Smart Summarizer

Navya Sree Sagi
Department of Computer Science
University of Central Florida
Oviedo, Florida, USA
navya.sagi@knights.ucf.edu

Priyanka Gopi
Department of Computer Science
University of Central Florida
Oviedo, Florida, USA
priyankagopi@knights.ucf.edu

ABSTRACT

Techniques for automatically summarizing text have drawn a lot of attention recently because of their capacity to draw out crucial information from massive amounts of text. With different working modes, the two primary families of approaches are extractive and abstractive. While abstractive techniques rework sentences by integrating information, extractive methods directly choose sentences from the source text. However, it is important to assess the quality of the generated summaries, for which ROUGE will be utilized. This study attempts to examine the better model in both the proposed summarizing methods. Overall, our work emphasizes the need for additional research to enhance the performance of text summarization models and to create more reliable measures for judging the caliber of generated summaries. For scholars and practitioners in the field of artificial text summarization, our work offers useful insights.

KEYWORDS

Text Rank, Automatic Text Summarization, Bart, Extractive, Abstractive, ROUGE, Bert

I. INTRODUCTION

In 1958, an IBM researcher named Hans Peter Luhn was credited with creating the first automatic text summarizing system. To identify the crucial sentences for summarizing, Luhn's method examined the document's usage of key terms and phrases. Subsequently, Edmundson built on Luhn's approach by classifying phrases according to their relevance using a score system. The work of these two scientists has significantly advanced the field of automatic text summarization and paved the ground for further development.

There are two methods to this: Extractive and Abstractive. Extractive approach focuses on small group of words with most crucial meaning and highlights the main sentence elements. To find the text's most informative sentences, extractive approaches typically use a variety of ranking algorithms, such as TextRank, Latent Semantic Analysis (LSA), and neural networks like Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). We used the Text Rank method and pre-trained language model used by BERT, a supervised deep learning model, to anticipate the most pertinent sentences. Text summarization is one of many natural language processing (NLP) tasks for which BERT has proven to be quite successful.

Second, the abstractive method uses effective Natural Language Processing (NLP) models, and provides summaries with relative content, by creating a new, condensed version of the data with important information. On the other side, abstractive summarization seeks to provide a summary by paraphrasing and rephrasing the input text while maintaining the most crucial details. The development of abstractive methods is more difficult and frequently depends on sophisticated approaches like Natural Language Generation (NLG) and Sequence-to-Sequence models, such the well-known encoder-decoder models BERT, GPT, and T5.

In the literature, several models for extractive and abstractive methods of automatic text summarization have been put forth. In research, extractive techniques like TextRank and LSA have been widely applied and are regarded as the industry standard for text summarizing jobs. We will employ Seq2Seq and BART models for the abstractive strategy. It has demonstrated success in producing abstractive summaries from the input material by rephrasing and paraphrasing it. While maintaining the context and coherence of the input text, BART is a pre-trained sequence-to-sequence model that can produce high-quality summaries.

By putting these models into practice, we will be able to assess each model's efficacy and compare how well extractive and abstractive technique’s function. ROUGE, a commonly used metric for assessing the quality of summaries, will be one of many metrics considered in the study. With this project, we hope to create a text summary system that can produce excellent summaries while retaining the key details from the source text.

In general, computerized text summary methods hold significant promise for promoting effective information intake. However, more investigation is necessary to create more reliable and efficient models that can provide high-quality summaries that accurately capture the crucial data from the input text.

2. PROBLEM STATEMENT

Information retrieval has become difficult due to the exponential growth of online data, requiring time-consuming and exhausting manual efforts. The difficulty of processing and extracting useful insights from enormous amounts of unstructured text data is becoming more and more difficult as a result of the exponential

growth of digital information. As a result, there is an increasing need for fast and effective text summary approaches that can create condensed versions of text documents automatically, allowing users to swiftly comprehend the most crucial information and make wise judgments. Existing methods for summarizing texts, however, have a number of problems, including maintaining the coherence and quality of the content summarized, dealing with a variety of texts, and dealing with the problem of information loss throughout the summation process. Therefore, there is an urgent need to create unique and reliable text summary techniques that may get around these obstacles and enhance the effectiveness and precision of summarizing. This era mandates the usage of text summarization as a useful technique to speed up comprehension of massive amounts of textual information. It is quite difficult for humans to manually summarize such information and finding meaningful data on the internet is challenging and gets worse when trying to extract the key points from the page.

