Text Summarization a ML Vs Non ML comparison

Pavan Yachamaneni

CMSE 890: Advanced Machine Learning
Spring 2023

Table of Contents

1. Introduction

Summarization is the process of reducing a larger piece of text into a shorter version while preserving its most important information. There are two main approaches to summarization:

Extractive summarization involves selecting and extracting the most important sentences or words from the original text to create a summary. The extracted sentences or words are usually taken directly from the original text, without any modification. This method is relatively simple and easy to implement but may not produce a coherent summary if the original text is complex or includes a lot of redundancy.

Abstractive summarization, on the other hand, involves generating new text that captures the main ideas of the original text using its own wording. This method requires a more sophisticated approach that can understand the nuances of language, including its syntax and context. Abstractive summarization models use machine learning techniques to analyze the input text and generate a summary that is coherent and concise.

2. Project Motivation and Assumptions

The Internet and computer technology have made vast amounts of information easily accessible. Although this presents great opportunities, it also poses challenges. One of these challenges is the limited ability of the human brain to process such a large amount of information in a useful and efficient way. Moreover, online publications often prioritize grabbing readers' attention with sensational headlines to increase their page views, rather than focusing on conveying accurate information and this actually creating new opportunities for story telling in different ways with technology incorporated.

To address these challenges, Natural Language Processing (NLP) models have many potential applications. These models can perform tasks such as content classification, text translation, and automatic summarization.

Text summarization is a widely researched area in the field of Natural Language Processing (NLP). With the ever-increasing amount of information available online, summarization techniques have become increasingly important to provide users with an efficient and concise way to consume information. Summarization techniques can be broadly classified into two categories: ML (Machine Learning)-based and non-ML-based techniques.

The purpose of this report is to compare these two categories of summarization techniques. I had explored the advantages and disadvantages of each approach. Through this report, I aim to provide insights and guidance for researchers and practitioners in the field of NLP, particularly those interested in text summarization.

3. Methodology

For the entire project, In the process of how data is processed and prepared. While ML and Non-ML approaches share the same initial steps in data preparation, such as cleaning and preprocessing, the differences become apparent in the summarization stage. ML approaches require the dataset to be divided into training, testing, and validation sets, each with its own role. Non-ML approaches, on the other hand, do not involve a training stage. Instead, they rely on predefined rules that are processed to achieve the final objective of summarization. Below there are illustrative figures to provide a detailed explanation of the key differences between these two approaches.

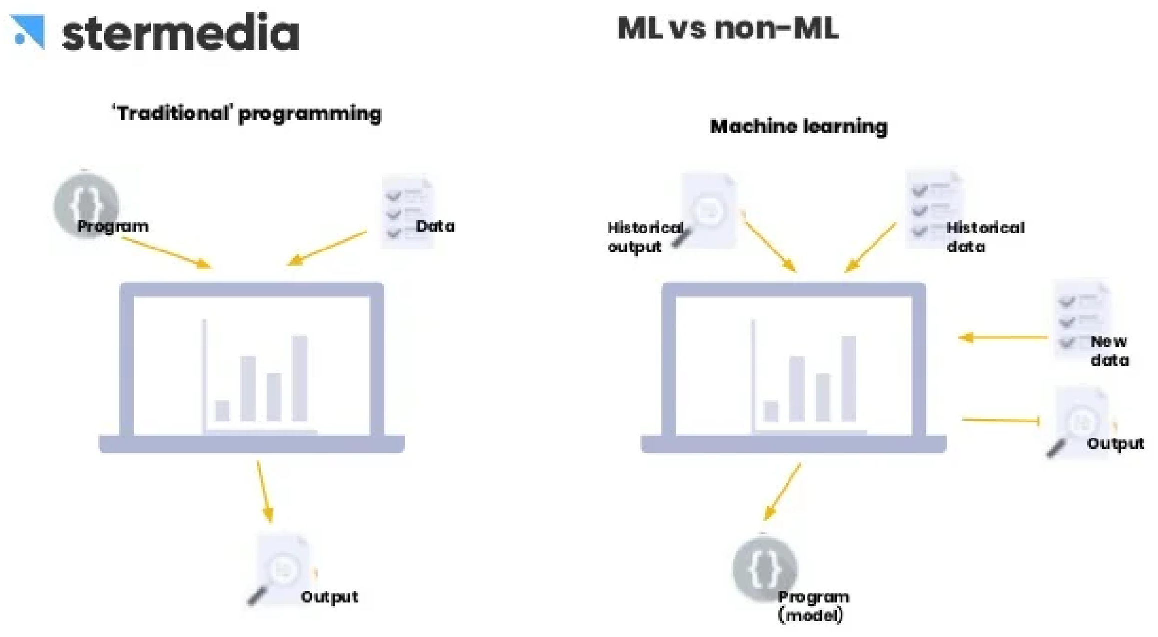


Fig-1: ML Vs Non-ML comparison, Source: Python and ML applications in Industry

For Non-ML approaches, there is a predefined program that takes data and gives output. **But for ML** approaches, we use data to train and get the program that further gives output.

This report will first provide an overview of contemporary summarisation techniques in the context of **Extractive summarization**. After introducing the methodology, a detailed walkthrough of construction of a sequence-to-sequence model is given. This model will generate short summaries (headlines) of up to 20 words for news articles.

4. Dataset Description

The CNN/Daily Mail dataset is a popular dataset used for text summarization tasks. It was introduced in 2015 by the researchers at the University of Oxford and the dataset is based on articles from the CNN and Daily Mail news websites.

The dataset consists of over 300,000 news articles along with corresponding summaries written by professional editors. The articles are relatively long, ranging from 200 to 800 words, while the summaries are much shorter, typically 3 to 7 sentences long. The articles and summaries are paired together so that a model can be trained to generate a summary of an article given the text of the article.

The CNN/Daily Mail dataset is often used for benchmarking summarization models due to its large size and high quality summaries. The dataset is divided into training, validation, and test sets, with roughly 287,000, 13,000, and 11,000 examples respectively. Further details are given in Exploratory Data Analysis.

Particularly, scope for this dataset is more advanced and further beyond this project. The CNN/Daily Mail dataset has been used as a benchmark for several text summarization techniques, including extractive and abstractive summarization methods.

One extractive approach that has been benchmarked on this dataset is the LexRank algorithm, which uses graph-based centrality measures to identify the most important sentences in a text. Another extractive technique is the TextRank algorithm, which is similar to **LexRank** but uses a slightly different approach to computing sentence importance.

