Dissertation Submitted for the partial fulfillment of the **M.Sc. as a part of M.Sc. (Integrated) Five Years Program AIML/Data Science** degree to the Department of AIML & Data Science.

Project Dissertation

Extractive and Abstractive Text Summarization:

A Deep Learning & NLP Approach

Submitted to

By

Agarwal Shruti Hemant (DS-01)

Sharma Radhika Shivkumar (DS-15)

Semester-VIII

Under the guidance of

Rashmi Pandey[1], Dr. Ravi Gor[2]

M.Sc. (Integrated) Five Years Program AIML/Data Science

Department of AIML & Data Science.

School of Emerging Science and Technology

Gujarat University

April 2023

DECLARATION

This is to certify that the research work reported in this dissertation entitled "**Extractive and Abstractive Text Summarization: A Deep Learning & NLP Approach**" for the partial fulfilment of M.Sc. as a part of M.Sc. (Integrated) Data Science degree is the result of investigation done by myself.

Place: Ahmedabad

Agarwal Shruti Hemant

Date: 24th April 2023

DECLARATION

This is to certify that the research work reported in this dissertation entitled "**Extractive and Abstractive Text Summarization: A Deep Learning & NLP Approach**" for the partial fulfilment of M.Sc. as a part of M.Sc. (Integrated) Data Science degree is the result of investigation done by myself.

Place: Ahmedabad

Sharma Radhika Shivkumar

Date: 24th April 2023

ACKNOWLEDGMENT

We are writing this acknowledgment to express our sincere gratitude to everyone who has supported us throughout our project.

Firstly, we would like to extend our heartfelt thanks to Dr. Ravi Gor, Co-Ordinator of Department of AIML and Data Science, School of Emerging Science and Technology at Gujarat University for providing us with the necessary resources and guidance required for completing this project.

We would also like to express my gratitude to my mentor Mrs. Rashmi Pandey who helped me to navigate through the challenges faced during the project. Your expertise and knowledge in the subject matter were instrumental in shaping our ideas and concepts.

Our sincere thanks also go to the admin staff for their tireless efforts in providing administrative support and facilitating the smooth functioning of the project.

Last but not least, we would like to thank our friends and family for their unwavering support and encouragement throughout this journey. Your constant motivation and belief have been the driving force for us. We could only have done this project and made it a pleasurable experience because of them.

Thank you once again, everyone.

~ Agarwal Shruti Hemant

~ Sharma Radhika Shivkumar

Table of Contents

Chapter 1
Abstract & Keywords

ABSTRACT

In today's era of digitalization and vast amounts of online data, it has become increasingly important to help people efficiently extract relevant information from large volumes of text. Short summaries that capture the essential points are preferred over reading lengthy reports. However, manually summarizing lengthy articles can be a time-consuming and exhausting task.

To address this, various attempts have been made to automate the summarization process, and text summarization is one such technique.

There are two main methods of generating summaries - extractive and abstractive summarization.

The aim of this project is to use Machine Learning, Deep Learning and NLP to automate the summarization process while focusing on the sections that convey useful information and preserve the overall meaning.

The project involves the use of various NLP techniques such as tokenization, part-of-speech tagging, named entity recognition, and sentence parsing to extract relevant information from the text. The extracted information is then ranked based on its importance and relevance to the document as a whole. Finally, the system generates a summary by selecting the most important sentences or phrases from the document.

The application of this project is useful in scenarios where large amounts of information need to be quickly and efficiently processed, such as in news articles, scientific papers, and business reports. The text summarization system can help users quickly extract the key information from a large document, saving time and improving productivity.

KEYWORDS: Summarization, Deep Learning, Natural Language Processing, Extractive Text Summarization, Abstractive Text Summarization

Chapter 2
Introduction

Background & History

Before going to the Text summarization, first let's know what a summary is. A summary is a text that is produced from one or more texts, that conveys important information in the original text and is of a shorter form.

Text summarization has been an active research area in natural language processing (NLP) for several decades. The early approaches to text summarization were mainly rule-based and relied on hand-crafted heuristics and linguistic knowledge.

In the 1990s, researchers began to explore machine learning techniques for text summarization. One of the earliest approaches was the naive Bayes algorithm, which was used to classify sentences as important or not.

In the early 2000s, graph-based methods such as TextRank and LexRank were proposed for extractive text summarization. These methods represented the text as a graph, where nodes correspond to sentences and edges correspond to the similarity between sentences.

In the mid-2000s, researchers began to explore abstractive text summarization, which involves generating new sentences that capture the essential information in the original text. This led to the development of various neural network-based models such as recurrent neural networks (RNNs), convolutional neural networks (CNNs), and sequence-to-sequence models.

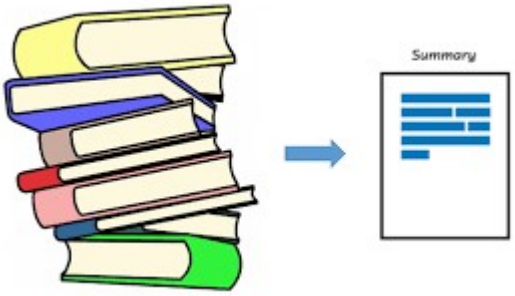
In recent years, deep learning and NLP techniques such as transformers and pre-trained language models such as BERT and GPT have been widely adopted for text summarization. These models have achieved **state-of-the-art performance** on various benchmark datasets.

Overall, the history of text summarization has witnessed a shift from rule-based and hand-crafted heuristics to machine learning and deep learning-based approaches. The field continues to evolve, and there is ongoing research to improve the quality and efficiency of text summarization systems.

Problem Statement

With the rise of digitalization, there is an explosion of information and text available on the internet. This abundance of information is a valuable source of knowledge, but it can also be overwhelming for people to process and comprehend. As a result, there is a growing need to extract the most important and relevant information from this large volume of text and present it in a concise and understandable format. Manually generating summaries of lengthy articles is a tedious and time-consuming task, and it is not feasible to process and summarize all the available text by hand. This has led to the development of automated text summarization techniques, which use natural language processing and machine learning algorithms to extract the most relevant information and generate summaries automatically. However, while extractive and abstractive summarization techniques have made significant progress, there are still challenges to be addressed, such as ensuring that the summaries are accurate, fluent, and capture the essence of the original text. Therefore, there is a continued need for research and development in this area to improve the effectiveness and efficiency of text summarization and to enable people to access and utilize the vast amount of information available to them.

Introduction



Text summarization is the process of creating a shortened version of a text document while preserving its most important information.

It is a useful technique for processing large volumes of text and making it more accessible and comprehensible for readers.

It is the process of condensing a long document or text into a shorter version while retaining its most important information. The objective of text summarization is to reduce the amount of time and effort required to read and understand a lengthy document.

It can be done manually by human editors or automatically by computer algorithms. Automatic text summarization is usually done using natural language processing (NLP) and machine learning (ML) algorithms that can analyze the input text and identify its most important information.

Some Use cases of Text Summarization:

- **Google News**: Google News uses text summarization to generate short summaries of news articles from various sources. When a user searches for a topic or clicks on a news article in the Google News app, the app generates a brief summary of the article at the top of the page. The summary includes the most important information from the article, such as the headline, author, and key points. Google News uses natural language processing (NLP) and machine learning (ML) algorithms to generate these summaries. The app analyzes the text of the article to identify the most important sentences and phrases and then generates a summary based on those key points. The algorithm used by Google News for text summarization is based on a neural network architecture called Sequence-to-Sequence (Seq2Seq). This architecture is commonly used in natural language processing tasks such as machine translation and text summarization. The Seq2Seq model uses an encoder-decoder framework to generate summaries. The encoder takes in the input text and converts it into a fixed-length vector representation, while the decoder uses this vector to generate the summary. Overall, Google News uses text summarization to provide users with a quick and easy way to get an overview of news articles, allowing them to quickly decide which articles they want to read in full.

