



Information Retrieval and Web Analytics –
IT3041

End-to-End Text Summarization and Analysis
System
Report

	Name	Batch subgroup	Registration Number	Email
1	Thilagaratnam Madhuwarsha	Y3.S1.WE.01.01	IT22364760	it22364760@my.sliit.lk
2	I.M.M.Mujahid	Y3.S1.WE.01.02	IT22266750	it22266750@my.sliit.lk
3	Fernando.S.K	Y3.S1.WE.01.02	IT22129680	it22129680@my.sliit.lk
4	Navashanthan. T	Y3.S1.WE.01.02	IT22274984	it22274984@my.sliit.lk

Github Link - <https://github.com/IT22274984/summarization.git>

Submission date – 04.10.2024

Table of Contents



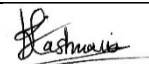

Declaration	4
1. Abstract.....	5
2. Introduction.....	6
3. Problem Definition	7
Data Sources.....	7
4. Exploratory Data Analysis	8
Distribution of Article and Summary Length	8
Box Plot of Summary and Article Length	9
Word cloud of Summaries and Articles	10
Methods Used.....	11
5. System Design and Architecture	12
T5 Transformer	14
Technology Stack:	14
Huggingface	15
6. Implementation.....	17
• Backend Development	17
• Frontend Development.....	18
Key Elements of the User Interface:	18
Key UI Features and User Experience Considerations:	19
7. Evaluation and Testing.....	24
• Performance Evaluation.....	24
Rouge Performance on Test Set.....	26
Experimental Results and Model Comparison.....	27
8. Challenges faced and solutions	28
9. Conclusion and Future work	30
Limitations:.....	30

Future Work:	30
10. Appendix	32
11. UI Page	38
12. References	41

Declaration

We hereby declare that this project report, or any part thereof, is an original work and has not been submitted or copied, either wholly or in part, from any other individual, group, organization, university, or institution, including SLIIT, for academic or non-academic purposes. Furthermore, we affirm that this report does not contain any material directly copied from the internet or any other external sources, unless appropriately cited and referenced according to standard academic guidelines.

We also confirm that no part of this project has been previously submitted by any student or project group to SLIIT or any other academic institution for the fulfillment of any degree, diploma, or certificate. This project is the result of our independent effort, and any assistance, ideas, or data from other sources have been acknowledged appropriately in the text.

Registration Number	Name	Signature
IT22364760	Thilagaratnam Madhuwarsha	
IT22266750	I.M.M.Mujahid	
IT22129680	Fernando.S.K	
IT22274984	Navashanthan. T	

1. Abstract

In today's fast-paced digital world, the exponential growth of information, especially in the form of news articles, necessitates effective methods for content summarization and analysis. This project presents a comprehensive **news summarization and analysis system** that integrates natural language processing (NLP) techniques to facilitate the extraction of key insights from large textual data. The system employs both extractive and abstractive summarization methods, leveraging a fine-tuned **T5 model** for generating coherent summaries while providing additional functionalities such as sentiment analysis, keyword extraction, and topic modeling. Built using **Streamlit**, the user-friendly web interface allows users to input articles and customize summarization parameters dynamically, ensuring an interactive experience. Despite certain limitations regarding long text processing and summary coherence, the project demonstrates a robust framework for summarizing news articles, offering significant potential for future enhancements and applications in information retrieval and content analysis.

2. Introduction

The rapid proliferation of news content across various media platforms presents both opportunities and challenges for consumers and content creators alike. As information overload becomes a common issue, individuals often struggle to keep up with the sheer volume of news articles, making it imperative to develop effective tools for content summarization. Traditional reading methods may be inefficient, and manual summarization can be time-consuming, prompting the need for automated solutions that can distill essential information rapidly.

Natural Language Processing (NLP) has emerged as a pivotal technology in addressing these challenges, providing methods to analyze, summarize, and interpret text data efficiently. This project aims to create an end-to-end **news summarization and analysis system** that harnesses various NLP techniques, enabling users to derive insights from articles quickly. By integrating both extractive and abstractive summarization methods, the system offers a dual approach to generating summaries that retain critical information while presenting it in a concise manner.

The system is designed with user experience in mind, employing **Streamlit** to provide an intuitive interface that allows users to input articles, configure summarization parameters, and view results in real-time. In addition to summarization capabilities, the system includes sentiment analysis to gauge the emotional tone of articles, keyword extraction for identifying significant terms, and topic modeling for understanding broader themes within the text.

In conclusion, this project aims to bridge the gap between the overwhelming amount of news available and the need for efficient information consumption, employing state-of-the-art NLP techniques to achieve this goal.

3. Problem Definition

Given a fresh news article as input, the system will generate a text summary of it. This project will involve exploring extractive and abstractive text summarization techniques. Extractive technique generates summary using important sentences from the original text, whereas abstractive technique summarizes most important information in a new way using advanced natural language processing.

Data Sources

The CNN / Dailymail News Dataset, which contains news stories and is available on both Papers with Code and Kaggle, will be used for the research.

Each article has the article body and a human generated summary. The dataset was collected for a question and answering system, and can also be used to create a text summarization system.

Dataset Link: <https://paperswithcode.com/dataset/cnn-daily-mail-1>

4. Exploratory Data Analysis

The data is divided into training, validation and test sets. Each article has a human generated summary associated with it.

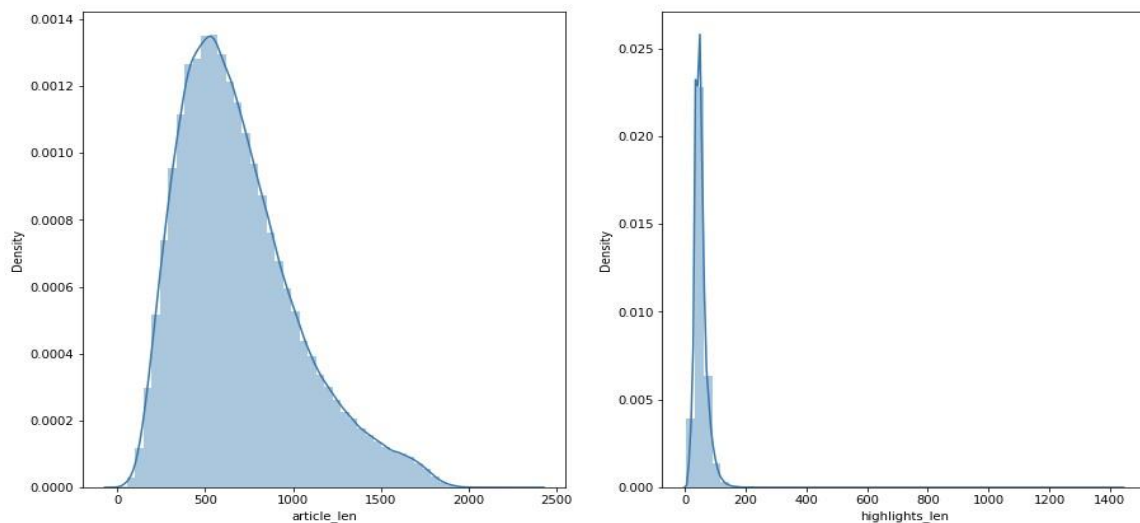
- Training set: 287K articles
- Validation set: 13K articles
- Test set: 11K articles
- Fields:
- ID : Unique ID of article
- Article: News article body
- Highlights: News article summary

Here is the snapshot of the data showcasing all fields:

	id	article	highlights
0	fef38572c5452c1a79af7cfba24ac0ead40c0ba	By . Joshua Gardner . and Ap Reporter . A Cal...	Robert Cox of Manteca, California was just pas...
1	d222143d15cc93b65d7f76500e878a290f9b85a4	Sinister: David Russell lured his 19-year-old ...	David Russell has been jailed for life for kid...
2	cc1af28a940a9d8fe9fa2dd2f24dd9b9cc916a43	By . Victoria Woollaston . and Jonathan O'Call...	Using modern dating techniques, \na Cambridge...
3	7a17330f65227aa1155ec9fab9972dd35d207612	(CNN)The first indication that something was w...	All three girls skipped school and took a flig...
4	80a774435cfe17b62d5103733378d0a94b4663d5	The Islamic State terror group has issued batt...	Terror group has published safety advise for i...