On the other hand, abstractive summarization models like the Pointer-Generator Network and the Transformer have also been benchmarked on the CNN/Daily Mail dataset. The **Pointer-Generator Network** is a neural network that learns to generate abstractive summaries by copying words from the source text, while also learning to generate new words when necessary. The Transformer model, on the other hand, is a more recent neural network architecture that has shown strong performance on a variety of natural language processing tasks, including text summarization. Overall, the

CNN/Daily Mail dataset has been a useful benchmark for evaluating the performance of different text summarization techniques.

However, it's worth noting that the dataset is relatively small and may not fully capture the complexity and diversity of real-world text summarization tasks. Therefore, researchers often use other larger and more diverse datasets, such as the Webis-TLDR-17 dataset, to further evaluate the performance of text summarization models.

5. Exploratory Data Analysis

Exploratory data analysis (EDA) is an essential step in understanding any data, including text data. Here are some common EDA techniques used in analyzing text data:

1. **Dataset Distribution**: Analyzing the distribution of the text dataset is also an important EDA technique that can help us understand the data better. For text datasets, we can analyze the distribution of various characteristics such as the length of the text, the number of words, and the frequency of different parts of speech.

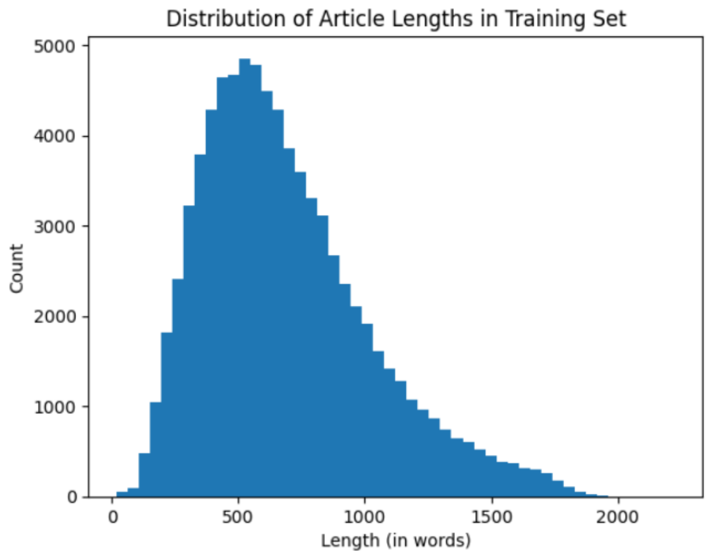
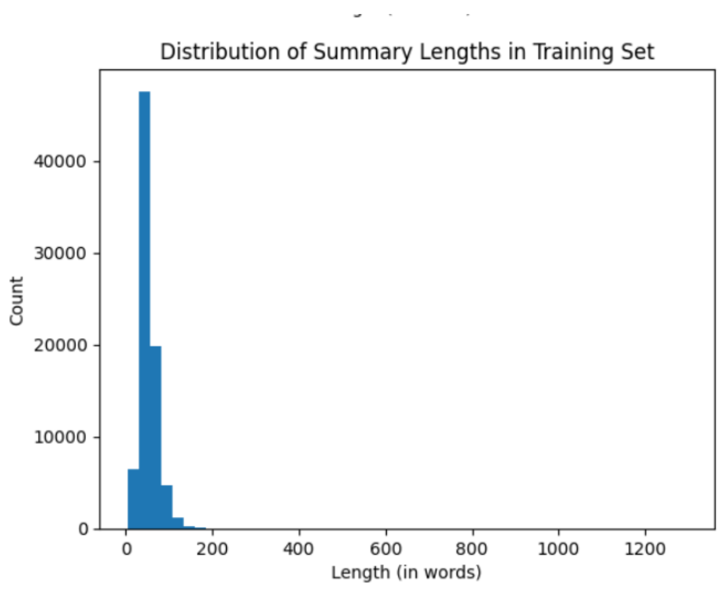
    a. 

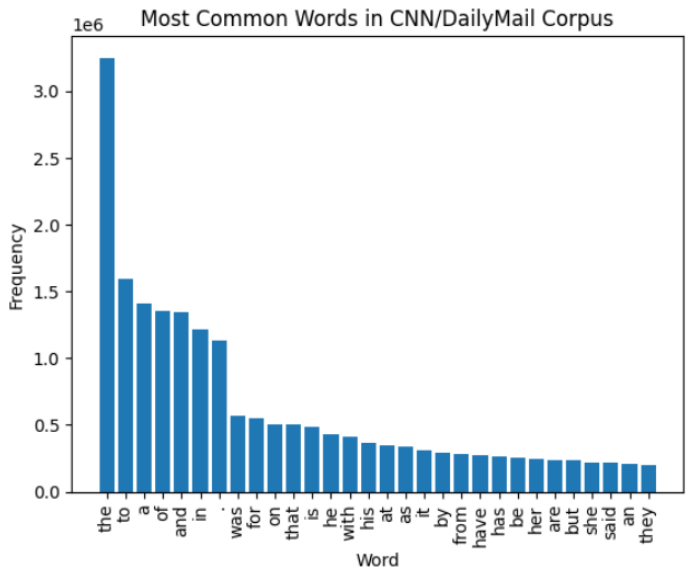
Fig-2: Articles Distribution



b.

Fig-3: Summary Distribution

2. **Word Frequency Analysis**: This technique involves counting the number of times each word appears in a document or corpus. It is useful in identifying the most common words and phrases used in a text dataset.



a.

Fig-4: Most Common Words before stop words removal

3. **Stop Word Removal**: Stop words are common words that are often irrelevant to the analysis, such as "the," "and," and "a." Removing them can help to reduce noise in the dataset and improve the accuracy of the analysis.

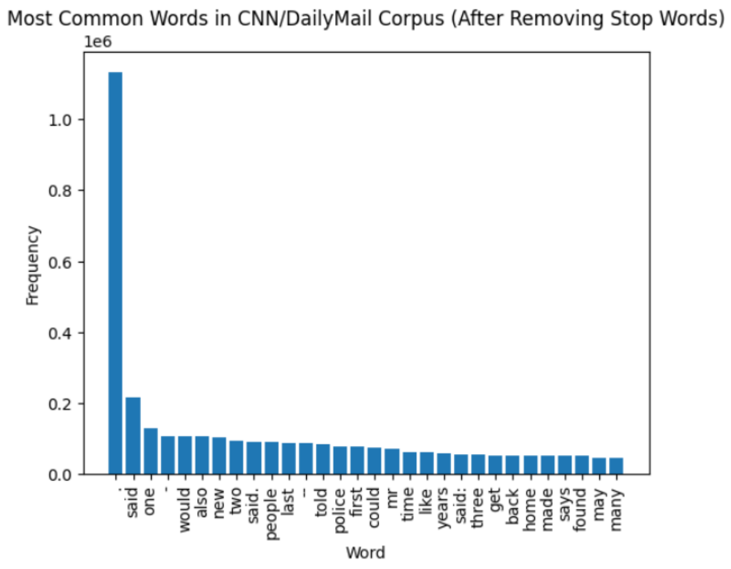
a. 