Techniques for automatically summarizing text, such as abstractive and extractive approaches, have looked promising in resolving this issue. In particular, the sequence-to-sequence, BART, for abstractive and text ranking, and BERT models for extractive, will be assessed and compared for their efficiency in producing text summaries in this study.

3. RELATED WORK

Text summary has been the subject of numerous studies. Rush et al. (2015)'s "A Neural Attention Model for Abstractive Sentence Summarization" is one such study that put forth a neural attention model for summarizing phrases. To create a summary that extracts the most crucial details from the input text, they employed an encoder-decoder model with an attention mechanism. The model produced cutting-edge results after being trained on the CNN/Daily Mail dataset.

Several studies have been done to assess how well various summarizing strategy's function. In research by Paulus et al. (2018), they discovered that on the CNN/Daily Mail dataset, abstractive methods beat extractive methods in terms of ROUGE scores. Another study by Li et al. (2021) compared the performance of various extractive and abstractive models on the same dataset and discovered that the BART-based abstractive model outperformed the extractive models in terms of F1 scores, while the BERT-based extractive model outperformed the latter.

Additionally, Google has created the cutting-edge abstractive text summarization model PEGASUS.There are single and multi-document summarizations depending on the document count. Meanwhile, the extractive and abstractive outcomes are based on the summary results.

There are several commercial applications of text summarization outside of academia, such as Quillbot, which provides a variety of summary tools by combining extractive and abstractive summary methods. A customizable summary length, summarizing entire articles, and summarizing multiple articles are just a few of the choices for summarization that Quill Bot provides. Quillbot applications also have other text features like text paraphrasing and text generation.

There are also several daily used applications which follow this summarization technique. Google news uses text summarization for providing headlines to their news articles. Summly is a mobile application available which can help people with summarizing text articles in real time. It uses machine learning and natural language processing algorithms for proving accurate summary. Microsoft power BI, which is the most popular and powerful visualization tool also uses text summarization in a feature called quick insights. This feature will automatically summarize the text and provide data visualizations automatically. In this way summarization has become a part of the daily usage of an individual by making their work easier and more comfortable.

4. TECHNIQUE

In this project, we are implementing two types of summarizers. They are extractive and abstractive summarizers.

4.1 EXTRACTIVE SUMMARIZER

Extractive summarizer will summarize the text by choosing the sentences from the original text. It selects the important phrases from the original text article and will combine them to form the summary. Mostly the summaries formed from this type are not meaningful and grammatically irrelevant as it chooses only important phrases.

4.2 ABSTRACTIVE SUMMARIZER

Abstractive summarizer generates the summary by selecting the important phrases from the original text but constructs its own sentences. The summary generated is on its own without changing the original meaning of the original article. It just acts like a human taking important notes. The summary generated is grammatically correct and in an understandable manner.

4.3 IMPLEMENTATION

The first step was to obtain the required data from Kaggle's CNN daily news data. The CNN Daily News dataset is a collection of news stories that were scraped from the CNN website. The collection includes over 120,000 items that were gathered over the course of 15 days in July 2017 on a variety of subjects, such as politics, entertainment, sports, and technology. The dataset was already split and provided to us, wherein the train and test data consisted of 287113 and 11490 articles with summaries, respectively.

To implement a summarization model effectively and to help extract the text's semantics, data pre-processing is a crucial step. We began with dropping the duplicate values. To make sure that the remaining data is clean and contains only unique samples, duplicate values at the beginning of the dataset are often removed. Prior to dividing the dataset into training, validation, and testing sets, this is typically done as a preprocessing step. This makes sure that each example is only used once during training, validation, or

testing and prevents the model from becoming biased against repeated cases.

```
# drop duplicate articles to reduce redundancy

train.drop_duplicates(subset=['article'],inplace=True)
test.drop_duplicates(subset=['article'],inplace=True)
```

Figure 1: Drop duplicate articles

Next, the articles texts were divided into sentences, all words are lowercased, stop words and extraneous punctuation are removed, and other unimportant jobs are completed. To maintain data consistency throughout the analysis, the text was also changed to lowercase.