Distribution of Article and Summary Length

Doing analysis of the article and summary length will help in deciding model parameters. For this purpose, the distribution of article and summary length can be examined.

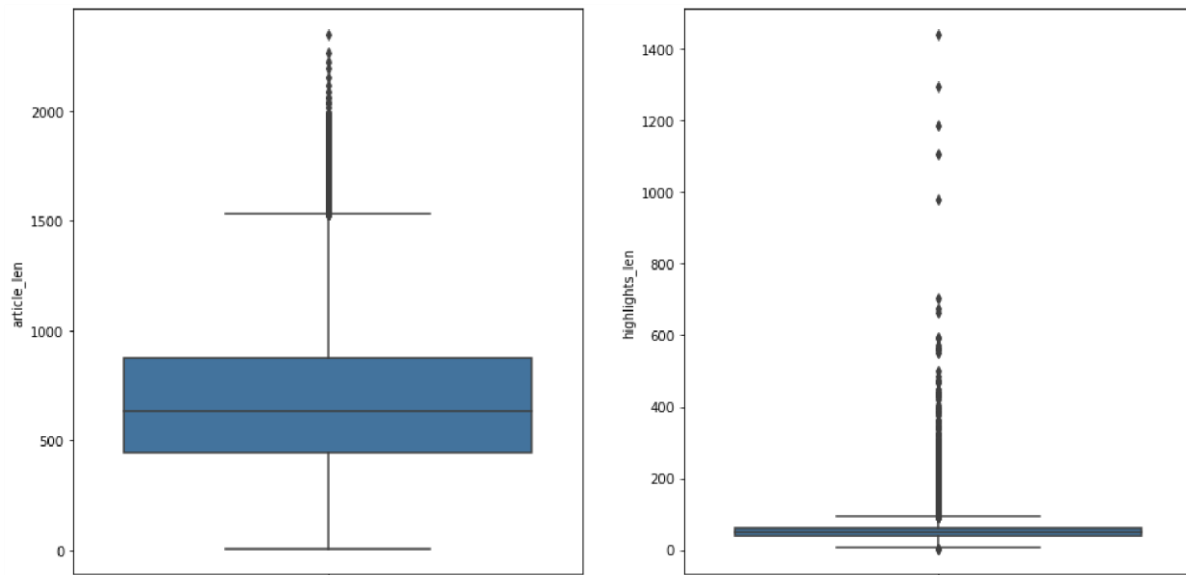


From the length distributions we can see that articles have a median length of 700 words and summaries have a median length of 60.

Both the distributions are right skewed, indicating that most of the articles have length less than or equal to 700, with a few articles having length greater than 1600 words.

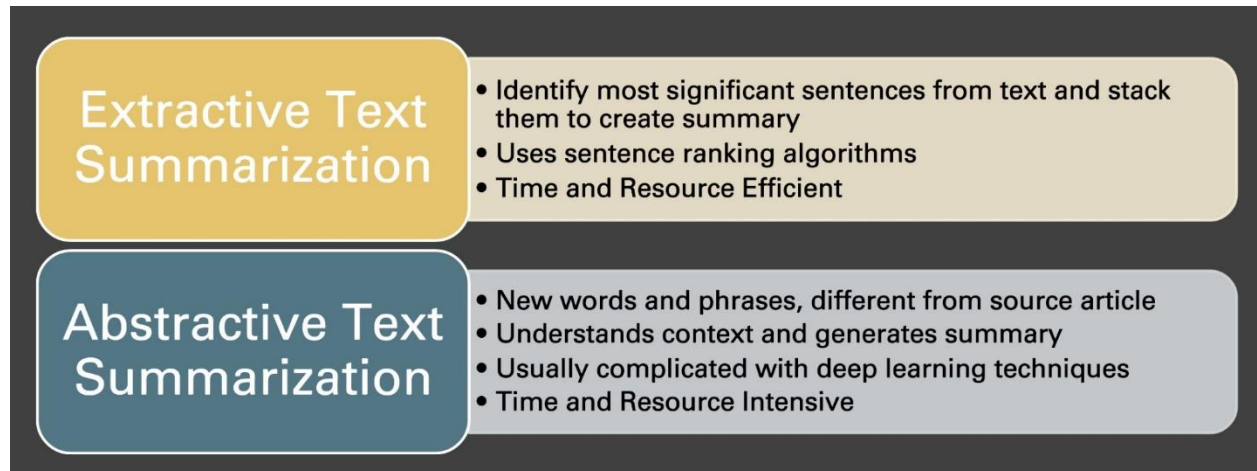
Box Plot of Summary and Article Length

Summary lengths and article lengths both have a few outliers, indicating few articles have longer length and corresponding longer summaries. This can be verified with the below box plots. Both indicate that outliers are towards the higher word length side.



Methods Used

Text summarization can be done using two ways: Extractive and Abstractive summarization.



In extractive summarization, we identify important sentences from the article and make a summary by selecting the most important sentences.

Whereas, for abstractive summarization the model understands the context and generates a summary with the important points with new phrases and language. Abstractive summarization is more similar to the way a human summarizes any content. A person might read the entire document, remember a few key points and while writing the summary, will make new sentences that include these points. Abstractive summarization follows the same concept.

5. System Design and Architecture

The system is designed as a modular, multi-component structure, providing an interactive platform for summarizing and analyzing news articles. The key components are divided into the frontend, backend, and (potentially) the database, working together to perform complex natural language processing (NLP) tasks.

1. Frontend (User Interface):

- a. The frontend is built using **Streamlit**, a Python-based framework for creating web applications with minimal effort. Streamlit allows for the rapid development of an interactive interface that provides a smooth user experience. Users can submit news articles, adjust parameters (e.g., summary length, keywords), and immediately see the results, all within a single-page web application.
- b. **Input Widgets:** Users can input a news article using a text area widget. On the right side, users can configure various summarization options through **number input fields**. They can adjust the length of the extractive summary by specifying the number of sentences, and for abstractive summarization, they can set both the **maximum and minimum word limits**. These options are neatly arranged in a sidebar, providing a clean and intuitive interface for easy customization.
- c. **Display Elements:** The results (summaries, sentiment analysis, keywords, topics) are displayed in a structured, user-friendly format, making it easy for the user to interpret. Additional images (handled using **PIL**) enhance the interface visually, contributing to an intuitive, aesthetically pleasing design.