Fig-5: Most Common Words after stop words removal

4. **N-grams**: N-grams are a technique used in natural language processing to analyze sequential data, such as text. An n-gram is a sequence of n contiguous words in a text, where n can be any integer greater than or equal to 1. For example, in the sentence "The quick brown fox jumps over the lazy dog," some possible n-grams are:
    a. 1-grams (or unigrams): The, quick, brown, fox, jumps, over, the, lazy, dog. I have not done 1 gram as it does not make any sense for larger datasets.
    b. 2-grams (or bigrams): The quick, quick brown, brown fox, fox jumps, jumps over, over the, the lazy, lazy dog. For my dataset, 2-grams are:

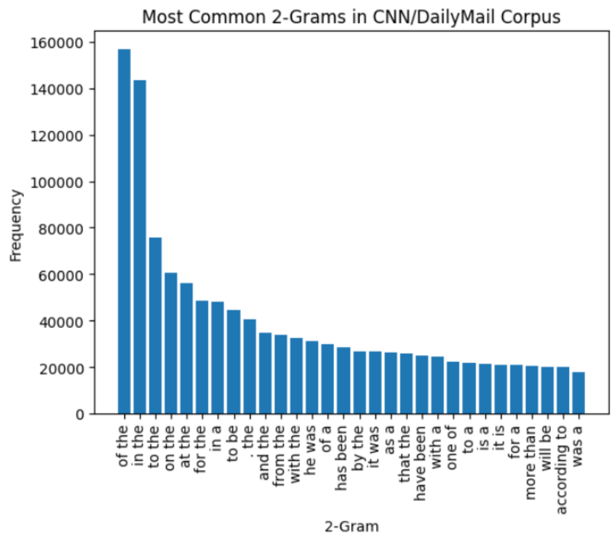
i. 

Fig-6

c. 3-grams (or trigrams): The quick brown, quick brown fox, brown fox jumps, fox jumps over, jumps over the, over the lazy, the lazy dog

i.

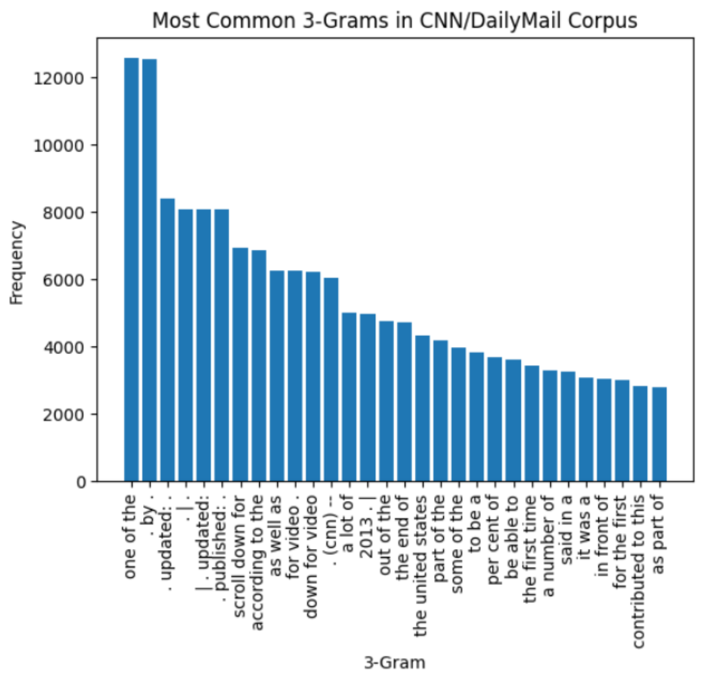


Fig-7

d. 4-grams:

i.

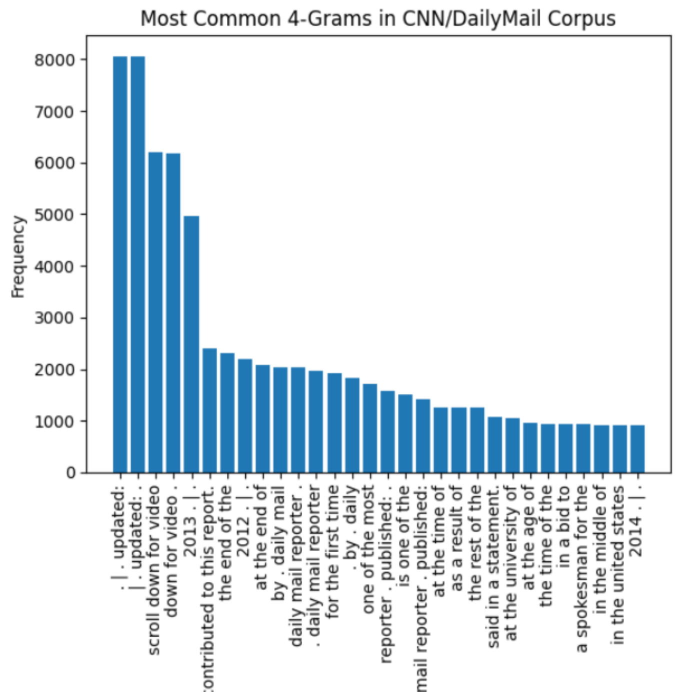


Fig-8

5.a) Inference From EDA: Here are some inferences that can be drawn from the EDA on the CNN/DailyMail dataset:

a. Dataset Distribution: The distribution of articles and summaries across different categories is not balanced. Some categories have significantly more articles and summaries than others.
b. Word Frequency Analysis: The most common words in the corpus are stop words like 'the', 'and', 'in', etc. These words don't carry much meaning and can be removed for better text summarization.
c. Stop Word Removal: After removing the stop words, the most common words are more meaningful and provide better insights into the dataset. It can be observed that the most common words are related to news and current events.
d. N-grams: The most common n-grams in the corpus are mostly 2-grams, indicating that bigrams can be useful for text summarization. The most common n-grams are related to news and current events. And, there are various punctuation marks that need to be processed before preparing and modeling data.