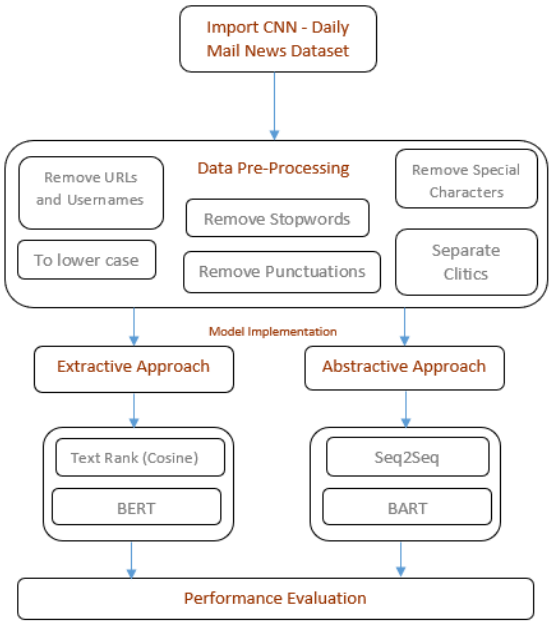


Figure 2: Project Methodology

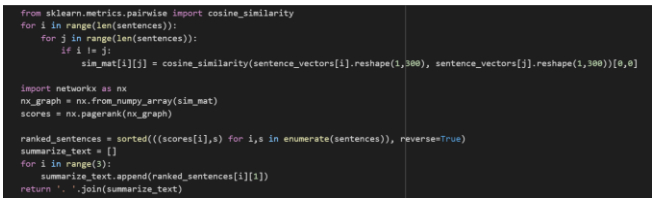
```
def preprocess_text(text,n,punkt=True):
    text = text.lower()
    text = BeautifulSoup(text, "lxml").text
    text = re.sub(r'\([^)]*\)', '', text)
    text = re.sub('"','', text)
    text = " ".join([word.strip() for word in text.split()])
    text = contractions.fix(text)
    text = re.sub(r'[^\w\s]', '', text) if punkt is True else text
    text = re.sub(r"'s\b","",text)
    text = re.sub("[^a-zA-Z]", " ", text)
    if(n==0):
        tokens = [w for w in text.split() if not w in stop_words]
    else:
        tokens=text.split()
    long_words=[]
    for i in tokens:
        if len(i)>1:
            long_words.append(i)
    return (" ".join(long_words)).strip()
```

Figure 3: Text preprocessing

The material must be accurately identified and divided into sentences for the algorithm to choose the appropriate sentences for the summary. Additionally, it's crucial to exclude stop words, all punctuation, digits, and special characters from the text, as well as usernames and URLs from the news piece. The quality of the summary can also be improved by dividing clitics in the words and changing everything to lower case.

4.3.1 TEXT RANK MODEL

Next, an extractive and unsupervised text summarizing method TextRank was implemented which uses a cosine similarity to score sentences. The input data is separated into sentences and for each sentence, a score was generated using the words tokens. The vector representation (word embeddings) for each sentence is then found, which is further used to find the cosine similarities between sentence vectors and store them in a matrix. This matrix is then turned into a graph with similarity scores acting as edges and texts serving as vertex points. Based on the significance of the sentences it is connected to in the graph, the TextRank search algorithm assigns a score to each sentence. The top-rated sentences are chosen as the summary after each sentence is scored according to its score. The final summary is made up of the top three sentences with the greatest rank. In this baseline model, the number of terms with the highest frequency that are present in a sentence is used to determine the sentence score.



```
from sklearn.metrics.pairwise import cosine_similarity
for i in range(len(sentences)):
    for j in range(len(sentences)):
        if i != j:
            sim_mat[i][j] = cosine_similarity(sentence_vectors[i].reshape(1,300), sentence_vectors[j].reshape(1,300))[0,0]

import networkx as nx
nx_graph = nx.from_numpy_array(sim_mat)
scores = nx.pagerank(nx_graph)

ranked_sentences = sorted(((scores[i],s) for i,s in enumerate(sentences)), reverse=True)
summarize_text = []
for i in range(3):
    summarize_text.append(ranked_sentences[i][1])
return ". ".join(summarize_text)
```