2. Backend (Core Processing and Computation):

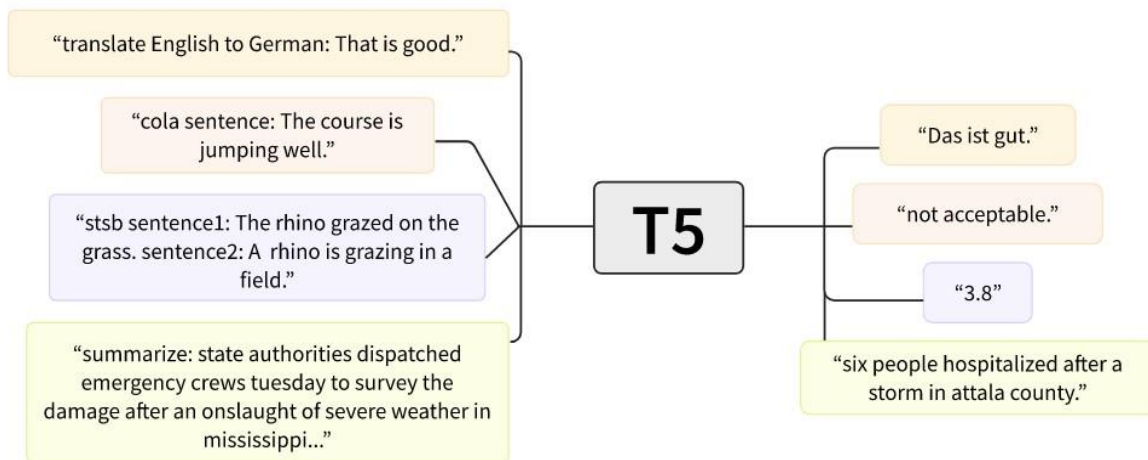
- a. The backend is implemented in **Python**, utilizing several specialized NLP libraries. The backend is responsible for processing the input news articles and generating results like summaries, keywords, topics, and sentiment scores. The modular design makes the backend easily scalable for future additions or improvements.
- b. **Preprocessing Pipeline:** The **Preprocess** class handles all text preprocessing tasks such as converting text to lowercase, tokenizing it into sentences and words, removing stopwords, and lemmatizing words. This cleaned and tokenized text is essential for downstream tasks like summarization and keyword extraction. For tokenization and lemmatization, it uses **NLTK** (Natural Language Toolkit), a comprehensive library for text processing.

- c. **Extractive Summarization:** The extractive summarization is implemented in the **NewsSummarization** class. This method selects important sentences from the article based on sentence scoring. Sentence scoring is achieved by calculating word frequencies after preprocessing, where more frequent words are given higher importance. The system uses **heapq** to extract the top-N sentences, which are then combined into a coherent summary. This method works well for quickly identifying the most important parts of an article.
- d. **Abstractive Summarization:** For abstractive summarization, the system leverages a **Hugging Face Transformers** pipeline, specifically the pre-trained **T5 model**. Unlike extractive summarization, this method generates new sentences by understanding the underlying meaning of the text, producing more human-like and concise summaries. The user can specify parameters such as maximum and minimum word limits for the generated summary, making the system flexible to various needs.
- e. **Sentiment Analysis:** Sentiment analysis is handled by a custom **SentimentAnalysis** class that utilizes **NLTK's VADER (Valence Aware Dictionary and sEntiment Reasoner)** model to assess the emotional tone of the input article. The sentiment scores help users understand whether the article has a positive, neutral, or negative tone.
- f. **Keyword Extraction:** The system includes a **KeywordExtraction** class that uses **TF-IDF (Term Frequency-Inverse Document Frequency)** vectorization to identify the most relevant words in the article. This method ensures that the keywords selected are not only frequent in the article but also carry significant meaning relative to less common words, making it a powerful tool for content summarization.
- g. **Topic Modeling:** The **TopicModeling** class utilizes **Latent Dirichlet Allocation (LDA)**, a widely used topic modeling technique. By converting the input text into a document-term matrix using **CountVectorizer**, the LDA model identifies the major themes or topics within the article. Users can adjust the number of topics they want to extract, making this a flexible tool for understanding large, complex articles.
- h. **Evaluation and ROUGE Scoring:** To evaluate the performance of both extractive and abstractive summarization methods, the **NewsSummarization** class includes functionality for **ROUGE scoring**. ROUGE (Recall-Oriented Understudy for Gisting Evaluation) compares the overlap of n-grams between the generated summary and a reference summary

T5 Transformer

The T5 Text-to-Text Transfer Transformer Model was proposed by Google in [1]. This proposes a unified framework to convert all NLP problems into a text-to-text format. This means, one model will be able to translate, summarize, classify etc based on the input prefix.

The model recognises the task by the input prefix and gives the output as a “text”. The T5 model was trained on the C4 (Colossal Clean Crawled Corpus). This dataset was obtained by web scraping and removing HTML tags. The dataset was cleaned by applying various filters like removing offensive words, removing duplicates etc and T5 was trained on this cleaned dataset.



Technology Stack:

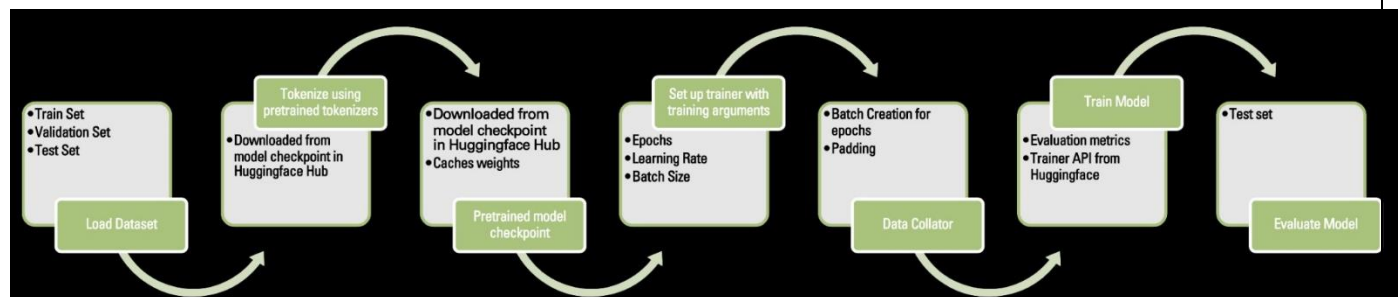
- **Streamlit:** The core framework for building the interactive user interface, enabling real-time interaction between users and the backend processing components.
- **Python:** The main programming language used throughout the backend to handle tasks such as text preprocessing, summarization, sentiment analysis, keyword extraction, and topic modeling.
- **NLTK:** The **Natural Language Toolkit** provides essential utilities for text preprocessing, including tokenization, stopword removal, lemmatization, and sentiment analysis using VADER.

- **Pandas and Numpy:** Used for data manipulation and numerical operations, especially in transforming text data into a format suitable for NLP tasks.
- **TF-IDF (TfidfVectorizer):** Employed for keyword extraction by calculating the importance of words in the context of the article.
- **CountVectorizer and LDA (Latent Dirichlet Allocation):** Used for topic modeling, where LDA identifies clusters of words that represent distinct topics in the article.
- **Heapq:** Used to select top-N sentences for extractive summarization based on sentence scores.
- **Rouge Scorer:** Utilized to compute **ROUGE** scores, a standard metric in text summarization evaluation, which ensures the generated summaries are as close as possible to human-written ones.
- **PIL (Python Imaging Library):** Responsible for displaying images in the user interface, enhancing the visual appeal of the application.

