train_df = pd.read_csv(’/content/drive/MyDrive/Project/train_data.csv’)
    validation_df = pd.read_csv( ’/content/drive/MyDrive/Project/validation_data.csv’)

    # select sample subsets from train, test and validation
    train_subset = train_df.sample(n=5000,random_state=26)
    test_subset = test_df.sample(n=1000,random_state=26)
    val_subset = validation_df.sample(n=100,random_state=26)

    Exploratory Data Analysis

    # Check the shape of each dataset
```

```
print(f”Train set: train_df.shape”)
print(f”Test set: test_df.shape”)
print(f”Validation set: validation_df.shape”)

# Check for the basic informations and null values
print(train_subset.info())

# information about the test subset
print(test_subset.info())

# information about the validation subset
print(val_subset.info())

print(train_subset.head())

# length of words in article and summary
train_subset[’article_length’] = train_subset[’article’].apply( lambda x: len(x.split()))
train_subset[’summary_length’] = train_subset[’highlights’].apply( lambda x: len(x.split()))

# Visualization of article length distribution
sns.histplot(train_subset[’article_length’], bins=50, kde=True)
plt.show()

# Visualization of summary length distribution
sns.histplot(train_subset[’summary_length’], bins=50, kde=True)
plt.show()

# Summary statistics
summary_stats = train_subset[’summary_length’].describe()
print(summary_stats)

# setup the stopwords and lemmatizer
stop_words = set(stopwords.words(’english’))
lemmatizer = WordNetLemmatizer()

# Wordcloud of the train subset
wordcloud = WordCloud(stopwords=stop_words, background_color=”white”).generate(
” ”.join(train_subset[’article’]))
plt.imshow(wordcloud, interpolation=’bilinear’)
plt.axis(”off”)
plt.show()
```

TextRank

Data Preprocessing for TextRank

```
def preprocess_text_tr(text):
”””
```

```
Preprocesses text for TextRank model
Args:
text (str): input text to be preprocessed
Returns:
str: cleaned and preprocessed text
”””
# Convert text to lowercase
text = text.lower()
# Remove special characters, numbers, and extra spaces
text = re.sub(r”,’ ’,text)
# tokenize sentences and words
words = word_tokenize(text)
# Remove stopwords and perform lemmatization
processed_words = [lemmatizer.lemmatize(word) for word in words if word not in stop_words]
# Return cleaned text as a single string
return ’ ’.join(processed_words)

# Function to split text into sentences
def split_into_sentences(text):
”””
Splits text into sentences
Args:
text (str): input text to be split
Returns:
list: list of sentences
”””
return sent_tokenize(text)

# convert the validation summaries to a list
val_summaries = val_subset[’highlights’].tolist()
```

TextRank Model Implementation

```
# TextRank Summarization Function
def textrank_summarizer(text, top_n=2):
”””
Performs text summarization using TextRank algorithm
Args:
text (str): input text to be summarized
top_n (int): number of sentences to include in the summary
Returns:
str: summarized text
””” # Preprocess and split text into sentences
sentences = split_into_sentences(text)
# If there is only one sentence, return it as the summary
if len(sentences) ¡= 1:
return text
# Preprocess each sentence
cleaned_sentences = [preprocess_text_tr(sentence) for sentence in sentences]
```

```
# Vectorize the sentences using TF-IDF
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(cleaned_sentences)
# Compute cosine similarity matrix
similarity_matrix = cosine_similarity(tfidf_matrix, tfidf_matrix)
# graph is created from similarity matrix
nx_graph = nx.from_numpy_array(similarity_matrix)
#calculate sentence scores using PageRank algorithm
scores = nx.pagerank(nx_graph)
# Rank the sentences based on their scores
ranked_sentences = sorted(((
scores[i], s) for i, s in enumerate(sentences)), reverse=True)
# Select the top N sentences for the summary
summary = " ".join([ranked_sentences[i][1] for i in range( min(top_n, len(ranked_sentences)))])
return summary
```

TextRank Evaluation