Figure 4: Generate similarity matrix and rank sentences

As text rank model is an unsupervised model, it does not require training data which is an advantage when compared to other models or algorithms when it takes a lot of time for training of the model. In this model, the original text is directly given as input, then its preprocessed and finally the text rank algorithm is applied on it to generate the summary. This algorithm is also language independent hence it can be used for multilingual purposes also. The summary generated is always relevant and coherent as the algorithm generated summary by constructing a graph when it captures the relationship between the sentences in the text. It is easy to implement, and it always retains the original information as it chooses summary from the original text only.

4.3.2 BERT MODEL

BERT (Bidirectional Encoder Representations from Transformers) model is used for extractive approach which works more accurately for text summarization. The transformer-based BERT model was also executed on the CNN-daily dataset, where the 'bert-extractive-summarizer' was installed, and its library 'summarizer' was imported to generate the summaries. In this model, the original article is given directly as an input. The input text is then

preprocessed and then fed into the Bert summarizer where the important phrases are picked to generate summary.

As this is bidirectional summarizer, it captures the dependencies between the words and contexts in both directions. Hence it understands better the meaning and relationship between the sentences. This model can perform well on long articles and multiple documents. Having access to more data, BERT can certainly make better decisions on the sentences to include in the summary. It's been fine-tuned to this specific task of text summarization, which is why it is expected to perform much better than the generic algorithm like TextRank.

```
bert_model = Summarizer()
def generate_summary(article):
  bert_summary = ''.join(bert_model(article, min_length=100))
  return bert_summary
```

Figure 5: BERT summarizer

4.3.3 SEQUENCE TO SEQUENCE MODEL

Sequence-to-sequence (Seq2Seq) may process sequences of varying lengths and produce output sequences of any length. An encoder and a decoding network make up their two major parts.

The context vector is the result of the encoder network's conversion of a variable-length input sequence, like a sentence or document, into a fixed-length vector representation. The encoder, which commonly consists of a recurrent neural network (RNN) or a transformer network, analyses the input sequence token-by-token and produces a series of hidden states that represent the context of the input sequence. Using the context vector created by the encoder, the decoder network creates an output sequence, such as a summary or translation. The decoder, which also frequently uses an RNN or a transformer network, produces the output sequence one token at a time while taking the context vector as input and the context from the tokens that have already been produced. The input to the decoder consists of articles with sentences having start and end tokens.

```
# for decoder input to identify the start and end tokens of sentences

df_train['short_summary'] = df_train['short_summary'].apply(lambda x : 'sostok '+ x + ' eostok')
df_test['short_summary'] = df_test['short_summary'].apply(lambda x : 'sostok '+ x + ' eostok')
```

Figure 6: Addition of start, 'sostok' and end tokens, 'eostok' to articles for the decoder for sentence identification

We implemented code using keras model for seq2seq learning with attention mechanism. The model here takes two inputs. The input for the encoder has the shape (max_len_text), which represents a sequence of numbers that correspond to words in the source language, while the input for the decoder has the shape (None), which corresponds to words in the target language. The encoder input is passed through an embedding layer. The embedding layer converts each word in the encoder input into a high-dimensional vector representation before moving on to the next step. To obtain the final encoder output and hidden states, the embedded input is then processed through two LSTM layers with dropout and recurrent dropout. To produce a high-dimensional vector representation of each word in the target language, the decoder input is also processed through an embedding layer. Using the final encoder hidden states as the beginning state, the decoder embedded input is then processed through an LSTM layer with dropout and recurrent dropout. Next, sequence alignment in Seq2Seq models is accomplished via the Bahdanau attention mechanism, which is implemented by the AttentionLayer third-party class. The attention weights are computed for each step in the decoder sequence using the encoder output sequence and the decoder output sequence as its two inputs. Following that, context vectors—weighted sums of the encoder output sequence—are computed using these attention weights. Context vectors and attention weights are returned by the layer as outputs.