Huggingface

Huggingface provides access to thousands of pre-trained models that can be downloaded and used for multiple tasks. These models can also be finetuned to a particular dataset and provide transfer learning.

For this project, the T5 transformer from Huggingface hub was fine tuned on the CNN/Dailymail dataset. The process to finetune a model is as follows (Code Snippets for steps in Appendix):



1. **Load dataset:** Huggingface has a repository of many datasets. The CNN/Dailymail dataset is available in this. The dataset can be downloaded and it is organized into a training set, validation set and test set.

2. **Pretrained Tokenizer:** Each pretrained model comes with its pretrained tokenizer in Huggingface hub. This tokenizer is able to tokenize the data into the way the model expects it. The dataset is tokenized using the T5 tokenizer.
3. **Pretrained Model:** Huggingface provides pre-trained model checkpoints of all models. This has the model architecture and the trained weights that can be used for any NLP task. This model is downloaded and the weights can be fine-tuned for our specific news summarization task.
4. **Training Arguments:** Every model has various training hyperparameters that can be tuned. These arguments decide the training run of the model.
5. **Data Collator:** Huggingface provides a data collator for each NLP task. The data collator efficiently creates batches for each epoch and also pads the input.
6. **Train model:** The model is finetuned with the help of the Trainer API and the predefined training arguments
7. **Evaluate Model:** The model is evaluated on the test set using ROUGE metrics.

This comprehensive system incorporates multiple advanced NLP techniques and provides users with a flexible, intuitive interface to perform news article summarization, keyword extraction, topic modeling, and sentiment analysis.

6. Implementation

- Backend Development

The backend of the system is built entirely in **Python** and acts as the engine that processes all user inputs and performs natural language processing (NLP) tasks. It is designed in a modular structure, with distinct classes and functions responsible for different operations like text preprocessing, summarization, sentiment analysis, keyword extraction, and topic modeling. This separation of concerns ensures the system is scalable and maintainable, with each module dedicated to a specific task.

Key backend components include:

- **Preprocessing Module:** This handles cleaning the text input by converting it to lowercase, tokenizing it into sentences and words, removing stopwords, and lemmatizing words. This ensures that the text is in an optimal form for subsequent NLP tasks.
- **Summarization Module:** Both extractive and abstractive summarization methods are implemented. The extractive approach selects the most relevant sentences from the article, while the abstractive approach leverages a pre-trained **T5 model** for generating summaries.
- **Keyword Extraction and Topic Modeling:** Implemented using **TF-IDF** for keyword extraction and **Latent Dirichlet Allocation (LDA)** for topic modeling, these features provide additional insights into the content of the articles.
- **Sentiment Analysis Module:** This uses the **VADER sentiment analysis model** from **NLTK** to gauge the emotional tone of the article.

Key Features Implemented:

1. **Text Preprocessing:** Includes text normalization, tokenization, stopword removal, and lemmatization using **NLTK**.
2. **Extractive Summarization:** Scores sentences based on word frequencies and selects the most important sentences to generate a concise summary.
3. **Abstractive Summarization:** Uses a transformer-based model from **Hugging Face Transformers** to generate human-like summaries.
4. **Sentiment Analysis:** Employs **VADER** to determine whether the text expresses positive, negative, or neutral sentiment.

5. **Keyword Extraction:** Utilizes **TF-IDF** vectorization to identify the most important keywords from the text.
6. **Topic Modeling:** Applies **LDA** to extract key topics from the article, offering a deeper understanding of its content.
7. **ROUGE Evaluation:** ROUGE scores are computed to evaluate the quality of the generated summaries.

- **Frontend Development**

The frontend of the text summarization system is developed using **Streamlit**, providing a highly interactive and responsive web-based user interface. Streamlit was selected for its seamless integration with Python and its ability to offer real-time updates without requiring extensive knowledge of frontend frameworks like React or JavaScript. This ensures that the system remains both lightweight and user-friendly.

Key Elements of the User Interface:

- **Input Section:**
 - Users are presented with a **text area widget** on the left side of the interface, allowing them to input the news article they want to summarize. This widget has been configured to handle a significant amount of text and offers a placeholder to guide users on what to enter.
- **Sidebar Controls:**

The **right sidebar** contains user input options, allowing customization of summarization parameters:

- **Number Input Fields** let users control the number of sentences for the extractive summary and set minimum and maximum word limits for the abstractive summary.
 - A **"Summarize!" button** triggers the summarization and analysis once the user has finished entering the article and configuring the parameters.
- **Display of Results:**

After clicking the "Summarize!" button, the system dynamically generates and displays:

- **Extractive Summary:** Key sentences are extracted and displayed concisely.
 - **Abstractive Summary:** Generated using the T5 model, a coherent and shorter summary is provided.

- **Sentiment Analysis:** The emotional tone of the article is broken down into sentiment scores (positive, negative, neutral).
- **Keyword Extraction and Topic Modeling:** The top keywords and main topics of the article are extracted and presented in an easy-to-understand format.

- **Visual Enhancements:**

The interface has been enhanced visually through the use of **custom CSS** to modify the theme's colors and styles. This includes background color changes, custom styling for buttons, and image integration using the **PIL** library to display aesthetic visuals, ensuring the interface is professional and visually appealing.

Key UI Features and User Experience Considerations:

- **Responsive Design:**

The layout is designed to be visually balanced, with the input section and results side-by-side to ensure the user doesn't need to scroll excessively. The interface adjusts dynamically to fit different screen sizes.

- **Interactive Elements:**

Users experience real-time interaction with the system. Summarization, sentiment analysis, and keyword/topic extraction are executed promptly after the user submits an article, with results displayed instantly.

- **Customizable Summarization Parameters:**

The **number input fields** for extractive and abstractive summary lengths allow users to fine-tune the length and depth of the summaries, offering flexibility in how the results are presented.

- **Visual and Informative Feedback:**

Summarization results, sentiment analysis, and topic modeling are clearly laid out, making it easy for users to interpret the output of their analysis at a glance.

- **Key Features and algorithms**

- Detailed Explanation of Key Algorithms and Models Implemented:

1. **Text Preprocessing:**

- a. Convert text to lowercase

```
pass
def toLower(self, x):
    '''Converts string to lowercase'''
    return x.lower()
```

- b. Remove punctuation

```
def preprocess_sentences(self, all_sentences):
    '''Tokenizes sentences into words, removes punctuations, stopwords and
    performs tokenization'''
    word_tokenizer = nltk.RegexpTokenizer(r'\w+')
    sentences = []
    special_characters = re.compile("[^A-Za-z0-9 ]")
    for s in all_sentences:
        # remove punctuation
        s = re.sub(special_characters, " ", s)
```

- c. Remove stopwords (common words like "the", "is", which do not contribute to meaning)

```
def removeStopwords(self, sentence):
    '''Removes stopwords from a sentence'''
    stop_words = stopwords.words('english')
    tokens = [token for token in sentence if token not in stop_words]
    return tokens
```

2. **Tokenization:**

- a. Tokenize the text into **sentences** and **words**

```
def preprocess_sentences(self, all_sentences):
    '''Tokenizes sentences into words, removes punctuations, stopwords and
    performs tokenization'''
    word_tokenizer = nltk.RegexpTokenizer(r'\w+')
    sentences = []
    special_characters = re.compile("[^A-Za-z0-9 ]")
    for s in all_sentences:
        # remove punctuation
        s = re.sub(special_characters, " ", s)
        # Word tokenize
        words = word_tokenizer.tokenize(s)
        # Remove Stopwords
        words = self.removeStopwords(words)
        # Perform lemmatization
        words = self.wordnet_lemmatize(words)
        sentences.append(words)
    return sentences
```

3. Word Frequency Calculation:

- a. Calculate word frequency for all words, excluding stopwords
- b. Normalize frequencies using the maximum frequency, to adjust for varying article lengths

4. Extractive Summarization Algorithm:

- a. Based on word frequencies and sentence scoring. The algorithm identifies sentences that contain high-frequency words, under the assumption that these sentences are the most important.
- b. The **NewsSummarization** class performs this task by tokenizing the text, calculating word frequencies, and then scoring sentences based on how many important words they contain. The highest-scoring sentences are then selected to form the summary.
- c. Tools used: **NLTK** for text tokenization and processing, **heapq** for selecting top sentences.