```
# initialize time
start_time = time.time()
# Run summarization on validation set for initial evaluation
textrank_val_summaries = [textrank_summarizer(article) for article in val_subset['article']]
# end the time
end_time = time.time()
# get the time taken for textrank summarization
tR_time = end_time - start_time
print(f"Elapsed time: elapsed_time:.2f seconds")

# calculate the ROUGE scores for textrank
rouge = Rouge()
scores = rouge.get_scores(textrank_val_summaries,val_summaries,avg=True)
print("TextRank ROUGE Scores:")
scores
```

Word2Vec

Data Preprocessing for Word2Vec

```
def preprocess_text_wv(text):
"""
Preprocesses text for Word2Vec model
Args:
text (str): input text to be preprocessed
Returns:
list: list of preprocessed sentences
"""
# Basic text cleaning
text=re.sub(r"',' ', text)
```

```
# Sentence tokenization
sentences = sent_tokenize(text)
processed_sentences = []
# Word tokenization and stopword removal
for sentence in sentences:
words = word_tokenize(sentence.lower())
words = [lemmatizer.lemmatize( word) for word in words if word not in stop_words]
processed_sentences.append(words)
return sentences, processed_sentences

# Preprocess articles and tokenize into sentences using Word2Vec preprocessing function
processed_w2v = [preprocess_text_wv( article) for article in tqdm(train_subset['article'])]

# Flatten and collect tokenized sentences from the processed data
tokenized_sentences = [tokens for _, tokenized in processed_w2v for tokens in tokenized]
```

Word2Vec Training and Implementation

```
start_time = time.time()
# Train Word2Vec model with skip-gram on tokenized sentences
w_model = Word2Vec(sentences=tokenized_sentences, vector_size=50, window=5,min_count=5,workers=5,
sg=1)
end_time = time.time()
w2v_time = end_time - start_time
print(f”Elapsed time: elapsed_time:.2f seconds”)

def sentence_to_vector(sentence, model):
”””
Converts a sentence to a vector by averaging word vectors.
Args:
sentence (str): input sentence to be converted
model : w_model
Returns:
numpy.ndarray: vector representation of the sentence
”””
words = preprocess_text_wv(sentence)
# Flatten the list if ‘preprocess_text‘ returns a list of lists
if isinstance(words[0], list): # check if words is list of lists
words = [word for sublist in words for word in sublist] # flatten it
word_vectors = []

# Get word vectors for words in the sentence
for word in words:
# Convert the word to string before checking if it’s in the vocabulary
if isinstance(word, list): # Check if word is a list
word = ’ ’.join(word) # Convert the list to string
# Check if the word is in the model’s vocabulary
if word in model.wv.key_to_index:
```

```
word_vectors.append(model.wv[word])
# Return the average word vector for the sentence
if word_vectors:
return np.mean(word_vectors, axis=0)
else:
# If no word vectors found, return a zero vector
return np.zeros(model.vector_size)

def calculate_sentence_similarity(sentences, model):
”””
Calculates cosine similarity between sentences.
Args:
sentences (list): list of sentences
model : w_model
Returns:
numpy.ndarray: similarity matrix
”””
# Convert each sentence to its vector representation
sentence_vectors = np.array( [sentence_to_vector(sentence, model) for sentence in sentences])
# Compute the cosine similarity matrix
similarity_matrix = cosine_similarity(sentence_vectors)
return similarity_matrix

def extractive_summary(text, model, top_n=2):
”””
Generates an extractive summary by selecting top n sentences based on cosine similarity.
Args:
text (str): input text to be summarized
model : w_model
top_n (int): number of sentences to include in the summary
Returns:
str: extractive summary
”””
sentences = text.split('.') # Split text into sentences
similarity_matrix = calculate_sentence_similarity(sentences, model)
# Get similarity scores for each sentence
similarity_scores = similarity_matrix[0] # Assuming first sentence as reference
# Rank sentences by similarity scores
sorted_similarities = sorted(enumerate( similarity_scores), key=lambda x: x[1], reverse=True)
# Extract top N sentences
top_sentences = [sentences[idx] for idx, _ in sorted_similarities[:top_n]]
return ' '.join(top_sentences)
```

Word2Vec Evaluation