- **Inshorts:** Inshorts is a news app that uses text summarization to provide users with short summaries of news articles. The app's tagline, "news in 60 words," reflects its focus on providing brief, to-the-point summaries of news stories. Inshorts uses a combination of human editors and artificial intelligence (AI) to generate these summaries. The app's editors curate news stories from various sources and condense them into short summaries that can be read quickly. The app also uses natural language processing (NLP) and machine learning (ML) algorithms to assist the editors in generating summaries. When an article is selected, Inshorts algorithm analyzes the text to identify the most relevant and important information in the article. The algorithm then condenses the article into a brief summary of 60 words or less. In addition to text summarization, Inshorts also uses image recognition to provide users with relevant visuals that accompany each news story. The app's AI analyzes the article and selects appropriate images from its database to display alongside the summary. Overall, Inshorts uses text summarization to provide users with a quick and easy way to stay informed about the latest news. By condensing news stories into brief, easy-to-read summaries, the app allows users to stay up-to-date on current events without having to spend a lot of time reading long articles.

- **ChatGPT:** As an AI language model, ChatGPT can also use text summarization to generate summaries of long texts. When a user inputs a long text, such as an article or a document, ChatGPT can generate a summary of the most important points of the text. ChatGPT uses a state-of-the-art natural language processing (NLP) and machine learning (ML) architecture known as GPT (Generative Pretrained Transformer). The GPT architecture uses a deep neural network to analyze and understand the input text, which allows it to generate high-quality summaries. To generate a summary, ChatGPT analyzes the input text to identify the most important sentences and phrases. It then uses this information to generate a shorter version of the text that captures the key ideas and themes. In addition to generating summaries, ChatGPT can also answer questions, generate text, and perform other language tasks. The model has been trained on a large corpus of text data, which allows it to generate high-quality summaries and other language output. Overall, ChatGPT uses text summarization as one of its many language capabilities, allowing it to quickly and accurately summarize long texts for a variety of use cases.

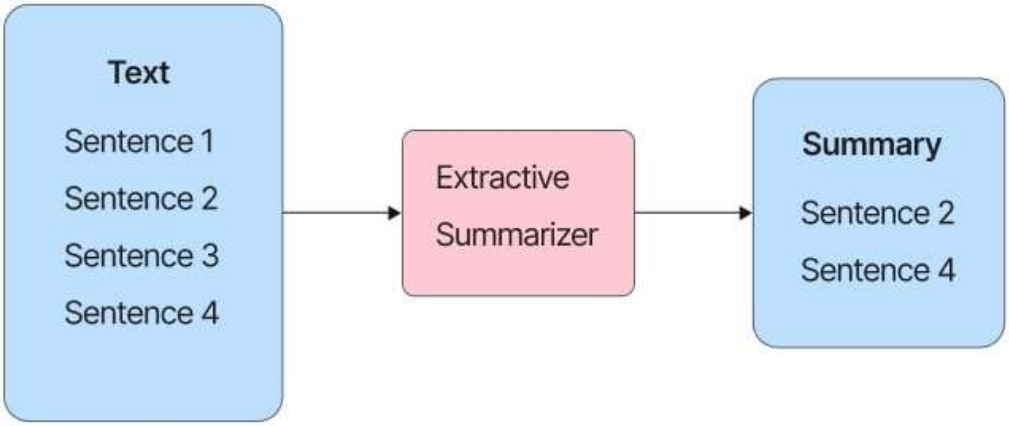
There are two main types of text summarization:

- **Extractive summarization:**
  It is a technique of summarizing a document by selecting a subset of its original sentences or phrases to create a condensed version that retains the most important information. In extractive summarization, the summary is composed of text snippets that are extracted directly from the original document, without any modification.

  The process of extractive summarization involves several steps, including:

  1. Text Pre-processing: The original document is pre-processed to remove any unnecessary information, such as stop words and punctuation marks.
  2. Sentence Extraction: The most important sentences in the document are selected based on their relevance to the main topic and the frequency of their occurrence in the document. Several algorithms can be used to score sentences based on various factors such as word frequency, word position, and sentence length.
  3. Ranking and Selection: The selected sentences are ranked according to their importance and relevance to the main topic. The top-ranked sentences are then combined to form the final summary.

  Extractive summarization is a widely used technique in various applications such as news article summarization, social media summarization, and scientific paper summarization. One of the advantages of extractive summarization is that it preserves the original meaning and context of the document, making it easier for readers to understand the content. However, it may not be able to generate summaries that capture the nuances and contextual information present in the original document.

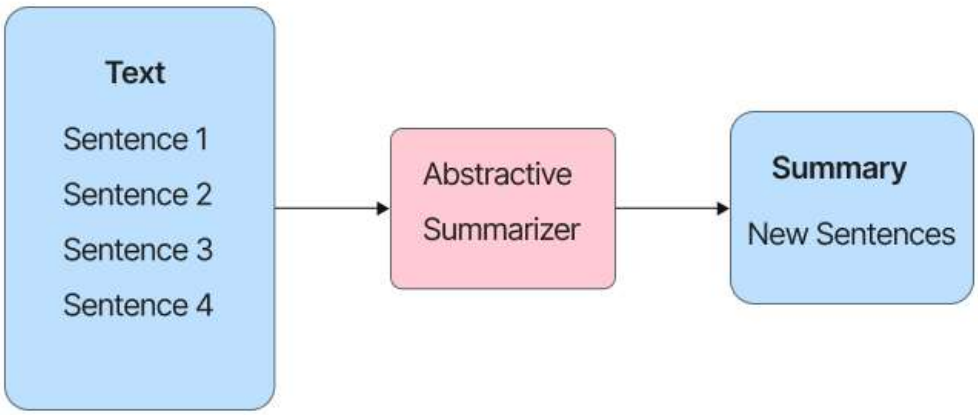


- **Abstractive summarization:**
  Abstractive summarization is a type of text summarization technique in which a summary is generated by interpreting and paraphrasing the original text, rather than selecting sentences or phrases directly. In other words, abstractive summarization aims to capture the meaning and context of the original text, rather than simply extracting sentences or phrases from it.

  Abstractive summarization requires more advanced natural language processing and machine learning techniques compared to extractive summarization. It involves analyzing the text at a deeper level, understanding the relationships between different words and phrases, and generating new sentences that capture the essence of the original text.

  One advantage of abstractive summarization is that it can produce more human-like summaries that are easier to read and understand. It can also capture more complex information and contextual details that may not be present in the original text. However, abstractive summarization can also be more challenging and resource-intensive compared to extractive summarization, as it requires more advanced algorithms and processing power.

  Overall, abstractive summarization is a powerful technique for summarizing text and has the potential to greatly enhance the accessibility and usefulness of large volumes of text data.



Objective

The objectives of this project are as follows:

- To eliminate manual efforts: One of the main objectives of this project is to eliminate the need for manual efforts in summarizing lengthy text documents. This can save time and resources and improve the efficiency of the summarization process.

- To create a shorter version of a given text document: Another objective of the project is to create a shorter version of a given text document while preserving the most important information. This can make the document more accessible and easier to understand for readers.

- To preserve the most important information contained in the original text: The project aims to ensure that the most important information contained in the original text is preserved in the summary. This can help readers to quickly grasp the main ideas of the document without having to read through the entire text.