5. Abstractive Summarization Model:

- a. A pre-trained **T5 model** from the **Hugging Face Transformers library** is used for this task. This model generates summaries by understanding the underlying meaning of the text and generating new sentences rather than merely selecting important ones.
- b. The user can control the maximum and minimum word limits of the summary, allowing flexibility in summarization depth.
- c. Tools used: **Hugging Face Transformers**, specifically the model **shivaniNK8/t5-small-finetuned-cnn-news**.

6. Sentiment Analysis Model:

- a. Sentiment analysis is performed using **VADER**, an open-source sentiment analysis tool in **NLTK**. VADER is particularly effective for analyzing social media text and short news articles, providing a sentiment polarity score for the input text.
- b. The output includes the percentage of positive, negative, and neutral sentiment in the article, offering users insight into the emotional tone.
- c. Tools used: **VADER from NLTK**.

```
def sentiment_analysis(self, text):  
    """Performs sentiment analysis on the article"""  
    sentiment_result = self.sentiment_analyzer(text)[0]  
    return sentiment_result['label'], sentiment_result['score']
```

7. Keyword Extraction Algorithm:

- The system uses **TF-IDF** to extract the most important keywords from the article. The **TfidfVectorizer** class from **scikit-learn** is employed, which assigns higher weights to words that are frequent in the document but infrequent across a broader corpus.
- The extracted keywords help users quickly identify the main topics and themes of the article.
- Tools used: **TF-IDF Vectorizer from scikit-learn**.

```
def extract_keywords(self, text, num_keywords=5):
    '''Extracts top N keywords from the text using TF-IDF'''
    vectorizer = TfidfVectorizer(max_df=0.85, stop_words='english')
    tfidf_matrix = vectorizer.fit_transform([text])
    feature_names = vectorizer.get_feature_names_out()
    tfidf_scores = tfidf_matrix.toarray()[0]

    keyword_indices = tfidf_scores.argsort()[::-num_keywords][::-1]
    keywords = [feature_names[i] for i in keyword_indices]
    return keywords
```

8. Topic Modeling Algorithm:

- The **Latent Dirichlet Allocation (LDA)** algorithm is used for topic modeling. This probabilistic model clusters words into topics by analyzing word co-occurrence patterns in the text. Users can configure the number of topics they wish to extract.
- The **CountVectorizer** from **scikit-learn** is used to convert the text into a document-term matrix, which serves as the input to the LDA model.
- Tools used: **LDA and CountVectorizer from scikit-learn**.

```
def topic_modeling(self, text, num_topics=5, num_words=5):
    '''Applies LDA for topic modeling to discover main topics in the text'''
    vectorizer = TfidfVectorizer(max_df=0.85, stop_words='english')
    X = vectorizer.fit_transform([text])

    self.lda_model.fit(X)
    feature_names = vectorizer.get_feature_names_out()

    topics = {}
```

9. Evaluation (ROUGE Scoring):

- To evaluate the performance of the summarization models, the system uses the **ROUGE metric**, a standard evaluation method for summarization tasks. It compares the overlap of n-grams between the generated summary and a reference summary.
- The **rouge_scorer** from the **rouge-score** library is used to compute the ROUGE-1 and ROUGE-L scores for both extractive and abstractive summaries.

10. Fine-tuning of T5Model:

- a. The abstractive summarization model is fine-tuned on the **CNN/DailyMail** dataset using the **T5 Transformer model**. The fine-tuning process allows the model to generate more coherent and contextually relevant summaries for news articles.
- b. The model's **pretrained tokenizer** from Huggingface is used to tokenize the input data. During the fine-tuning process, hyperparameters such as learning rate, batch size, and number of epochs are adjusted to improve model performance.
- c. After fine-tuning, the model is evaluated using ROUGE metrics to assess improvements in the quality of the generated summaries.

This comprehensive architecture ensures that the system not only performs complex NLP tasks efficiently but also provides a smooth and intuitive user experience through the frontend. The system's modular design makes it easy to extend with additional features or models in the future.

7. Evaluation and Testing

- Performance Evaluation

Evaluation Metric

Since the target in summarization is a text summary, it becomes complex to measure model performance. The widely used metric for text summarization is ROUGE (Recall Oriented Understudy for Gisting Evaluation). It generates a precision, recall and f1 score of the overlap between actual summary and generated summary.

ROUGE makes use of n-grams for calculating the overlap. As such, this gives rise to different ROUGE metrics like ROUGE-1, ROUGE-2, which consider 1-gram (basically word overlap) and bigrams. Another ROUGE metric is the ROUGE-L, which considers the Longest Common Subsequence (LCS) of matching words. (Code Snippets in Appendix)

Following images explain the ROUGE metric with an example of “I really loved reading th Hunger Games” as the generated summary and “I loved reading the Hunger Games” as a reference summary.

This has a ROUGE-1 recall of 100%, ROUGE-1 precision 85%, ROUGE-1 F1 Score 91% ROUGE-L recall 100% and ROUGE-L precision 85% as explained below.

I really loved reading the Hunger Games

Machine generated summary

$$\text{ROUGE-1 recall} = \frac{\text{Num word matches}}{\text{Num words in reference}} = \frac{6}{6}$$

I loved reading the Hunger Games

$$\text{ROUGE-1 precision} = \frac{\text{Num word matches}}{\text{Num words in summary}} = \frac{6}{7}$$

Human reference summary

$$\text{ROUGE-1 F1-score} = 2 \left(\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \right)$$

I really loved reading the Hunger Games

Machine generated summary

$$\text{ROUGE-L recall} = \frac{\text{LCS}(\text{gen, ref})}{\text{Num words in reference}} = \frac{6}{6}$$

I loved reading the Hunger Games

$$\text{ROUGE-L precision} = \frac{\text{LCS}(\text{gen, ref})}{\text{Num words in summary}} = \frac{6}{7}$$

Human reference summary

This project evaluates extractive and abstractive summarization on the basis of different ROUGE scores achieved.