```
#evaluate using validation data
validation_summaries_wv = []
```

```
for idx, row in val_subset.iterrows():
text = row[’article’]
# Extract top 2 sentences for the summary
summary = extractive_summary(text, w_model, top_n=2)
validation_summaries_wv.append(summary)

# Add the summary to the validation DataFrame
val_subset[’summary_wv’] = validation_summaries_wv

# Convert word2vec summaries to a list format
w2v_val_summaries = val_subset[’summary_wv’].tolist()

# Calculate the rouge scores
w2v_scores = rouge.get_scores(w2v_val_summaries, val_summaries,avg=True)
w2v_scores
```

T5

Data Preprocessing for T5

```
# Convert pandas DataFrame to Hugging Face Dataset
train_dataset = Dataset.from_pandas(train_subset)
test_dataset = Dataset.from_pandas(test_subset)

def preprocess_data_t5(data, tokenizer, max_input_length=512, max_target_length=128):
”””
Preprocesses input data for T5 model by tokenizing articles and summaries.
Args:
data (pd.DataFrame): Data containing ’article’ and ’highlights’ columns.
tokenizer (transformers.AutoTokenizer): Tokenizer for the T5 model.
max_input_length (int): Maximum token length for input articles. Default is 512.
max_target_length (int): Maximum token length for target summaries. Default is 128.
Returns:
model_inputs : Tokenized inputs and labels for model training.
”””
# Add the ’summarize:’ prefix to each article to indicate the task type
inputs = [”summarize: ” + doc for doc in data[”article”]]
# Get the target summaries
targets = data[’highlights’]
# Tokenize the input articles
model_inputs = tokenizer(inputs, max_length=max_input_length, truncation=True)
# Tokenize the target summaries
labels = tokenizer(targets, max_length=max_target_length, truncation=True)
# Assign the tokenized target summaries as labels for the model
model_inputs[”labels”] = labels[”input_ids”]
# Return the processed inputs and labels
return model_inputs
```

```
# Load tokenizer and model
model_name = ”t5-small”
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)

# preprocess and tokenize datasets
train_dataset = train_dataset.map( lambda x: preprocess_data_t5(x, tokenizer), batched=True)
test_dataset = test_dataset.map( lambda x: preprocess_data_t5(x, tokenizer), batched=True)

# Define the list of columns to be included in the PyTorch format
columns = [’input_ids’, ’attention_mask’, ’labels’]

# Convert the train dataset into a PyTorch-compatible format
#by setting the specified columns as torch tensors
train_dataset.set_format(type=”torch”, columns=columns)

# Convert the test dataset into a PyTorch-compatible format
# by setting the specified columns as torch tensors
test_dataset.set_format(type=”torch”, columns=columns)
```

T5 Initialization and Fine-Tuning

```
# Define data collator
data_collator = DataCollatorForSeq2Seq(
tokenizer=tokenizer,
model=model_name)

# Define the training arguments for fine-tuning the model
training_args = Seq2SeqTrainingArguments(
output_dir=”/content/drive/MyDrive/Project/t5-small-summarizer_checkpoints”,
#Directory to save model checkpoints and results
eval_strategy=”epoch”, # Evaluation strategy: perform evaluation at the end of
each training epoch
learning_rate=2e-5, # Learning rate for the optimizer
per_device_train_batch_size=18, # Batch size for training on each device
per_device_eval_batch_size=18, # Batch size for evaluation on each device
num_train_epochs=5, # Number of times to go through the entire training dataset
save_steps=500, # Number of steps between model checkpoint saves
save_total_limit=2, # Limit the number of saved checkpoints to avoid excessive storage
usage
predict_with_generate=True, # Whether to use the model’s text generation functionality
for predictions
fp16=True, # Whether to use mixed precision training (use half-precision floating point
format) to reduce memory usage and speed up training
weight_decay=0.01, # Weight decay (L2 regularization) to prevent overfitting
metric_for_best_model=”loss”) # The metric used to determine the best model (based
on the lowest loss)