- To allow readers to quickly grasp the main ideas of a document without having to read through the entire text: The ultimate objective of the project is to enable readers to quickly grasp the main ideas of a document without having to read through the entire text. This can improve the accessibility and usefulness of the document for a wide range of audiences.

By achieving these objectives, the text summarization project can provide a valuable tool for processing and understanding the vast amount of information available in digital text form.

Chapter 3
Literature review

Extractive and abstractive text summarization using deep learning and NLP techniques have been a popular research area in recent years. In this literature review, we will discuss some of the key research works related to this project.

- "TextRank: Bringing Order into Texts" by Rada Mihalcea and Paul Tarau (2004):

  This paper introduced TextRank, a graph-based method for extractive text summarization. TextRank uses PageRank algorithm to identify the most important sentences in a text based on the co-occurrence of words and phrases in the text.

- "Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond" by Ramesh Nallapati, Feifei Zhai, and Bowen Zhou (2016):

  This paper proposed an abstractive summarization model based on a sequence-to-sequence recurrent neural network (RNN) architecture. The model uses an attention mechanism to identify important words and phrases in the input text and generate a summary that captures the essential information.

- "Get To The Point: Summarization with Pointer-Generator Networks" by Abigail See, Peter J. Liu, and Christopher D. Manning (2017): (Abigail See, 2017)

  This paper proposed a pointer-generator network for abstractive text summarization that can generate a summary by copying words and phrases from the input text. The model uses a hybrid approach that combines extractive and abstractive techniques.

- "Attention Is All You Need" by Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. (2017):

  This paper introduced the transformer model, a new neural network architecture based solely on self-attention mechanisms. The transformer model has achieved state-of-the-art performance in various NLP tasks, including text summarization.

- "Fine-tune BERT for Extractive Summarization" by Yang Liu and Mirella Lapata (2019):

  This paper proposed a BERT-based model for extractive text summarization that fine-tunes a pre-trained BERT model on a summarization task. The model achieved state-of-the-art performance on several benchmark datasets.

- "Unified Language Model Pre-training for Natural Language Understanding and Generation" by Dongdong Zhang, Guoqing Luo, Haiqing Chen, et al. (2020):

  This paper proposed a unified language model pre-training (ULMPre) approach that can pre-train a single model for both language understanding and generation tasks, including text summarization. The model achieved state-of-the-art performance on several summarization datasets.

These research works demonstrate the effectiveness of deep learning and NLP techniques for extractive and abstractive text summarization.

In our project, we have leveraged these techniques to develop a summarization system that can generate high-quality and readable summaries for various types of texts.

Chapter 4
Basic Terminology

- **NLP:** Stands for Natural Language Processing. It is a field of computer science and artificial intelligence that focuses on enabling machines to understand, interpret, and manipulate human language. NLP is a subset of AI that deals with the interaction between computers and human language.

- **Pretrained Model**: Pre-trained models are machine learning models that have been trained on large datasets before being made available for use. These models have already learned to recognize patterns and features in data and can be fine-tuned or adapted for specific tasks, such as classification or generation. Pre-training is an essential step in the development of machine learning models, as it allows them to learn from large amounts of data and improve their performance. By training a model on a large dataset, it can learn to recognize patterns and features that are relevant to a particular task. For natural language processing tasks, pre-trained models are often used to generate language, classify text, or extract features from text. Pre-trained models are trained on vast amounts of text, and they learn the statistical relationships between words, phrases, and sentences. These models can then be used as a starting point for more specific tasks, such as sentiment analysis or text summarization, by fine-tuning them on smaller datasets. Pre-trained models can save a lot of time and resources because they are already trained on a large dataset, and they can be fine-tuned to specific tasks much faster than starting from scratch. Additionally, pre-trained models are often publicly available, which means that developers can use them without having to train their models from scratch, thus enabling more rapid development of new applications.

- **Document:** Refers to the original text that needs to be summarized.

- **Summary:** A shortened version of the document that retains the most important information.

- **Extractive Summarization:** A technique for summarization that involves selecting and combining sentences or phrases from the original text to form a summary.

- **Abstractive Summarization:** A technique for summarization that involves generating new sentences based on the understanding of the original text.

- **Tokenization:** The process of breaking down a document into smaller units such as words or phrases.

- **Stop words:** Words that are commonly used in a language but have little or no contribution to the meaning of a sentence, such as "the", "and", and "a".

- **Stemming:** The process of reducing a word to its base or root form.

- **Cosine similarity:** A measure of similarity between two documents based on the cosine of the angle between their vectors in a high-dimensional space.

- **Rouge score:** A metric for evaluating the quality of a summary based on its overlap with the original text.

Chapter 5
Methodology

Selection of OS

Microsoft Windows was used for this project because it is user friendly & it's robust. The code works on all OS.

Selection of Coding Language

- Python:

Python is a high-level, general-purpose programming language that is widely used for web development, data analysis, scientific computing, and many other purposes. It is known for its simplicity, readability, and flexibility, as well as its large and active developer community. Some of the key features of Python include A large standard library that supports many common programming tasks, such as connecting to web servers, reading and writing files, and working with data, An interactive interpreter, which allows you to try out code snippets and experiment with the language in an interactive environment, Support for object-oriented, imperative, and functional programming styles, Dynamically-typed, which means that you don't have to specify the data type of a variable when you declare it, Cross-platform compatibility, which means that Python programs can run on multiple operating systems.

Libraries Used

- Numpy: NumPy is a library for the Python programming language that is used for scientific computing and data analysis. It provides functions and utilities for working with large, multi-dimensional arrays and matrices of numerical data, as well as for performing mathematical operations on these data. One of the main advantages of NumPy is that it is very efficient for performing operations on large arrays and matrices, as it is implemented in C and can make use of the performance of compiled code.

- Pandas: Pandas is a library for the Python programming language that is used for data manipulation and analysis. It provides functions and utilities for working with tabular data, such as data stored in a spreadsheet or a database table. One of the main advantages of Pandas is that it provides a high-level interface for working with data, making it easy to perform tasks such as filtering, aggregating, and transforming data. It also integrates well with other libraries for data analysis, such as NumPy and Matplotlib.

- NLTK: NLTK is a toolkit build for working with NLP in Python. It provides us various text processing libraries with a lot of test datasets. A variety of tasks can be performed using NLTK such as tokenizing, parse tree visualization etc.

- Spacy: It is a free, open-source library for advanced Natural Language Processing (NLP) in Python. It can be used to build information extraction or natural language understanding systems, or to pre-process text for deep learning.

- Gensim: Gensim is a very handy python library for performing NLP tasks. The text summarization process using gensim library is based on TextRank Algorithm.

- Transformers: Transformers provides thousands of pretrained models to perform tasks on different modalities such as text, vision, and audio.

- Gradio: Gradio is a Python library for quickly creating customizable UIs for machine learning models, deep learning models, and other data science projects. It allows you to build interfaces for your models using a simple and intuitive API, without requiring any knowledge of web development. With Gradio, you can create user-friendly interfaces that allow users to interact with your models using text inputs, sliders, checkboxes, and other familiar UI components. This makes it easy to demonstrate the capabilities of your models, test their accuracy, and get feedback from other users. Gradio also includes built-in support for a wide variety of machine learning frameworks, including TensorFlow, PyTorch, and scikit-learn, so you can easily integrate your models into your Gradio interface. Additionally, Gradio provides a cloud-based deployment platform that makes it easy to share your models with others.