Overall, the EDA on the CNN/DailyMail dataset indicates that the dataset is well-suited for text summarization tasks, and that removing stop words and using bigrams can be useful techniques for improving text summarization performance. Link for EDA file in github: Link

6. Data Preparation

1. Text Cleaning: This step involves removing any irrelevant or unwanted characters or symbols from the text, such as HTML tags, URLs, special characters, etc. This can be done using various techniques such as regular expressions, BeautifulSoup, etc.
2. Removing stop words: Stop words are common words that do not carry much meaning, such as "the", "and", "is", etc. Removing stop words helps to reduce the size of the data and improve the efficiency of the summarization algorithm.
3. Handling text contractions: Text contractions such as "can't", "won't", "it's", etc. should be expanded to their full forms to avoid any ambiguity in the text.
4. Lemmatizing: Lemmatization involves converting words to their base or root form, which can help to reduce the size of the data and improve the accuracy of the summarization algorithm.
5. Regex operations for removing punctuations, and other redundant patterns: This step involves using regular expressions to remove any unnecessary punctuation marks or other redundant patterns that do not add any value to the text.
6. Tokenization: Tokenization involves breaking down the text into individual words

or tokens. This is an important step in text summarization as it helps to create a representation of the text that can be easily processed by the summarization algorithm.

7. Padding: Padding is the process of adding zeros or other placeholders to the text to make it uniform in size. This is important when using deep learning models for text summarization, as they require fixed-length inputs.

Overall, these data preparation steps help to clean and prepare the text data in a way that is suitable for text summarization algorithms. I had created a custom function to implement all the above steps. Link for Data Preparation file in github: Link

6a) Key Strategies Used:

In the link provided, a **utils_preprocess_text** function is created. The function is a text pre-processing function that cleans the input text data. It performs various operations on the input text such as removing punctuations and characters, removing extra spaces, removing numeric characters, removing urls and email addresses, converting text to lowercase, and converting slang words to their original form.

Additionally, the function can perform tokenization, stemming, lemmatization, and removing stopwords based on the input parameters provided. Tokenization refers to converting text data into tokens or words. Stemming is the process of reducing words to their base or root form.

Lemmatization is similar to stemming but it converts words into their base or root form by taking into account the context of the sentence. Stopwords are words that are commonly used in a language but do not contribute much to the meaning of a sentence, such as "the", "and", "a", etc. And, based on EDA additional stop words are added.

Overall, the utils_preprocess_text function is an essential component of text preprocessing for this project, which cleans and prepares the raw text data for further processing and analysis.

TensorFlow and Keras libraries are used to tokenize and pad the cleaned text data. The Tokenizer class is used to tokenize the text data into sequences and limit the number of words to max_features which is set to 10,000 in this case. The pad_sequences function is then used to create sequences of the same length, maxlen is set to 500 here.

7. Modeling Algorithms and its Parameters

Note: During the data preparation phase, I encountered several challenges related to computational complexity. In an effort to address these issues, I tried various platforms to optimize the process. However, for modeling, the available resources were not sufficient. Therefore, I opted to use pre-trained networks and fine-tuned them to suit my requirements. I also attempted to build a seq2seq model from scratch, but encountered numerous challenges during fitting and training. For a more detailed comparison, please refer to the subsequent sections.

7a) Pretrained T5 Algorithm:

The T5 (Text-to-Text Transfer Transformer) algorithm is a powerful pre-trained language model developed by Google's AI research team. It is based on the Transformer architecture, which is a type of neural network architecture that was introduced in a 2017 paper by Vaswani et al.

T5 is unique because it is a "text-to-text" model, meaning it can perform a wide range of natural language processing tasks by inputting textual data and generating textual output. This includes tasks like summarization, translation, question answering, and more.

The model was pre-trained on a massive amount of text data, including books, articles, and web pages, using a combination of unsupervised and supervised learning techniques. This training data is used to learn the statistical patterns and relationships in the language, which allows the model to generate high-quality outputs for a wide range of tasks.

One of the major advantages of T5 is its ability to perform "zero-shot" learning, meaning it can perform tasks that it was not specifically trained for. For example, the model can translate from one language to another even if it was not specifically trained. But I used this vast model just for text summarization. The process of using T5 for text summarization typically involves the following steps:

1. Collect a large dataset of news articles and their corresponding summaries.
2. Preprocess the data by cleaning and tokenizing the text.
3. Fine-tune the T5 model on the preprocessed data using a supervised learning approach, where the input to the model is the article text and the output is the corresponding summary.

4. Evaluate the performance of the model on a separate test set using metrics such as ROUGE (Recall-Oriented Understudy for Gisting Evaluation).
5. Once the model is trained and evaluated, it can be used to generate summaries for new articles. T5 can also be used for other text generation tasks such as translation, question answering, and conversation generation, among others. But these steps are applied to only text summarization

7a1) Approach:

Algorithm is taken from the Hugging Face Transformers library, which is a popular open-source library used for natural language processing tasks. Specifically, it is used to initialize a pre-trained T5 Transformer model for sequence-to-sequence tasks.

7a2) Here is how API's work and used:

AutoTokenizer.from_pretrained('t5-base'): This line loads a pre-trained T5 tokenizer from the Hugging Face model hub. The 't5-base' model is a pre-trained T5 model with 11 billion parameters.

T5ForConditionalGeneration.from_pretrained('t5-base'): This line loads a pre-trained T5 model for sequence-to-sequence tasks. The 'T5ForConditionalGeneration' is a class that defines a T5 model architecture for text generation, which can be fine-tuned for various NLP tasks.

Similar to data preprocessing, custom functions are created for modeling, getting summaries of desired lengths and getting ROUGE scores. This github Link has the code in more detail.

Fig-9 and Fig-10 describes how to initialize the functions in the code where the article is taken form df dataframe which is processed, cleaned and well prepared. And, K is the desired length of summary that needs to be generated. Output is generated and highlighted in the original text.