The encoder output and decoder output are combined in an attention layer to create a context vector, which is then concatenated with the decoder output at each time step. To determine the output probabilities for each word in the target language at each time step, the concatenated output is then processed through a dense layer with softmax activation.

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 300)]	0	[]
embedding (Embedding)	(None, 300, 100)	6759300	['input_1[0][0]']
input_2 (InputLayer)	[(None, None)]	0	[]
lstm (LSTM)	[(None, 300, 300), (None, 300), (None, 300)]	481200	['embedding[0][0]']
embedding_1 (Embedding)	(None, None, 100)	1686000	['input_2[0][0]']
lstm_1 (LSTM)	[(None, 300, 300), (None, 300), (None, 300)]	721200	['lstm[0][0]']
lstm_2 (LSTM)	[(None, None, 300), (None, 300), (None, 300)]	481200	['embedding_1[0][0]', 'lstm_1[0][1]', 'lstm_1[0][2]']
attention_layer (AttentionLayer)	((None, None, 300), (None, None, 300))	180300	['lstm_1[0][0]', 'lstm_2[0][0]']
concatenate (Concatenate)	(None, None, 600)	0	['lstm_2[0][0]', 'attention_layer[0][0]']
time_distributed (TimeDistributed)	(None, None, 16860)	10132860	['concatenate[0][0]']

Figure 7: Layers of Seq2Seq Abstractive Model

The model is trained to reduce the cross-entropy loss between the genuine target language terms and the predicted output probabilities. The model is trained for 20 epochs and then predicted with testing data.

4.3.4 BART MODEL

In 2019, Facebook AI Research unveiled the pre-trained transformer-based neural network model BART (Bidirectional and Auto-Regressive Transformer). It was made to handle a range of NLP tasks, including text summarization. Because it combines auto regressive and denoising objectives in pre-training, BART is unique in this regard. The model has thus been trained to produce text as well as to reconstruct erratic input sequences. By doing this, BART can be improved on several NLP tasks, such as text summarization, with only modest additional training.

BART can be fine-tuned for text summarizing on a dataset of document-summary pairs, where the model is trained to accept a document as input and produce a summary as an output. On various benchmark text summarization datasets, such as the CNN/Daily Mail datasets, BART has demonstrated state-of-the-art results.

The pre-trained tokenizer is loaded by the BartTokenizer.from_pretrained() method, and it oversees turning unprocessed text into tokens that the BART model can use. The term "facebook/bart-large-cnn" refers to a particular BART model that has been pre-trained on a sizable corpus of text data. A modified version of the original BART model that has been optimized for text summarization tasks is the pre-trained BART model, which is eventually loaded using the pre-defined function BartForConditionalGeneration.from_pretrained() function. Given a text or sequence of tokens as input, this model can produce summary text.

```
# Load pre-trained BART tokenizer
tokenizer = BartTokenizer.from_pretrained('facebook/bart-large-cnn')
bart_model = BartForConditionalGeneration.from_pretrained('facebook/bart-large-cnn')
```

Figure 8: Import BART Tokenizer

The input text is preprocessed and tokenized using the pre-defined Bart tokenizer. The tokenized words are given as input to the Bart model and summaries are generated as output. The model is tested for accuracy by comparing the generated summaries by the model and the original summaries.

```
def generate_summary(article):
    input_ids = tokenizer.encode(article, return_tensors='pt')
    output = bart_model.generate(
        input_ids,
        max_length=100,
        num_beams=4,
        no_repeat_ngram_size=3,
        early_stopping=True
    )
    bart_summary = tokenizer.decode(output[0], skip_special_tokens=True)
    return bart_summary
```