Algorithms Used

Text Rank

Text Rank is an extractive algorithm and an unsupervised approach to text summarization that does not require pre-defined rules or training data.

Instead, it relies on the analysis of relationships between sentences in a text document to determine the most important ones.

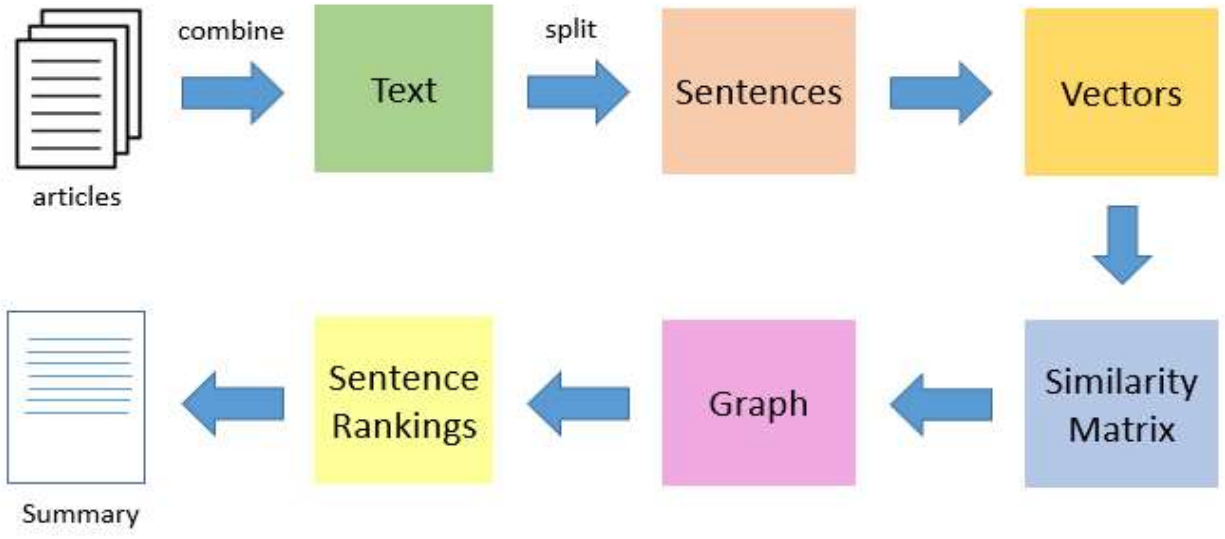
This makes it a flexible and adaptable method for summarization, as it can be applied to a wide range of text documents without the need for specific training or customization.

The algorithm calculates sentence importance based on the sum of scores of the sentences that connect to it in the graph.

In each iteration, the score of each sentence is updated based on the scores of its neighbouring sentences until convergence is achieved.

The sentences with the highest scores are then selected to form the summary.

Let's take a look at the flow of the Text Rank algorithm that we will be following:



Steps:

1. The first step would be to concatenate all the text contained in the articles
2. Then split the text into individual sentences
3. In the next step, we will find vector representation (word embedding) for each and every sentence
4. Similarities between sentence vectors are then calculated and stored in a matrix
5. The similarity matrix is then converted into a graph, with sentences as vertices and similarity scores as edges, for sentence rank calculation
6. Finally, a certain number of top-ranked sentences form the final summary

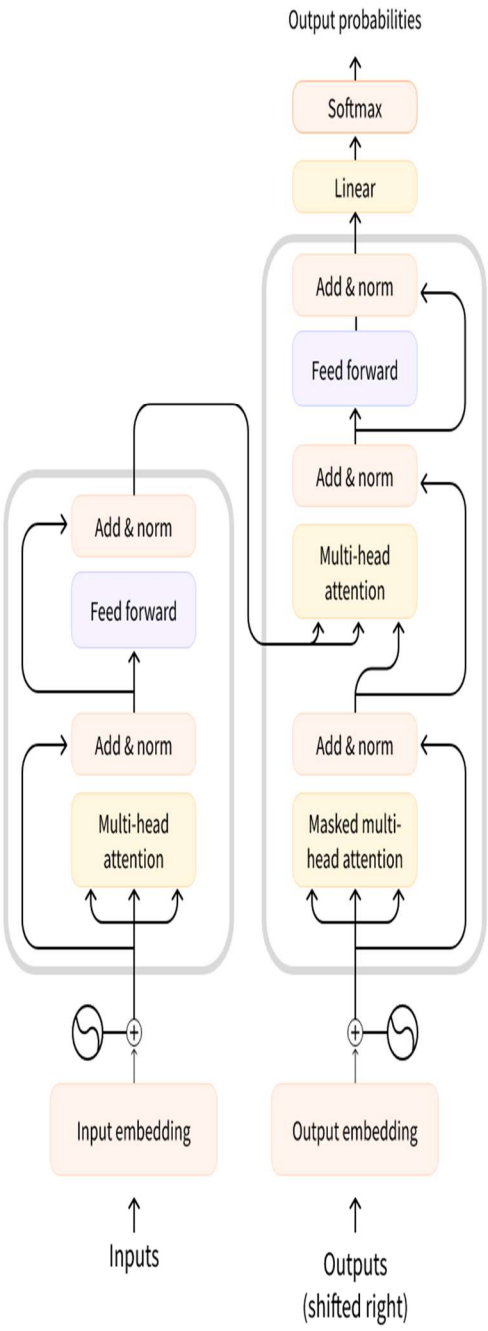
Transformers

Transformers are a type of neural network architecture that has become very popular in natural language processing (NLP). The transformer architecture was introduced by Vaswani et al. in their 2017 paper "Attention Is All You Need".

The transformer architecture uses a self-attention mechanism to model the relationships between different parts of a text sequence. This allows the model to learn dependencies between any two positions in the input sequence, regardless of their distance from each other.

The transformer architecture consists of an encoder and a decoder, both of which are composed of multiple layers of self-attention and feedforward networks. The encoder processes the input sequence and generates a sequence of hidden states, which represent the input information. The decoder then generates the output sequence by attending to the hidden states of the encoder and predicting the next token in the sequence.

Transformers have several advantages over traditional recurrent neural networks (RNNs) and convolutional neural networks (CNNs) for NLP tasks. They can handle long sequences of text more efficiently, are more parallelizable, and can capture dependencies between distant parts of the input sequence. As a result, transformers have become the backbone of many state-of-the-art NLP models, including BERT, GPT, and T5.