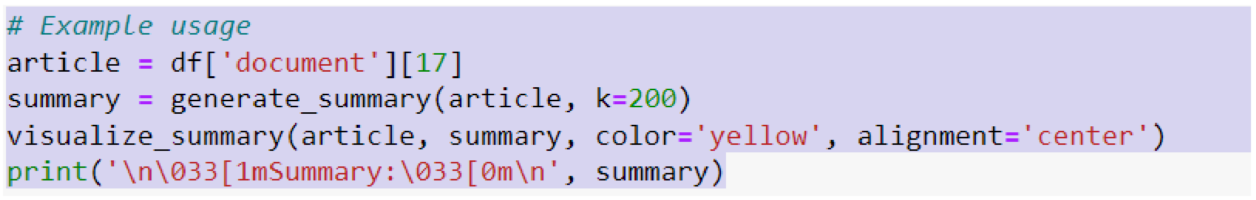
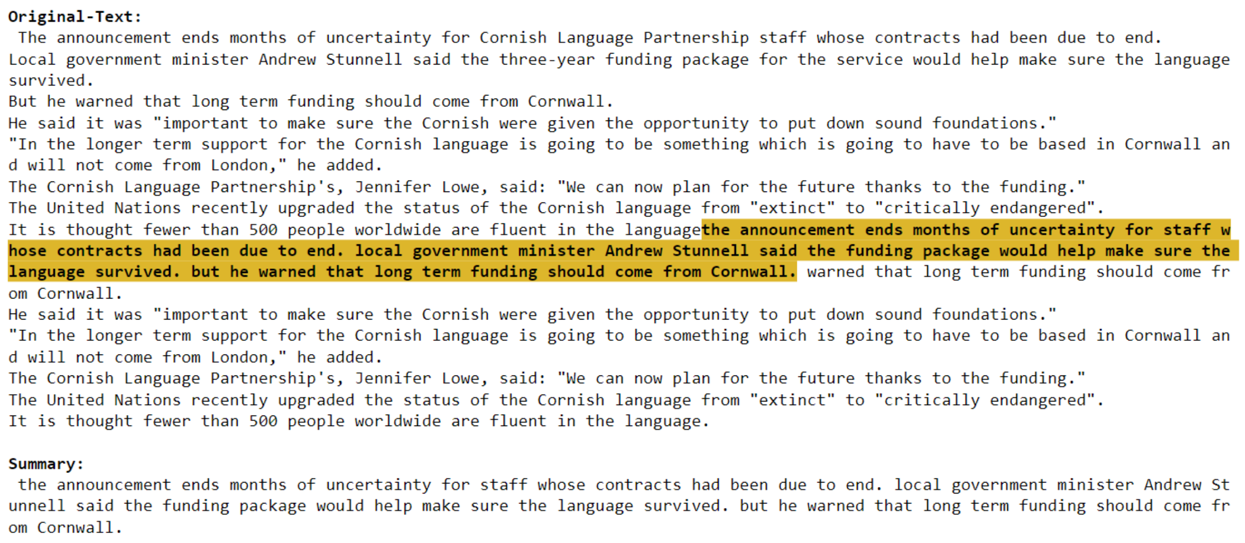

```
# Example usage
article = df['document'][17]
summary = generate_summary(article, k=200)
visualize_summary(article, summary, color='yellow', alignment='center')
print('\n\033[1mSummary:\033[0m\n', summary)
```

Fig-9: Code snippet on using T5 Algorithm



```
Original-Text:
 The announcement ends months of uncertainty for Cornish Language Partnership staff whose contracts had been due to end.
Local government minister Andrew Stunnell said the three-year funding package for the service would help make sure the language
survived.
But he warned that long term funding should come from Cornwall.
He said it was "important to make sure the Cornish were given the opportunity to put down sound foundations."
"In the longer term support for the Cornish language is going to be something which is going to have to be based in Cornwall an
d will not come from London," he added.
The Cornish Language Partnership's, Jennifer Lowe, said: "We can now plan for the future thanks to the funding."
The United Nations recently upgraded the status of the Cornish language from "extinct" to "critically endangered".
It is thought fewer than 500 people worldwide are fluent in the languagethe announcement ends months of uncertainty for staff w
hose contracts had been due to end. local government minister Andrew Stunnell said the funding package would help make sure the
language survived. but he warned that long term funding should come from Cornwall. warned that long term funding should come fr
om Cornwall.
He said it was "important to make sure the Cornish were given the opportunity to put down sound foundations."
"In the longer term support for the Cornish language is going to be something which is going to have to be based in Cornwall an
d will not come from London," he added.
The Cornish Language Partnership's, Jennifer Lowe, said: "We can now plan for the future thanks to the funding."
The United Nations recently upgraded the status of the Cornish language from "extinct" to "critically endangered".
It is thought fewer than 500 people worldwide are fluent in the language.

Summary:
 the announcement ends months of uncertainty for staff whose contracts had been due to end. local government minister Andrew St
unnell said the funding package would help make sure the language survived. but he warned that long term funding should come fr
om Cornwall.
```

Fig-10: Result of the code snippet in Fig-9

Scores for this algorithm are discussed in the results section.

7b) Seq2Seq Autoencoders:

A sequence-to-sequence (seq2seq) autoencoder is a neural network architecture that is commonly used for text summarization. The basic idea behind this model is to convert a longer sequence of text (e.g., an article) into a shorter summary sequence that captures the most important information from the original text.

The architecture of a seq2seq model consists of two main components: an encoder and a decoder. The encoder takes in the input sequence (e.g., an article) and produces a fixed-length representation of the sequence. This representation is then fed into the decoder, which generates the output sequence (e.g., a summary) one token at a time.

Here's an example of how a seq2seq autoencoder might work for text summarization:

1. First, we tokenize the input article and convert it into a sequence of word embeddings. After tokenizing the input article, the next step is to convert it into a sequence of word embeddings.
2. Word embeddings are a way to represent words in a vector space, where each word is represented by a dense vector of real numbers. The idea behind word embeddings is that words with similar meanings will have similar vectors in the vector space. So, by tokenizing the input article and converting it into a sequence of word embeddings, we are essentially

converting the raw text into a format that can be understood and processed by a machine learning model. This pre-processing step is crucial for text summarization, as it allows the model to capture the important information in the text and generate a concise summary.

3. We will use a sequence-to-sequence autoencoder model, which has an encoder and decoder network. The encoder network takes the input sequence and compresses it into a fixed-length vector representation, called the "context vector." The decoder network then takes this context vector and generates the output sequence, which is the summary in our case.
4. We pass the word embeddings through an encoder, which produces a fixed-length vector representation of the input sequence. The decoder takes the vector representation as input and generates the summary one token at a time.
5. We train the model on a dataset of articles and their corresponding summaries. During training, the encoder network learns to encode the input sequence into the context vector, and the decoder network learns to generate the output sequence from this context vector. The model is optimized to minimize the difference between the generated summary and the actual summary for each article.
6. At each step, it predicts the most likely next token based on the previous tokens it has generated. The decoder continues generating tokens until it reaches a predefined length (e.g., 50 words) or generates an end-of-sequence token.