Following table states the SOTA T5 metrics on the CNN/Dailymail Dataset

Metric	Score
ROUGE-1	43.52
ROUGE-2	21.55
ROUGE-L	40.69

Rouge Performance on Test Set

Metric	Score
ROUGE-1	38.74
ROUGE-2	17.24
ROUGE-L	26.73
ROUGE-Lsum	32.71

Evaluate Abstractive Summarization on Test Set

```
[ ]: abstractive_score = news_summarizer.evaluate_abstractive(test_df, rouge_score, summarizer)
      abstractive_score
```

```
[30]: {'rouge1': 38.74, 'rouge2': 17.24, 'rougeL': 26.73, 'rougeLsum': 32.71}
```

Extractive score for comparison

```
[ ]: extractive_score = news_summarizer.evaluate_extractive(test_df, rouge_score)
      extractive_score
```

```
[39]: {'rouge1': 28.59, 'rouge2': 11.93, 'rougeL': 17.54, 'rougeLsum': 23.76}
```

Metric	Extractive Summarizer	Abstractive Summarizer	SOTA T5 Summarizer
ROUGE-1	28.91	38.74	43.52
ROUGE-2	12.04	17.24	21.55
ROUGE-L	17.76	26.73	40.69
ROUGE-Lsum	24.14	32.71	

Experimental Results and Model Comparison

Example article about the James Webb Telescope from NASA that was provided to the summarizer is given below, along with the generated extractive and abstractive summaries.

Enter the article you want to summarize

Webb Space Telescope, offering what NASA said was “the deepest, sharpest infrared view of the universe” ever taken.

During a ceremony at the White House on Monday evening, Biden said it was “an historic day” as the world’s largest and most powerful space science telescope offered a “new window into the history of our universe”.

“Today we’re going to get a glimpse at the first light to shine through that window,” Biden said shortly before the release of the image, which showed bright white, yellow and orange lights that NASA said represented “galaxies once invisible to us”.

“Light from other worlds, orbiting stars far beyond our own,” Biden said. “The oldest-documented light in the history of the universe from over 13 billion – let me say that again – 13 billion years ago.”

The image will be followed on Tuesday by the release of four more galactic beauty shots from the

Comparing Extractive and Abstractive Technique and Conclusion

- Extractive summarizer has a ROUGE-1 score of 28.59 whereas Abstractive summarizer has a ROUGE score of 38.74. The State-of-the-art ROUGE score is 42.3
- Abstractive summaries capture more context than extractive summaries
- Extractive Summarizer is way faster than Abstractive. Generating 1 abstractive summary requires 4-5 seconds
- Abstractive summarizer has better performance for all n-grams (rouge-1, rouge-2 etc)
- Depending on the use case, either of the models can be utilised to generate a good summary of any news article

8. Challenges faced and solutions

During the development of the text summarization and analysis system, several technical challenges were encountered:

- **Model Selection for Abstractive Summarization:**

- One of the main challenges was selecting an appropriate pre-trained model for abstractive summarization. While larger models like BART or Pegasus were initially considered, resource limitations made it impractical to use them. This was resolved by using the **T5-small model** fine-tuned on the CNN/Daily Mail dataset, which offered a good balance between performance and resource efficiency.

- **Handling Long Text Inputs:**

News articles can often be lengthy, leading to issues with summarization models that have a token limit. For example, T5 has a token limit of around 512 tokens. To handle this, articles were truncated or summarized incrementally in batches, allowing the system to process longer texts without overwhelming the model.

- **Streamlit Layout Customization:**

Customizing the UI layout and injecting custom CSS into Streamlit was a challenge, as Streamlit is designed for simplicity and does not natively support complex theming or advanced layouts. This was solved by using **custom CSS injection** to change background colors, button styles, and layout configurations, giving the application a professional look while maintaining its responsiveness.

- **Real-time Performance Considerations:**

Since the system performs multiple NLP tasks (summarization, keyword extraction, sentiment analysis, and topic modeling) on the input text, there was a risk of the system becoming slow, especially with longer articles. Optimizations were made by preloading models and caching them to minimize loading time, and computationally intensive tasks were streamlined to improve performance.

- **Keyword and Topic Extraction Accuracy:**

Ensuring the accuracy and relevance of extracted keywords and topics was another challenge. Initial attempts with simple TF-IDF and LDA models provided some meaningful results but were not always reliable. The solution involved fine-tuning hyperparameters of the **TF-IDF vectorizer** and **Latent Dirichlet Allocation (LDA)** models, such as adjusting the minimum document frequency and the number of topics to ensure better performance.

9. Conclusion and Future work

This project successfully developed an end-to-end **news summarization and analysis system** capable of performing extractive and abstractive summarization, sentiment analysis, keyword extraction, and topic modeling. Using **Streamlit** for frontend development and **Python NLP libraries** for backend processing, the system offers a simple, intuitive user experience while providing powerful natural language processing capabilities. The integration of various NLP models, including the **T5-small model for abstractive summarization** and **VADER for sentiment analysis**, allowed for a rich feature set that can generate summaries and insights from large text inputs.

Limitations:

Despite the system's achievements, there are several limitations:

- **Handling Long Texts:** The token limit of the T5 model restricts the processing of very lengthy articles. Truncation is used as a workaround, but this may result in the loss of critical information.
- **Extractive Summarization Simplicity:** The extractive summarization technique, based purely on word frequencies, may miss the nuance of the text and is unable to capture contextual information as effectively as abstractive models.
- **Abstractive Summarization Quality:** While the T5-small model provides reasonable performance, it may not generate summaries as coherent or insightful as larger models like **GPT-3** or **Pegasus** due to its smaller size and limited capacity.

Future Work:

- **Enhancing Abstractive Summarization:** Future work could explore the use of larger, more sophisticated models (e.g., **BART**, **Pegasus**, **GPT models**) for abstractive summarization to improve the quality and coherence of generated summaries.
- **Improving Summarization Algorithms:** Incorporating **hybrid approaches** that combine both extractive and abstractive methods could offer more balanced summaries by leveraging the strengths of both techniques.
- **Better Handling of Long Texts:** Implementing advanced text chunking and long-document handling strategies, such as using **hierarchical transformers** or **Longformer**, would enable the system to handle longer inputs more efficiently.

- **User Experience Enhancements:** Adding features like **real-time feedback** during text processing and providing visualizations for topic modeling results (e.g., word clouds) could enhance the overall user experience.
- **Deployment and Scalability:** Expanding the system to handle real-world deployments with larger user bases and concurrent usage would require further optimizations and possibly migrating to **cloud-based services** like **AWS** or **Google Cloud** to ensure scalability.

This project lays the foundation for a versatile and expandable text summarization and analysis platform, with ample scope for future improvements.

10. Appendix

Preprocessing class for Extractive summarization:

Provides functions for lowercasing, stopwords removal, punctuation removal and other preprocessing tasks.

```
class Preprocess():
    def __init__(self):
        pass
    def toLower(self, x):
        '''Converts string to lowercase'''
        return x.lower()

    def sentenceTokenize(self, x):
        '''Tokenizes document into sentences'''
        sent_tokenizer = nltk.data.load("tokenizers/punkt/english.pickle")
        sentences = sent_tokenizer.tokenize(x)
        return sentences

    def preprocess_sentences(self, all_sentences):
        '''Tokenizes sentences into words, removes punctuations, stopwords and
        performs tokenization'''
        word_tokenizer = nltk.RegexpTokenizer(r"\w+")
        sentences = []
        special_characters = re.compile("[^A-Za-z0-9 ]")
        for s in all_sentences:
            # remove punctuation
            s = re.sub(special_characters, " ", s)
            # Word tokenize
            words = word_tokenizer.tokenize(s)
            # Remove Stopwords
            words = self.removeStopwords(words)
            # Perform lemmatization
            words = self.wordnet_lemmatize(words)
            sentences.append(words)
        return sentences

    def removeStopwords(self, sentence):
        '''Removes stopwords from a sentence'''
        stop_words = stopwords.words('english')
        tokens = [token for token in sentence if token not in stop_words]
        return tokens

    def wordnet_lemmatize(self, sentence):
        '''Lemmatizes tokens in a sentence'''
        lemmatizer = WordNetLemmatizer()
        tokens = [lemmatizer.lemmatize(token, pos='v') for token in sentence]
        return tokens
```



```

def complete_preprocess(self, text):
    '''Performs complete preprocessing on document'''
    #Convert text to lowercase
    text_lower = self.toLower(text)
    #Sentence tokenize the document
    sentences = self.sentenceTokenize(text_lower)
    #Preprocess all sentences
    preprocessed_sentences = self.preprocess_sentences(sentences)
    return preprocessed_sentences

def generate_wordcloud(self, text):
    word_cloud = WordCloud(collocations = False, background_color = 'white').generate(text)
    plt.figure(figsize=(15,8))
    plt.imshow(word_cloud, interpolation='bilinear')
    plt.axis("off")
    plt.show()

def calculate_length(self, df):
    df["article_len"] = df["article"].apply(lambda x: len(x.split()))
    df["highlights_len"] = df["highlights"].apply(lambda x: len(x.split()))
    return df

```

News Summarizer class: Contains summarization code, ROUGE evaluation.

```
class NewsSummarization():
    def __init__(self):
        # Sentiment analysis model
        self.sentiment_analyzer = pipeline("sentiment-analysis")
        # Number of topics for topic modeling
        self.num_topics = 5
        self.lda_model = LatentDirichletAllocation(n_components=self.num_topics, random_state=0)

    def extractive_summary(self, text, sentence_len=8, num_sentences=3):
        '''Generates extractive summary of num_sentences length using sentence scoring'''
        word_frequencies = {}
        preprocessor = Preprocess() # Assuming you have a custom preprocessor class
        tokenized_article = preprocessor.complete_preprocess(text)

        for sentence in tokenized_article:
            for word in sentence:
                if word not in word_frequencies.keys():
                    word_frequencies[word] = 1
                else:
                    word_frequencies[word] += 1

        maximum_frequency = max(word_frequencies.values())
        for word in word_frequencies.keys():
            word_frequencies[word] = (word_frequencies[word] / maximum_frequency)

        sentence_scores = {}
        sentence_list = nltk.sent_tokenize(text)

        for sent in sentence_list:
            for word in nltk.word_tokenize(sent.lower()):
                if word in word_frequencies.keys():
                    if len(sent.split(' ')) > sentence_len:
                        if sent not in sentence_scores.keys():
                            sentence_scores[sent] = word_frequencies[word]
                        else:
                            sentence_scores[sent] += word_frequencies[word]

        summary_sentences = heapq.nlargest(num_sentences, sentence_scores, key=sentence_scores.get)
        summary = ' '.join(summary_sentences)
        return summary
```

```

def get_rouge_score(self, actual_summary, generated_summary):
    '''Compute ROUGE score for actual and generated summary'''
    scorer = rouge_scorer.RougeScorer(['rouge1', 'rougeL'], use_stemmer=True)
    scores = scorer.score(actual_summary, generated_summary)
    return scores

def evaluate_extractive(self, dataset, metric):
    '''Evaluate the extractive summarizer using ROUGE metrics'''
    summaries = [self.extractive_summary(text) for text in dataset["article"]]
    score = metric.compute(predictions=summaries, references=dataset["highlights"])
    rouge_names = ["rouge1", "rouge2", "rougeL", "rougeLsum"]
    rouge_dict = dict((rn, round(score[rn].mid.fmeasure * 100, 2)) for rn in rouge_names)
    return rouge_dict

def evaluate_abstractive(self, dataset, metric, summarizer):
    '''Evaluate the abstractive summarizer using ROUGE metrics'''
    summaries = [summarizer(text, max_length=120, min_length=80, do_sample=False)[0]['summary_text'] for text in dataset["article"]]
    score = metric.compute(predictions=summaries, references=dataset["highlights"])
    rouge_names = ["rouge1", "rouge2", "rougeL", "rougeLsum"]
    rouge_dict = dict((rn, round(score[rn].mid.fmeasure * 100, 2)) for rn in rouge_names)
    return rouge_dict

def extract_keywords(self, text, num_keywords=5):
    '''Extracts top N keywords from the text using TF-IDF'''
    vectorizer = TfidfVectorizer(max_df=0.85, stop_words='english')
    tfidf_matrix = vectorizer.fit_transform([text])
    feature_names = vectorizer.get_feature_names_out()
    tfidf_scores = tfidf_matrix.toarray()[0]

    keyword_indices = tfidf_scores.argsort()[-num_keywords:][::-1]
    keywords = [feature_names[i] for i in keyword_indices]
    return keywords

def sentiment_analysis(self, text):
    '''Performs sentiment analysis on the article'''
    sentiment_result = self.sentiment_analyzer(text)[0]
    return sentiment_result['label'], sentiment_result['score']

def topic_modeling(self, text, num_topics=5, num_words=5):
    '''Applies LDA for topic modeling to discover main topics in the text'''
    vectorizer = TfidfVectorizer(max_df=0.85, stop_words='english')
    X = vectorizer.fit_transform([text])

    self.lda_model.fit(X)
    feature_names = vectorizer.get_feature_names_out()

    topics = {}

def summarize_article(self, text, num_sentences=3):
    '''Combines extractive summarization, keyword extraction, sentiment analysis, and topic modeling'''
    summary = self.extractive_summary(text, num_sentences=num_sentences)
    keywords = self.extract_keywords(text)
    sentiment_label, sentiment_score = self.sentiment_analysis(text)
    topics = self.topic_modeling(text)

    return {
        'summary': summary,
        'keywords': keywords,
        'sentiment': {
            'label': sentiment_label,
            'score': sentiment_score
        },
        'topics': topics
    }

score = metric.compute(predictions=summaries, references=dataset["highlights"])
rouge_names = ["rouge1", "rouge2", "rougeL", "rougeLsum"]
rouge_dict = dict((rn, round(score[rn].mid.fmeasure * 100, 2)) for rn in rouge_names)
return rouge_dict

def evaluate_abstractive(self, dataset, metric, summarizer):
    summaries = [summarizer(text, max_length = 120, min_length = 80, do_sample = False)[0]['summary_text'] for text in dataset["article"]]
    score = metric.compute(predictions=summaries, references=dataset["highlights"])
    rouge_names = ["rouge1", "rouge2", "rougeL", "rougeLsum"]
    rouge_dict = dict((rn, round(score[rn].mid.fmeasure * 100, 2)) for rn in rouge_names)
    return rouge_dict

news_summarizer = NewsSummarization()

```

Evaluation code for extractive and abstractive summarization:

Computes ROUGE metrics for test set

```
def compute_metrics(eval_pred):
    '''Computes metrics that can be used while training'''
    predictions, labels = eval_pred
    decoded_preds = tokenizer.batch_decode(predictions, skip_special_tokens=True)
    # Replace -100 in the labels as we can't decode them
    labels = np.where(labels != -100, labels, tokenizer.pad_token_id)
    # Decode reference summaries into text
    decoded_labels = tokenizer.batch_decode(labels, skip_special_tokens=True)
    # newline after each sentence
    decoded_preds = ["\n".join(sent_tokenize(pred.strip())) for pred in decoded_preds]
    decoded_labels = ["\n".join(sent_tokenize(label.strip())) for label in decoded_labels]
    # Compute ROUGE scores
    result = rouge_score.compute(
        predictions=decoded_preds, references=decoded_labels, use_stemmer=True
    )
    # median scores, we get the fmeasure as percentage
    result = {key: value.mid.fmeasure * 100 for key, value in result.items()}
    return {k: round(v, 4) for k, v in result.items()}
```

```
def preprocess_function(examples):
    '''Preprocesses the data as required by huggingface transformer'''
    model_inputs = tokenizer(
        examples["article"], max_length=max_input_length, truncation=True
    )
    with tokenizer.as_target_tokenizer():
        labels = tokenizer(
            examples["highlights"], max_length=max_target_length, truncation=True
        )

    model_inputs["labels"] = labels["input_ids"]
    return model_inputs
```

```
def evaluate_extractive(self, dataset, metric):
    summaries = [self.extractive_summary(text) for text in dataset["article"]]
    score = metric.compute(predictions=summaries, references=dataset["highlights"])
    rouge_names = ["rouge1", "rouge2", "rougeL", "rougeLsum"]
    rouge_dict = dict((rn, round(score[rn].mid.fmeasure * 100, 2)) for rn in rouge_names)
    return rouge_dict

def evaluate_abstractive(self, dataset, metric, summarizer):
    summaries = [summarizer(text, max_length = 120, min_length = 80, do_sample = False)[0]['summary_text'] for text in dataset["article"]]
    score = metric.compute(predictions=summaries, references=dataset["highlights"])
    rouge_names = ["rouge1", "rouge2", "rougeL", "rougeLsum"]
    rouge_dict = dict((rn, round(score[rn].mid.fmeasure * 100, 2)) for rn in rouge_names)
    return rouge_dict
```

Preprocess function for T5 transformer:

Function for preprocessing dataset and tokenizing it for T5 transformer **Training**

Arguments for T5 transformer:

Setting up Training Arguments

Let us set up the training parameters.

- Used batch size 8 to fit model in memory and avoid OOM issues
- Epochs: Recommended to train for 5-8 epochs to prevent overfitting.
- Learning rate: Small learning rate, as we want to finetune and not train from scratch. Used 5.6×10^{-4} after experimenting with multiple values
- Evaluation strategy: 'epoch', evaluates model after an epoch
- weight_decay: Used 0.01
- push_to_hub: Save model training files to huggingface hub, helpful as Google Colab can disconnect anytime
- predict_with_generate: True, to evaluate model after every epoch

```
[ ]
batch_size = 8
num_train_epochs = 5
# Show the training loss with every epoch
logging_steps = len(tokenized_datasets["train"]) // batch_size
model_name = model_checkpoint.split("/")[-1]

args = Seq2SeqTrainingArguments(
    output_dir=f"{model_name}-finetuned-cnn-news",
    evaluation_strategy="epoch",
    learning_rate=5.6e-4,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    weight_decay=0.01,
    save_total_limit=3,
    num_train_epochs=num_train_epochs,
    predict_with_generate=True,
    logging_steps=logging_steps,
    push_to_hub=True,
)
```


11. UI Page

Deploy

HIGHLIGHTS!

A News Summarizer

Provide a news article and get a summary within seconds!



Enter the article you want to summarize

Enter Article Body Here

Summary Options

Set parameters for summarization:

Summary Length (Extractive)

3 - +

Max words (Abstractive)

200 - +

Min words (Abstractive)

100 - +

Summarize!

More About Summarization

Enter the article you want to summarize

The choices made today by political leaders, civil society, and communities will shape the future of Sri Lanka's children for years to come.

The words of Article 3(1) of the United Nations Convention on the Rights of the Child (CRC) resonate profoundly at this critical time: "In all actions concerning children, the best interest of the child shall be a primary consideration." Sri Lanka, one of the first countries to ratify the CRC in 1991, has made notable strides in advancing children's rights. From achieving Universal Child Immunisation in 1989 to establishing the National Child Protection Authority in 1998, Sri Lanka has a proud history of prioritising children in public policy.

However, much remains to be done. This is a moment of choice—an opportunity to firmly place children at the centre of governance, ensuring that their protection, development, and future are prioritised in all national

Extractive Summary

The words of Article 3(1) of the United Nations Convention on the Rights of the Child (CRC) resonate profoundly at this critical time: "In all actions concerning children, the best interest of the child shall be a primary consideration." Sri Lanka, one of the first countries to ratify the CRC in 1991, has made notable strides in advancing children's rights. From achieving Universal Child Immunisation in 1989 to establishing the National Child Protection Authority in 1998, Sri Lanka has a proud history of prioritising children in public policy. The choices made today by political leaders, civil society, and communities will shape the future of Sri Lanka's children for years to come.

Abstractive Summary

Article 3(1) of the United Nations Convention on the Rights of the Child (CRC) resonates profoundly at this critical time. Sri Lanka, one of the first countries to ratify the CRC in 1991, has made notable strides in advancing children's rights. This is a moment of choice—an opportunity to firmly place children at the centre of governance, ensuring their protection, development, and future are prioritised in all national policies.

Sentiment Analysis

```
{  
  "neg" : 0.017  
  "neu" : 0.86  
  "pos" : 0.123  
  "compound" : 0.94  
}
```

Keyword Extraction

value

children

child

lanka

sri

rights

future

national

crc

protection

authority

Topic Modeling

```
{  
  "Topic 1" : [  
    0 : "development"  
    1 : "ensuring"  
    2 : "establishing"  
    3 : "firmly"  
    4 : "governance"  
    5 : "history"  
    6 : "immunisation"  
    7 : "words"  
    8 : "moment"  
    9 : "years"  
  ]  
  "Topic 2" : [  
    0 : "development"  
    1 : "ensuring"  
    2 : "establishing"  
    3 : "firmly"  
    4 : "governance"  
    5 : "history"
```

```
    4 : "governance"  
    5 : "history"  
    6 : "immunisation"  
    7 : "words"  
    8 : "moment"  
    9 : "years"  
  ]  
  "Topic 3" : [  
    0 : "authority"  
    1 : "protection"  
    2 : "crc"  
    3 : "national"  
    4 : "future"  
    5 : "rights"  
    6 : "sri"  
    7 : "lanka"  
    8 : "child"  
    9 : "children"  
  ]  
}
```


12. References

1. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2020, July 28). *Exploring the limits of transfer learning with a unified text-to-text transformer*. arXiv.org. Retrieved August 4, 2022, from <https://arxiv.org/abs/1910.10683>
2. Nallapati, R., Zhou, B., Santos, C. N. dos, Gulcehre, C., & Xiang, B. *Abstractive text summarization using sequence-to-sequence RNNs and beyond*. arXiv.org. Retrieved August 4, 2022, from <https://arxiv.org/abs/1602.06023v5>
3. Lin, C.-Y. (n.d.). *Rouge: A package for automatic evaluation of summaries*. ACL Anthology. Retrieved August 4, 2022, from <https://aclanthology.org/W04-1013/>
4. community, T. H. F. D. (n.d.). *Cnn_dailymail · datasets at hugging face*. cnn_dailymail · Datasets at Hugging Face. Retrieved August 4, 2022, from https://huggingface.co/datasets/cnn_dailymail
5. *Streamlit docs*. Streamlit documentation. (n.d.). Retrieved August 4, 2022, from <https://docs.streamlit.io/>