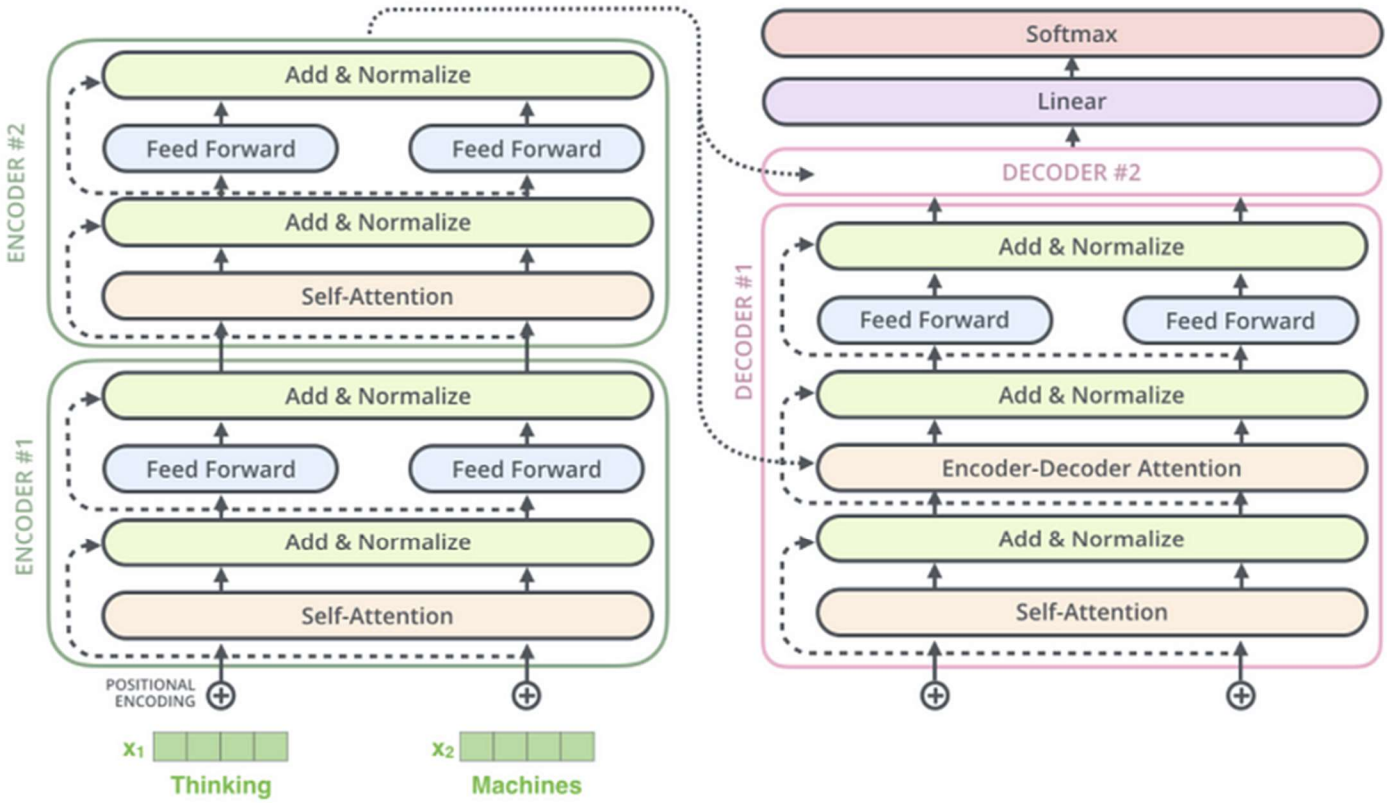

Output probabilities
Softmax
Linear
Add & norm
Feed forward
Add & norm
Multi-head attention
Add & norm
Masked multi-head attention
Add & norm
Feed forward
Add & norm
Multi-head attention
Input embedding
Output embedding
Inputs
Outputs (shifted right)

T5

T5 (Text-to-Text Transfer Transformer) is a natural language processing algorithm based on the Transformer architecture. T5 is designed to handle these tasks using a "text-to-text" approach, which means that it casts all NLP tasks as text-to-text transformations. This approach allows T5 to handle a wide range of tasks with a single, unified model.

The training process for T5 involves pre-training on a large corpus of text data and fine-tuning on specific tasks to achieve state-of-the-art performance. The model is trained on a diverse set of tasks using a unified objective function, which helps it learn to generalize across different NLP tasks.

T5 is useful for text summarization because it is a powerful language model that can learn to generate summaries from input texts in an unsupervised manner. T5's ability to handle multiple languages and diverse datasets also makes it a versatile tool for text summarization.



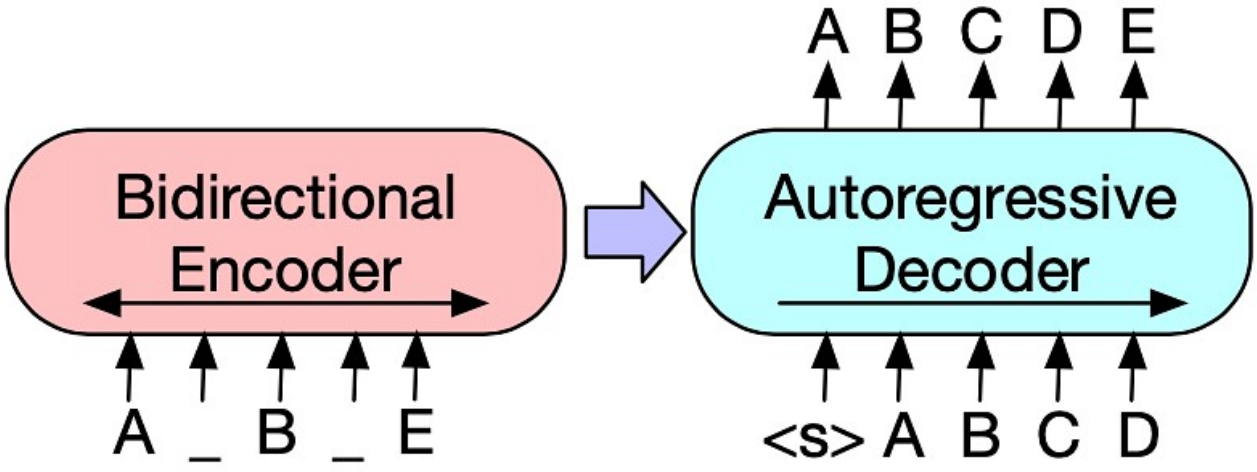
BART

BART (Bidirectional and Auto-Regressive Transformer) is a transformer-based language model developed by Facebook AI Research (FAIR) in 2019. It is designed to handle a wide range of natural language processing (NLP) tasks, including text generation, summarization, and question answering.

BART's architecture is based on the Transformer architecture used in other transformer-based models like GPT (Generative Pre-trained Transformer). It consists of an encoder-decoder architecture with multiple layers of self-attention and feedforward networks in both the encoder and decoder.

BART is trained using a combination of masked language modeling and denoising auto-encoding objectives. During training, a portion of the input sequence is randomly masked, and the model is trained to predict the masked tokens. BART is also trained to reconstruct the original input sequence from a corrupted version of the sequence.

BART has achieved state-of-the-art performance on various NLP tasks, such as language translation, summarization, and question answering. Its ability to handle bidirectional and auto-regressive tasks makes it a versatile and powerful tool in natural language processing.



GUI

- **Extractive Summarizer**

```
# Initialize the summarization pipeline
summarizer = pipeline("summarization")

def summarize_text(text):
    # Generate summary
    summary = summarizer(text, max_length=100, min_length=30, do_sample=False)[0]['summary_text']

    # Count the input and output words
    input_words = len(text.split())
    output_words = len(summary.split())

    return summary, str(input_words), str(output_words)

# Define the input and output interfaces for the Gradio app:
input_text = gr.inputs.Textbox(lines=10, label="Input Text")
output_text = gr.outputs.Textbox(label="Output Summary")
input_word_count = gr.outputs.Textbox(label="Input Word Count")
output_word_count = gr.outputs.Textbox(label="Summary Word Count")

# Create the Gradio app and run it:
summarization_app = gr.Interface(fn=summarize_text, inputs=input_text, outputs=[output_text, input_word_count, output_word_count], title="Extractive
summarization_app.launch()
```

This code creates a Gradio interface for a text summarization model using the Hugging Face transformers library.

The pipeline function from the transformers library is used to initialize a summarization pipeline, which is a pre-trained model that can be used for text summarization.

The summarize_text function takes an input text and generates a summary using the summarization pipeline.

The input text is first processed by the pipeline, and then the summary text is extracted from the pipeline output.

The function also calculates the number of input and output words using the len function and returns them along with the summary text.