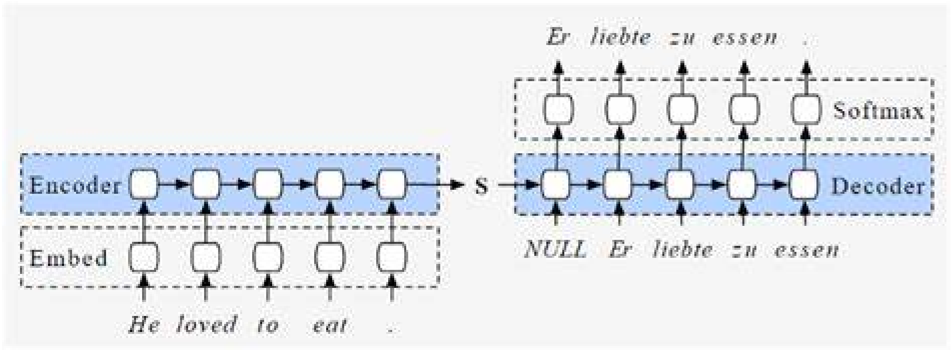


Fig-11: A encoder and decoder model for text translation, Source:Analytics Vidhya

In fig-11, it is used for text translation but the architecture is the same where the individual entities used in both encoder and decoder are Long-Short-Term-Memory-cells (LSTM) which is a Recurrent Neural Network. But the way of how data is processed and trained is mostly different as explained above. Link for seq2seq network in github is:

Link

7b1) Approach:

The code implementation of an encoder-decoder model with LSTM layers. The encoder part of the model consists of an embedding layer followed by an LSTM layer.

The embedding layer converts the input article into a sequence of word embeddings. The LSTM layer then takes the sequence of embeddings as input and outputs a sequence of hidden states.

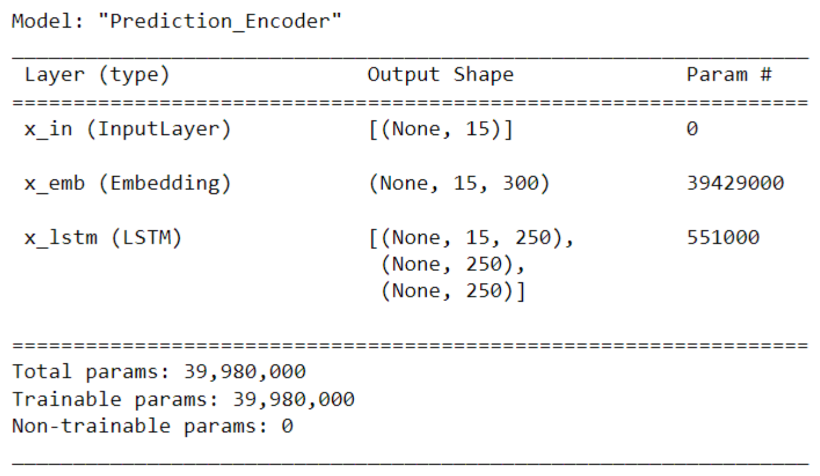
These hidden states capture the contextual information of the input sequence. The decoder part of the model is also composed of an embedding layer followed by an LSTM layer. The embedding layer converts the summary text into a sequence of word embeddings.

The LSTM layer takes these embeddings and outputs a sequence of hidden states. These hidden states are then fed into a dense layer which predicts the next token in the summary text.

The model has the following parameters:

- lstm_units: the number of LSTM units in the encoder and decoder layers.
- embeddings_size: the size of the word embeddings.
- dropout: the dropout rate used in the LSTM layers.
- return_sequences: a boolean value indicating whether the LSTM layer should return the full sequence of hidden states or just the last hidden state.
- return_state: a boolean value indicating whether the LSTM layer should return the last hidden state or both the last hidden state and the last cell state.

Overall, the encoder-decoder model is trained to minimize the difference between the generated summary and the actual summary. The model learns to encode the input article and decode it into a summary by capturing the contextual information of the input sequence.



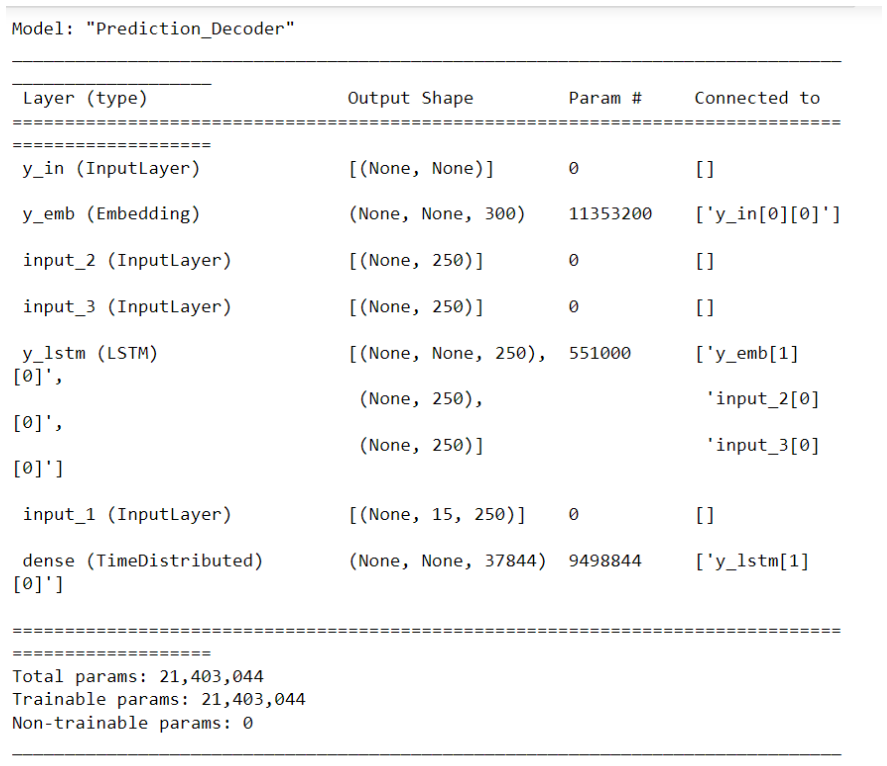
```
Model: "Prediction_Encoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 x_in (InputLayer)           [(None, 15)]              0

 x_emb (Embedding)           (None, 15, 300)           39429000

 x_lstm (LSTM)               [(None, 15, 250),         551000
                              (None, 250),
                              (None, 250)]

=================================================================
Total params: 39,980,000
Trainable params: 39,980,000
Non-trainable params: 0
_________________________________________________________________
```

Fig-12: Encoder Model Architecture



```
Model: "Prediction_Decoder"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 y_in (InputLayer)              [(None, None)]       0           []

 y_emb (Embedding)              (None, None, 300)    11353200    ['y_in[0][0]']

 input_2 (InputLayer)           [(None, 250)]        0           []

 input_3 (InputLayer)           [(None, 250)]        0           []

 y_lstm (LSTM)                  [(None, None, 250),  551000      ['y_emb[1]
[0]',
                                 (None, 250),                     'input_2[0]
[0]',
                                 (None, 250)]                     'input_3[0]
[0]']

 input_1 (InputLayer)           [(None, 15, 250)]    0           []

 dense (TimeDistributed)        (None, None, 37844)  9498844     ['y_lstm[1]
[0]']

==================================================================================================
Total params: 21,403,044
Trainable params: 21,403,044
Non-trainable params: 0
__________________________________________________________________________________________________
```

Fig-13: Decoder Model Architecture

Note: Due to the lack of computational resources, not all steps of the ML pipeline were utilized in this project. However, I was able to perform training and model evaluation. Moving forward, further progress can be made by extending the pipeline and incorporating additional steps.