# Initialize Trainer
```

```
trainer = Seq2SeqTrainer(
model=model,
args=training_args,
train_dataset=train_dataset,
eval_dataset=test_dataset,
tokenizer=tokenizer,
data_collator=data_collator)

start_time = time.time()
# train the t5 model
trainer.train()
end_time = time.time()
t5_time = end_time - start_time
print(f”Elapsed time: elapsed_time:.2f seconds”)
```

T5 Evaluation

```
# load the checkpoint
model_name=’/content/drive/MyDrive/Project/t5-small-summarizer_checkpoints/checkpoint-
1390’
#load tokenizer
tokenizer = AutoTokenizer.from_pretrained(model_name)
#load model
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)

# Initialize a list to store generated summaries and highlights
generated_summaries = []
# Iterate through each article in the validation dataset
for _, row in val_subset.iterrows():
article = row[’article’]
inputs = tokenizer(article, max_length=512, truncation=True, return_tensors=’pt’)
summary_ids = model.generate(**inputs,
max_length=150,
num_beams=5,
do_sample=True,
temperature=1.2,
top_k=100,
top_p=0.95,
repetition_penalty=1.1,
early_stopping=True)
summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)
generated_summaries.append(summary)

gen_summaries = generated_summaries

# calculate the rouge scores for t5 model
t5_scores=rouge.get_scores(gen_summaries,val_summaries,avg=True)
```

```
t5_scores

val_subset.iloc[10]['article']
gen_summaries[10]
val_summaries[10]
```

Comparison and Anlaysis

```
# Combine all scores into a single dictionary
rouge_scores =  'TextRank': scores,
'Word2Vec': w2v_scores,
'T5': t5_scores
# Print the combined dictionary
print(rouge_scores)

# Convert to DataFrame
df_results = pd.DataFrame.from_dict((model, metric): values
for model, metrics in rouge_scores.items()
for metric, values in metrics.items())
df_results = df_results.transpose()
df_results.reset_index(inplace=True)
df_results.columns = ['Model', 'Metric', 'Recall', 'Precision', 'F1-Score']
df_results

# Flattening the data
rows = []
for models, metrics in rouge_scores.items():
for metric, scores in metrics.items():
for score_type, value in scores.items():
rows.append(
'Model': models,
'Metric': metric,
'Score Type': score_type,
'Value': value)
# Creating the DataFrame
df_plot = pd.DataFrame(rows)

plt.figure(figsize=(8, 6))
sns.barplot(data=df_plot, x='Metric', y='Value', hue='Model', errorbar=None, palette='viridis')
plt.title('ROUGE Scores by Metric and Model', fontsize=14)
plt.ylabel('Score', fontsize=12)
plt.xlabel('ROUGE Metric', fontsize=12)
plt.legend(title='Model')
plt.grid(axis='y', linestyle='–', alpha=0.7)
plt.show()

# Separate the data by Metric and plot each as a bar plot
metrics = ['rouge-1', 'rouge-2', 'rouge-l']
for metric in metrics:
plt.figure(figsize=(8, 6))
```

```
    sns.barplot(data=df[df[’Metric’] == metric], x=’Score Type’, y=’Value’, hue=’Model’,
palette=’viridis’)
    plt.title(f’Comparison of metric.upper() Scores by Model’, fontsize=16)
    plt.ylabel(’Score’, fontsize=12)
    plt.xlabel(’Score Type (Recall, Precision, F1)’, fontsize=12)
    plt.legend(title=’Model’, loc=’upper center’)
    plt.grid(axis=’y’, linestyle=’–’, alpha=0.7)
    plt.show()

    # Define the data
    model = [’TextRank’, ’Word2Vec’, ’T5’]
    training_time = [7.57, 81.69, 742.45]
    # Create the bar plot
    plt.bar(model, training_time)
    # Set the title and labels
    plt.xlabel(’Model’)
    plt.ylabel(’Training Time (seconds)’)
    # Display the plot
    plt.show()
```