The Gradio interface is defined using gr.inputs.Textbox and gr.outputs.Textbox for the input and output, respectively. Additionally, gr.outputs.Textbox is used twice to output the word count of the input and output texts.

Finally, the Gradio app is launched using summarization_app.launch(), which creates a web interface where users can input text and receive a summary along with the input and output word counts.

- **Abstractive Summarizer**

```
MODELS = {
    "BART": "facebook/bart-large-cnn",
    "T5": "t5-base"
}

def summarize_text(text, model_choice):
    # Select the appropriate model based on user input
    model_name = MODELS[model_choice]
    model = pipeline("summarization", model=model_name)

    # Generate summary
    summary = model(text, max_length=100, min_length=30, do_sample=False)[0]['summary_text']

    # Count the input and output words
    input_words = len(text.split())
    output_words = len(summary.split())

    return summary, str(input_words), str(output_words)

# Define the input and output interfaces for the Gradio app:
input_text = gr.inputs.Textbox(lines=10, label="Input Text")
model_choice = gr.inputs.Radio(list(MODELS.keys()), label="Select Model")
output_text = gr.outputs.Textbox(label="Output Summary")
input_word_count = gr.outputs.Textbox(label="Input Word Count")
output_word_count = gr.outputs.Textbox(label="Summary Word Count")
```

This code defines a Gradio interface for an abstractive summarization app that uses two pre-trained models, namely BART and T5, provided by the Hugging Face Transformers library. The code first defines a dictionary called "MODELS" that maps the names of the available summarization models to their corresponding Hugging Face model IDs.

The "summarize_text" function takes as input the text to be summarized and the user's selected model. It then loads the selected model and uses it to generate a summary of the input text by setting the maximum and minimum lengths of the generated summary and using greedy decoding.

The function also counts the number of words in the input text and the generated summary and returns them along with the generated summary.

The code then defines the input and output interfaces for the Gradio app using the Gradio library. The input interface includes a textbox for the input text and a radio button group for selecting the summarization model. The output interface includes a textbox for the generated summary and two more textboxes for displaying the number of input and output words.

Finally, the Gradio interface is created using the "summarize_text" function as the main function, and the input and output interfaces are specified as the inputs and outputs of the interface. The title of the interface is set to "Abstractive Summarizer", and the interface is launched using the "launch()" method of the Gradio interface object.

Chapter 6
Data Analysis

Data pre-processing is an important step in text analysis and summarization. It involves cleaning and standardizing the input text so that it can be effectively analyzed and summarized. Here are some common techniques used in data pre-processing:

- Lower casing - This step involves converting all the text to the same casing format, usually lower case. This ensures that capitalization does not affect the summarization process.

- Eliminate punctuation, HTML tags, and links - Removal of punctuation, links, and HTML tags that do not add meaning to the text is essential to standardize the text. Punctuation like "!"#$%&'()*+,-./:;<=>?@[]^_{|}~`" is not necessary for summarization and can be eliminated.

- Eliminate stop words and frequently occurring words - Common words such as 'the', 'a', 'and', etc. are frequently used in a text but do not provide valuable information for downstream analysis. These words are called stop words and should be eliminated from the input text.

- Stemming - Stemming is the process of reducing the inflected words to their root form. For example, 'running', 'run', and 'ran' are all variations of the same word 'run'. Stemming reduces these variations to the root word, making it easier to analyze.

- Lemmatization - Lemmatization is the process of reducing derived words to their base or root form while making sure that root words belong to the language. This process helps to reduce the variations in words that are used in a text.

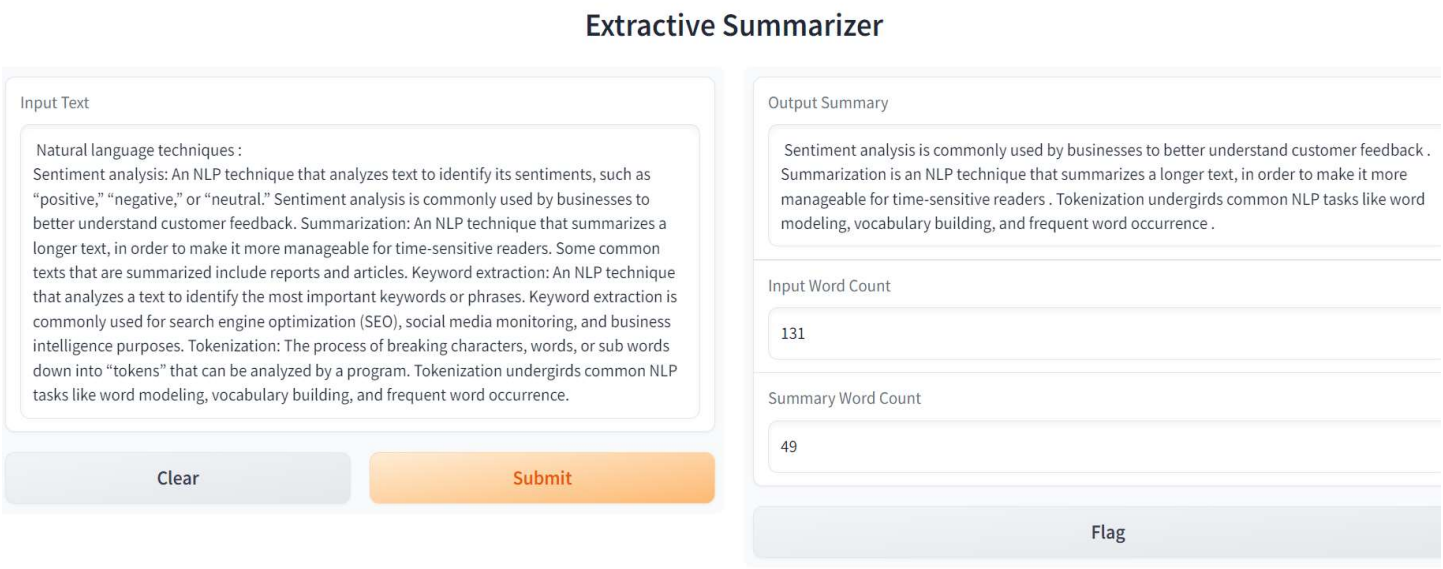
- Contraction mapping - Contraction mapping involves expanding the shortened version of words or syllables. For example, ‘can’t’ can be expanded to ‘cannot’ to ensure that the text is standardized.

By implementing these techniques in data pre-processing, the input text can be standardized and prepared for the summarization process. This ensures that the summarization algorithm can accurately analyze and summarize the text.

```
[ ] sentences=sent_tokenize(text)

[ ] nltk.download('stopwords')

    [nltk_data] Downloading package stopwords to /root/nltk_data...
    [nltk_data]   Unzipping corpora/stopwords.zip.
    True

    sentences_clean=[re.sub(r'[^\w\s]','',sentence.lower()) for sentence in sentences]
    stop_words = stopwords.words('english')
    sentence_tokens=[[words for words in sentence.split(' ') if words not in stop_words] for sentence in sentences_clean]

[ ] w2v = Word2Vec(sentence_tokens, vector_size=1, min_count=1, epochs=1000)
    sentence_embeddings = [[w2v.wv[word][0] for word in words] for words in sentence_tokens]
    max_len = max([len(tokens) for tokens in sentence_tokens])
    sentence_embeddings = [np.pad(embedding, (0, max_len - len(embedding)), 'constant') for embedding in sentence_embeddings]

[ ] similarity_matrix = np.zeros([len(sentence_tokens), len(sentence_tokens)])
    for i,row_embedding in enumerate(sentence_embeddings):
        for j,column_embedding in enumerate(sentence_embeddings):
            similarity_matrix[i][j]=1-spatial.distance.cosine(row_embedding,column_embedding)
```