Results of this model are given in the Results section.

8. Non-ML Algorithms

8a) Text Rank Algorithm:

TextRank is an unsupervised graph-based algorithm that uses the concept of PageRank from Google to rank the importance of sentences in a text document. TextRank is used for text summarization, keyword extraction, and document clustering.

The TextRank algorithm works by building a graph representation of the text document, where each sentence is a node in the graph. The edges between the nodes are weighted based on the similarity between the sentences. The weight of the edge between two nodes represents the similarity between the corresponding sentences.

The similarity between two sentences can be calculated using various methods, such as cosine similarity or Jaccard similarity. The cosine similarity is commonly used and is defined as:

$$cosine.\,similarity(s1, s2) = \frac{s1.s2}{|s1||s2|}$$

Where s1 and s2 are the vectors representing the two sentences, and ||s1|| and ||s2|| are their respective magnitudes.

Once the graph is constructed, the TextRank algorithm computes the PageRank score for each sentence, which represents its importance. The PageRank score of a sentence is calculated using the equation:

$$score(s) = (1-d) + d * \sum_{v \in V} \frac{w(v,s)}{\sum_{u \in Out(v)} w(v,u)} \cdot score(v)$$

Where s is the sentence for which the score is being calculated, v is a node that has an edge connecting it to s, w(v, s) is the weight of the edge between v and s, w(u, v) is the sum of the weights of all the edges originating from node u, and d is the damping factor, which is usually set to 0.85.

The algorithm iteratively computes the PageRank score for each sentence until convergence is achieved. The sentences with the highest PageRank scores are considered to be the most important and are selected for summarization or keyword extraction.

8b) Approach:

I had implemented a comprehensible custom text rank algorithm on my own. Follow this github link for further details: Link.

A class is created for the TextRank algorithm for automatic text summarization using Python and the Natural Language Toolkit (NLTK) library.

The TextRankSummarizer class has two main methods: fit and transform. The fit method takes a text document as input and processes it to build a graph representation of the document. The transform method takes the same document and a parameter n (the desired number of sentences in the summary), and returns a summary of the document consisting of the n most important sentences.

Here is a more detailed explanation of each method:

fit(text): This method performs the following steps:

- Tokenizes the text into individual words using a regular expression tokenizer. Removes stopwords (common words that do not carry much meaning) using the NLTK stopwords corpus. Lemmatizes each word (i.e., reduces it to its base form) using the WordNetLemmatizer from the NLTK library.
- Builds a vocabulary of unique words in the document.
- Creates an adjacency matrix for the graph, where each word is a node and edges are added between words that co-occur within a sentence.
- Normalizes the adjacency matrix by dividing each element by the sum of its row.
- Computes the PageRank scores iteratively until convergence, using a damping factor of 0.85. The resulting scores represent the importance of each word in the document.

transform(text, n): This method performs the following steps:

- Tokenizes the text into individual sentences using the NLTK sent_tokenize function.
- For each sentence, calculate a sentence score by taking the average of the PageRank scores of the words in the sentence.

- Selects the n sentences with the highest sentence scores, sorts them by their original order in the text, and concatenates them to form the summary.

9. Results

In this project, I had compared the performance of both machine learning and non-machine learning algorithms for generating summaries using the same set of performance metrics. Our goal was to evaluate the quality of the generated summaries, and I chose to use ROUGE scores as our performance metrics. ROUGE is a set of evaluation metrics widely used in NLP and text summarization research to measure the similarity between the generated summary and one or more reference summaries. By using a common set of performance metrics, we were able to effectively compare the performance of different summarization algorithms.

9.a) Rouge Scores

ROUGE stands for "Recall-Oriented Understudy for Gisting Evaluation''. It is a set of metrics used to evaluate the quality of summaries produced by automatic summarization systems. ROUGE measures the overlap between the output summary and the reference summaries, in terms of n-gram co-occurrences. The three most commonly used ROUGE measures are ROUGE-N, ROUGE-L, and ROUGE-W.

Here is an example of how to calculate ROUGE-N:

Let's assume we have a reference summary and a generated summary for a document.

The reference summary is: "The quick brown fox jumps over the lazy dog." The Generated summary is: "A brown fox jumped over a dog."

We want to calculate ROUGE-1, which measures the overlap of unigrams between the reference and generated summaries. First, we need to extract the unigrams from both summaries:

- **Reference summary unigrams:** {The, quick, brown, fox, jumps, over, the, lazy, dog}
- **Generated summary unigrams:** {A, brown, fox, jumped, over, a, dog}

Next, we calculate the number of overlapping unigrams between the reference and generated summaries. In this case, there are three overlapping unigrams: brown, fox, and over. Therefore, the ROUGE-1 score is 3/9 or 0.33.

ROUGE-2 and ROUGE-3 work similarly, except they measure the overlap of bigrams and trigrams, respectively. ROUGE-L measures the longest common subsequence (LCS) between the reference and generated summaries. The LCS is the longest sequence of words that appear in the same order in both summaries. ROUGE-L is useful when the generated summary contains additional words or phrases not present in the reference summary.

ROUGE-W is a weighted variant of ROUGE-L that takes into account the length of the LCS and the total number of words in the reference summary. In general, higher ROUGE scores indicate better summary quality, as they reflect a higher degree of overlap between the reference and generated summaries. However, the interpretation of ROUGE scores can vary depending on the specific use case and the evaluation methodology. I had used ROUGE-L scores to compare the performance of the algorithms.