Figure 9: Build BART Summarizer function

5. EVALUATION

To evaluate our models, we are using ROUGE (Recall-Oriented Understudy for Gisting Evaluation) as the metric. As we are dealing with the language models, its difficult to evaluate the accuracy, but with the ROUGE metric its easy to understand and implement. We are using ROUGE 1, ROUGE 2, ROUGE L to measure the accuracy of the summarizers. The system-generated summary and the reference summary's unigrams (single words) overlap according to the ROUGE-1 measure. ROUGE-1 is a straightforward statistic that assesses the degree of word overlap between the reference summary and the system-generated summary. The system-generated summary and the reference summary's bigrams (pairs of consecutive words) coincide according to the ROUGE-2 measure. ROUGE-2 is more limited than ROUGE-1 since it only considers situations in which two consecutive words exist in both the reference summary and the system-generated summary. The longest common subsequence (LCS) between the reference summary and the system-generated summary is calculated using the ROUGE-L metric. The LCS is made up of the words that appear the most frequently in both summaries in the same order. Compared to ROUGE-1 and ROUGE-2, ROUGE-L is more forgiving because it permits non-consecutive matches between the reference summary and the system-generated summary.

5.1 EXPERIMENT OUTPUT OF EACH MODEL

The F1 scores, precision and recall scores of all models are obtained as follows –

```
ROUGE-1: {'r': 0.21153846153846154, 'p': 0.15492957746478872, 'f': 0.1788617837371936}
ROUGE-2: {'r': 0.07462686567164178, 'p': 0.056818181818181816, 'f': 0.06451612412403783}
ROUGE-L: {'r': 0.19230769230769232, 'p': 0.14084507042253522, 'f': 0.16260162113556761}
```

Figure 10: Rouge values of TextRank model

```
ROUGE-1: {'r': 0.6153846153846154, 'p': 0.22377622377622378, 'f': 0.3282051242940171 6}
ROUGE-2: {'r': 0.4328358208955223 6, 'p': 0.1518324607329843, 'f': 0.2248061977053663}
ROUGE-L: {'r': 0.5961538461538461, 'p': 0.21678321678321677, 'f': 0.3179487140376069}
```

Figure 11: Rouge values of BERT model

```
ROUGE-1: Precision = 0.30277777777777737 Recall = 0.0868991506040644 9 F1-score = 0.13241783711910146
ROUGE-2: Precision = 0.0343809523809524 Recall = 0.01216131041031263 F1-score = 0.017493984562127834
ROUGE-L: Precision = 0.2806666666666663 Recall = 0.08130867961910827 F1-score = 0.12359396349067281
```

Figure 12: Rouge values of Seq2Seq model

```
Article 1 ROUGE Scores:
[{'rouge-1': {'r': 0.36363636363636365, 'p': 0.25, 'f': 0.2962962914677641}, 'rouge-2': {'

Article 2 ROUGE Scores:
[{'rouge-1': {'r': 0.5588235294117647, 'p': 0.4418604651162791, 'f': 0.4935064885748019},

Article 3 ROUGE Scores:
[{'rouge-1': {'r': 0.42857142857142855, 'p': 0.32432432432432434, 'f': 0.3692307643266272
```

Figure 13: Rouge values of BART model

In the case of extractive models, text rank model is considered as a baseline model and BERT model is compared with the baseline model. Whereas in the case of abstractive model, seq2seq model is considered as the baseline model and working of the BART model is compared with the baseline model.

The extractive models generally do not require any training data hence the original article is given the input directly and the output summary is generated. For each article, an individual summary is generated and is validated with the original summary. Rouge values are generated for each predicted summary. To calculate the final accuracy of the model, we have calculated the average of all the rouge values of all the articles and summaries generated. In case of abstractive summarizer, the models are trained with huge amounts of training data and then validated against the test data. The predicted summaries are compared to the original summaries of the test data to check the accuracy of the model. Hence the rouge values obtained in this case are the accuracies of the complete model validated against the test dataset.