Result & Discussion

Extractive Summarizer:



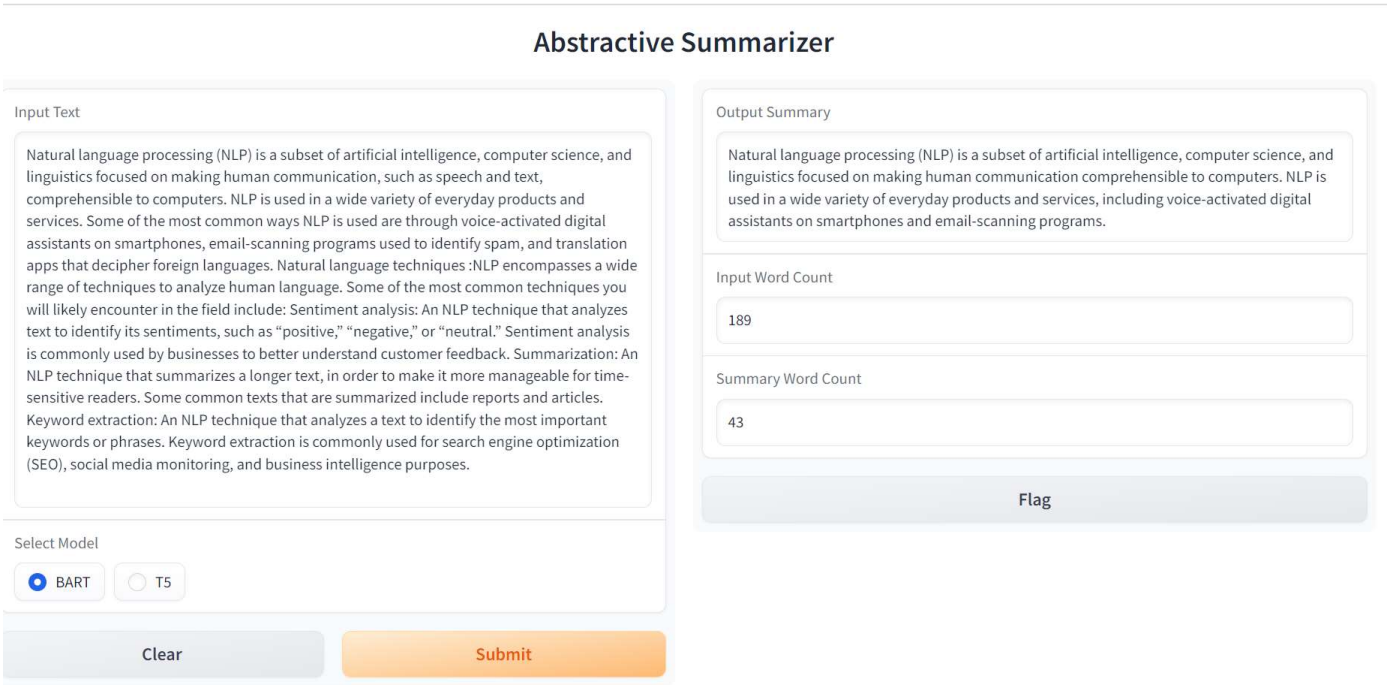
The result of this code is a web interface for a text summarization model that uses the Hugging Face transformers library. The interface allows users to input text and receive a summary of the text along with the word count of the input and output text.

As in the above snapshot of Extractive Summarizer, user had input a text where input text has 131 words and the output summary has 49 words

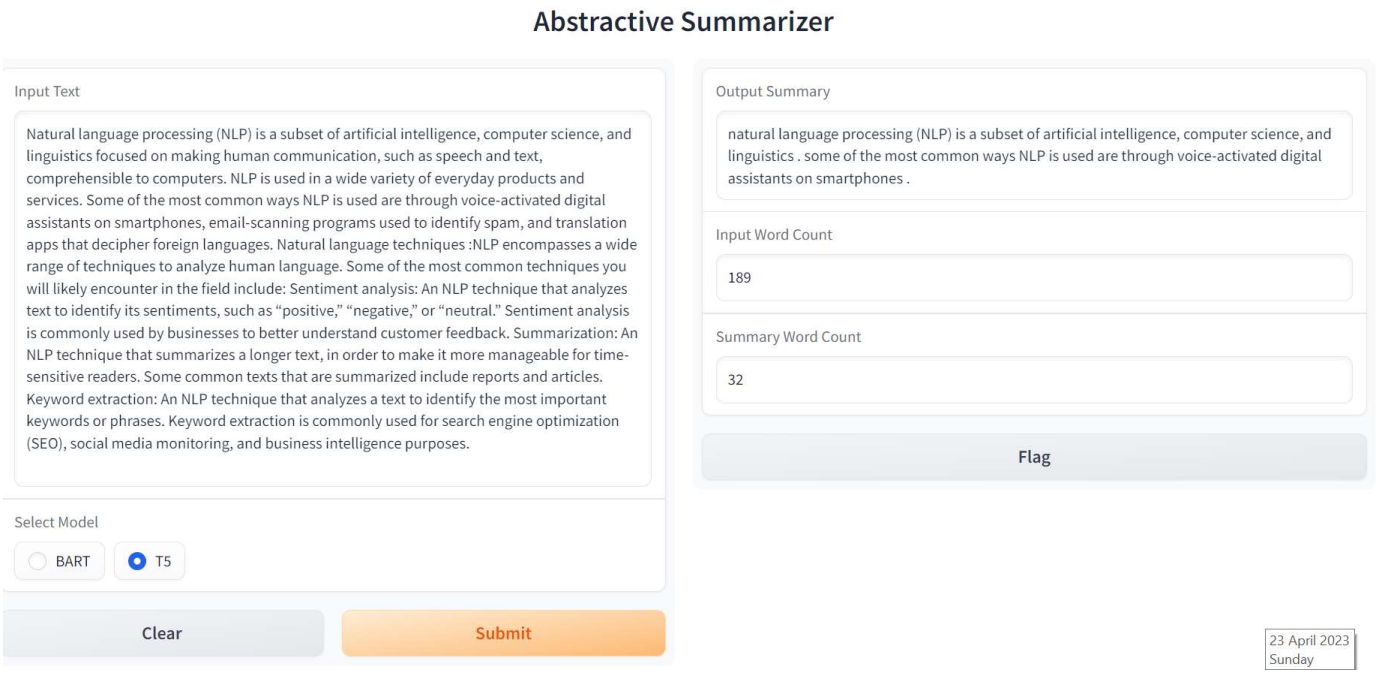
Overall, this code provides a simple and effective way for users to generate summaries of text using a pre-trained text summarization model. The Gradio interface makes the model accessible to anyone, regardless of their technical expertise, and the summarization model is capable of generating high-quality summaries of text.