9.b) Results Analysis:

ROUGE scores are used to evaluate the quality of summarization models, and they can be used to compare the performance of different summarization algorithms. In this case, I had compare the ROUGE scores of three models:

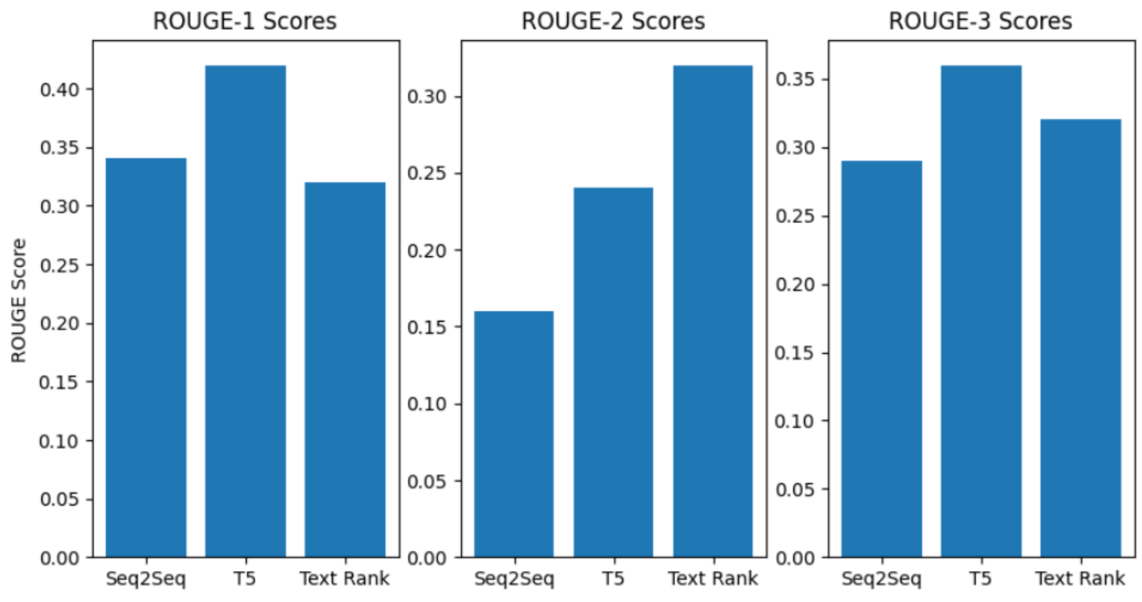


Fig-14: Final Modeling Results

Based on these scores, we can see that the **T5 model outperforms** the other two models across all ROUGE metrics, while the Seq2Seq model has the lowest ROUGE scores. This suggests that the T5 model may be the most effective at generating high-quality summaries. However, it's important to note that these results may vary depending on the specific dataset and summarization task being evaluated.

10: Conclusion and Difficulties Faced

In conclusion, the project "Text Summarization ML Vs Non-ML perspectives" aimed to compare the performance of ML and Non-ML based text summarization algorithms. The project involved building a custom seq2seq model, using the T5 pre-trained model for the ML part, and a custom-made Text Rank algorithm for the Non-ML part.

The evaluation of the models was done using the ROUGE score metric. The results indicated that the T5 model outperformed the other two models, with the highest ROUGE scores across all metrics. On the other hand, the Seq2Seq model had the lowest ROUGE scores, suggesting that it may not be as effective in generating high-quality summaries as the other models.

It's important to note that these results are based on the specific dataset and summarization task used in this project. The effectiveness of these models may vary for different datasets and tasks. Furthermore, other factors such as model complexity, training time, and resource requirements may also need to be considered when choosing an appropriate text summarization algorithm.

Overall, the project provides valuable insights into the performance of different text summarization algorithms and can serve as a reference for future research in this field.

During the course of this project, I faced several challenges that impacted our progress and performance. Some of the key difficulties encountered include:

- Lack of Computational Power: One of the major challenges I encountered was the lack of computational power required to run heavy algorithms. This made it difficult for us to test on vast hyperparameters and limited our ability to explore the best performing models.

- Building Complex Architectures from Scratch: Developing complex architectures from scratch required a deep understanding of the underlying algorithms and techniques, which proved to be a challenging task.
- Data Preprocessing Techniques: The quality of data and its preprocessing techniques have a direct impact on the performance of the models. It was difficult to identify and implement the most effective data preprocessing techniques.
- Dataset Size: The availability of a large and diverse dataset is crucial for the development and evaluation of text summarization models. However, acquiring and cleaning a large dataset was a time-consuming process.
- Reproducibility: Ensuring the reproducibility of the results was a difficult task as reproducing the experiments required significant time and resources.

In conclusion, these challenges highlighted the importance of having adequate resources, a deep understanding of the algorithms and techniques, and the need for effective data preprocessing techniques to develop high-performing text summarization models.

11: Future Work

In future work, I plan to address some of the limitations and challenges encountered during this project by:

1. **Incorporating Transfer Learning**: Transfer learning has shown promising results in natural language processing tasks, and I plan to explore its potential in text summarization. I plan to fine-tune pre-trained language models such as BERT and GPT-3 for text summarization tasks.
2. **Training Using GPUs and TPUs**: Training deep learning models on powerful hardware such as GPUs and TPUs can significantly reduce training time and enable us to explore a wider range of hyperparameters. I plan to train our models using these resources to improve their performance.
3. **Deploying the Model**: The ultimate goal of this project is to develop an end-to-end deployable app that can summarize text effectively. In future work, I plan to explore various deployment strategies, including web apps and mobile apps, to make our models accessible to a wider audience.

In conclusion, this project will be continued for further stages to address the challenges and limitations encountered, and to develop an effective and deployable text summarization app. Although the time and scope for this course project are limited,I believe that this project has laid the groundwork for future research in text summarization using ML and Non-ML approaches.

12: References

1. Nenkova, A., & McKeown, K. (2011). Automatic summarization. Foundations and Trends® in Information Retrieval, 5(2-3), 103-233. https://doi.org/10.1561/1500000015
2. Hovy, E., & Lin, C. Y. (1999). Automated text summarization in SUMMARIST. In Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics on Computational Linguistics-Volume 1 (pp. 468-475). Association for Computational Linguistics. https://www.aclweb.org/anthology/P99-1043/.
3. Rush, A. M., Chopra, S., & Weston, J. (2015). A neural attention model for abstractive sentence summarization. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (pp. 379-389).
4. "Text generation was performed using ChatGPT, a large language model developed by OpenAI based on the GPT-3.5 architecture (https://openai.com/)."
5. Mauro, Di Pietro. (2021, September 3). Text Summarization with NLP: TextRank vs Seq2Seq vs BART. Towards Data Science. https://towardsdatascience.com/text-summarization-with-nlp-textrank-vs-seq2seq-vs-bart-474943efeb09.
6. Amr, Z. (2021, February 15). Tutorial 3: What is Seq2Seq for Text Summarization and Why? Hacker Noon. https://medium.com/hackernoon/tutorial-3-what-is-seq2seq-for-text-summarization-and-why-68ebaa644db0.
7. Samhitha, A. (2019, July 18). Implement Seq2Seq for Text Summarization with Keras. Paperspace Blog. https://blog.paperspace.com/implement-seq2seq-for-text-summarization-keras/.
8. Pavan Yachamaneni, "Text Summarization ML Vs Non-ML Perspectives", GitHub repository, 2021. [Online]. Available: https://github.com/YP-17/Pavan-Yachamaneni-Text-Summarization-ML-Vs-Non-ML-Perspective.