Abstractive Summarizer:

- **Abstractive Summarizer using BART**



- **Abstractive Summarizer using T5**



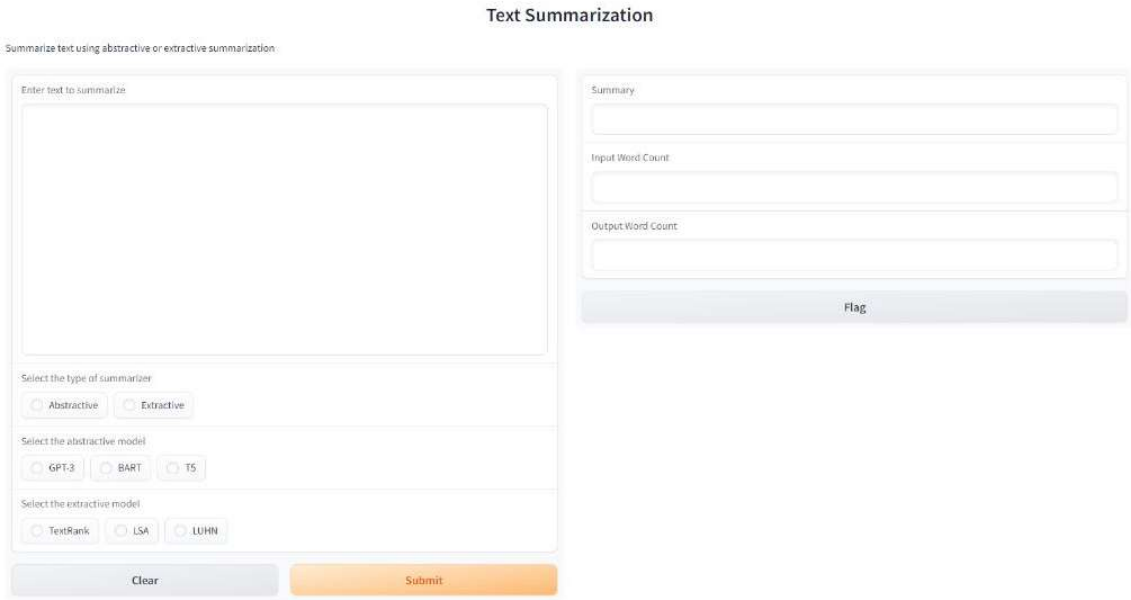
This code launches a Gradio interface for abstractive text summarization. The user can input a text in the input text box and select a model (BART or T5) using the radio buttons. After submitting the input, the selected model is used to generate a summary of the text, and the summary along with the input and output word count is displayed in the output text boxes.

As in the above snapshot of Abstractive Summarizer, user had input an text and selected the model BART from the options given where input text has 189 words and the output summary has 43 words

As in the above snapshot of Abstractive Summarizer, user had input a text and selected the model T5 from the options given where input text has 189 words and the output summary has 32 words

Overall, this code provides a user-friendly interface for text summarization using pre-trained models, which can be useful for various applications such as news article summarization, document summarization, and more.

- **Common Interface**



This code defines a interface for text summarization using either abstractive or extractive summarization techniques.

The code defines a Gradio interface with input fields for the text to be summarized, the type of summarization technique to use (abstractive or extractive), and the specific model to use for each technique. The interface also includes output fields for the summary itself, as well as the word count of the input and output texts.

The summarizer_callback function is defined to handle the input and generate the output for the Gradio interface. It takes the input text, summarization type, and specific models as arguments, counts the number of words in the input text, and then calls the appropriate summarization function based on the selected technique and model. It then counts the number of words in the output summary and returns the summary and the input and output word counts.

Finally, the Gradio interface is launched using the launch() method. When the interface is launched, users can enter text to be summarized and select the desired summarization technique and model. The interface then generates a summary using the selected technique and model, and displays the summary and word counts in the output fields.

But due to computational limitations we could not make this interface run successfully for every model.

Conclusion

Extractive summarization model often produced grammatically correct summaries, as it relies on information directly taken from the original text.

While Abstractive summarization generated more natural and fluent summaries that are not limited by the exact phrasing of the original text.

Abstractive summarization models can also better capture the context and meaning of the text, allowing them to generate more informative and coherent summaries.

However, abstractive summarization is generally more challenging and requires more advanced deep learning techniques, larger amounts of training data, and more computational resources.

Therefore, the choice between extractive and abstractive summarization depends on the specific requirements of the task, such as the desired level of summarization, the quality and fluency of the summary, and the available resources and constraints.

In some cases, a combination of both techniques may also be used to produce a summary that combines the strengths of both approaches.

In this project, we evaluated TextRank, LexRank, and LSA algorithms for extractive summarization and pre-trained transformer models, namely T5 and BART, for abstractive summarization.

Based on our findings, we concluded that TextRank algorithm outperformed the other two for extractive summarization, while the fine-tuned T5 model generated a fluent and accurate summary for abstractive summarization.

To compare the performance of each model, we computed ROUGE scores and found that the TextRank for extractive and T5 for abstractive models achieved better results than all the other models.

Bibliography

Bibliography

Abigail See, P. J. (2017). Get to the point: Summarization with point-generator networks.

Ashish Vaswani, N. S. (2017). Attention is all you need.

Chadrasekhar, G. (2020, August). *Understanding Abstractive Text Summarization from Scratch.* Retrieved from Towards AI: https://pub.towardsai.net/understanding-abstractive-text-summarization-from-scratch-baaf83d446b3

docs. (n.d.). Retrieved from gradio: https://gradio.app/docs/

Dongdong Zhang, G. L. (2020). Unified Language Model Pre-training for Natural Language Understanding and Generation.

Duseja, S. (2020, August). *Text Summarization Using Deep Neural Networks.* Retrieved from Towards Data Science: https://towardsdatascience.com/text-summarization-using-deep-neural-networks-e7ee7521d804

Gradio Docs. (n.d.). Retrieved from gradio: https://gradio.app/docs/

Joshi, P. (2018, February). *An Introduction to Text Summarization using the TextRank Algorithm.* Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2018/11/introduction-text-summarization-textrank-python/

Jwalapuram, N. (2021, August). *An Introduction to Natural Language Processing with Transformers*. Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2021/08/an-introduction-to-natural-language-processing-with-transformers/

Lapata, Y. L. (2019). Fine-tune BERT for Extractive Summarization.

NLP-Course. (n.d.). Retrieved from Hugging Face: https://huggingface.co/learn/nlp-course/chapter1/2?fw=pt

Pai, A. (2019, June). *Comprehensive Guide to Text Summarization using Deep Learning in Python.* Retrieved from Analytics Vidhya:

https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/

Ramesh Nallapati, F. Z. (2016). Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond.

Tarau, R. M. (2004). TextRank: Bringing Order into Texts .

Transformers Docs. (n.d.). Retrieved from Hugging Face: https://huggingface.co/transformers/v3.0.2/index.html

Future work

Here are some potential future directions in the field of text summarization:

- **Improving the quality of summaries:** While current state-of-the-art models for text summarization can produce high-quality summaries, there is still room for improvement. Future work could focus on developing more sophisticated algorithms that can generate more accurate and concise summaries.

- **Multi-document summarization:** Summarizing a single document is a relatively straightforward task, but summarizing multiple documents on the same topic is much more challenging. Future work could focus on developing models that can extract information from multiple sources and generate a coherent summary.

- **Summarization for different domains:** Summarization models are typically trained on generic text, but they may not perform as well on domain-specific text, such as scientific articles or legal documents. Future work could focus on developing models that are specifically designed for summarizing text in different domains.

- **Better evaluation metrics:** Current evaluation metrics for summarization, such as ROUGE and BLEU, have limitations in measuring the quality of summaries. Future work could focus on developing more robust evaluation metrics that take into account factors such as coherence and readability.

- **Extractive vs. abstractive summarization:** Extractive summarization involves selecting and reorganizing existing text to create a summary, while abstractive summarization involves generating new text to convey the key information in a document. Future work could focus on developing models that can perform both extractive and abstractive summarization more efficiently, depending on the task at hand.