TYPE	MODEL	ROUGE -L		
		Recall	Precision	F1
EXTRACTIVE SUMMARIZER	TextRank	0.19	0.14	0.16
	BERT	0.59	0.22	0.32
ABSTRACTIVE SUMMARIZER	Seq2Seq	0.08	0.28	0.12
	BART	0.53	0.42	0.47

Table 1: Table showing comparison of ROUGE – L metrics of different models.

The above table shows the values of recall, precision, and recall of ROUGE – L for different models in extractive and abstractive approaches. In the case of Extractive, with recall of 0.59 versus TextRank's 0.19, precision of 0.22 versus TextRank's 0.14, and F1 of 0.32 versus TextRank's 0.16, BERT outperforms TextRank among the extractive summarization models across all three ROUGE-L criteria. This shows that BERT can provide summaries that are of higher quality than TextRank and extract more pertinent information from the input text.

BART exceeds Seq2Seq among the abstractive summarization models on all three ROUGE-L measures, with recall of 0.53 versus Seq2Seq's 0.08, precision of 0.42 versus Seq2Seq's 0.28, and F1 of 0.47 versus Seq2Seq's 0.12. This suggests that BART is more adept than Seq2Seq at creating accurate and instructive summaries.

6. DISCUSSION

The dataset used for our study on abstractive and extractive approaches to summarization, was the CNN Daily Mail News dataset from Kaggle. As discussed before, two models such as Text Ranking and BERT were implemented, and we saw that the BERT outperformed the baseline model in the case of extractive method. For abstractive approach, BART model outperformed seq2seq baseline model. Because of computational resource constraints, we were unable to run the seq2seq model for more than 20 epochs during model training. Due to this restriction, the model may not have fully converged, which may have had an impact on its accuracy and loss. When a model is trained over a longer period, it can pick up more intricate patterns and possibly perform better. We were, however, unable to further investigate this possibility due to the limitations of computation resources. To perhaps increase the model's accuracy and loss, it would be advantageous to think about training it over a greater number of epochs in the future. To further enhance the model's performance within the constraints of the available computational resources, further methods like transfer learning could be investigated.

In this comparison of extractive and abstractive summarization methods, BERT and BART perform better than their respective alternatives overall. Before choosing a model for a specific use case, it's crucial to keep in mind that different summarization models may perform differently depending on the precise task and dataset they are evaluated on. As a result, it's crucial to take multiple evaluation metrics into account and carry out extensive experimentation.

7. CONCLUSION

In conclusion, Text Summarization is a key area in natural language processing that helps in reducing the manual effort to summarize the text. Extractive and abstractive approaches have been used for summarizing text, each having its own benefits and limitations. In this project, we have implemented both approaches and compared their performance using the CNN Daily Mail dataset. We tested four alternative summary methods in this study to see how well they summarized text. Two extractive models, TextRank and BERT, and two abstractive models, Seq2Seq and BART, were evaluated using ROUGE as the main metric. Our results demonstrated that the Bert extractive summarizer is efficient in selecting the required sentences. On the other hand, pre-trained models such as BART have shown better performance in abstractive summarization.

We determined the advantages and disadvantages of each model based on the findings of our evaluation as well as additional elements like model complexity and interpretability. For instance, BERT is a more potent extractive model than TextRank, but it needs more computing power and is more difficult to understand. Like Seq2Seq, BART is a more powerful abstractive model, but it is also more complex and may be more challenging to train and improve.

The results of our study have important implications for real-world uses of text summarization. The need for automated summarizing tools is rising across a range of industries, including news and media, scholarly publications, and legal papers. Our research sheds light on the benefits and drawbacks of several summarization models, enabling practitioners and academics to choose the most effective model for a particular task and dataset.

In conclusion, our study emphasizes the value of further study in text summarizing. The demand for efficient automatic summarization tools will only grow as the volume of digital data continues to rise. Future studies could look into topics like creating hybrid models that mix extractive and abstractive methods, enhancing the readability and explicability of models, and experimenting with various pre-training strategies.

ACKNOWLEDGMENTS

We would like to express our gratitude to Dr. Fereshteh Lux from the University of Central Florida for giving us the opportunity to work on this real-time project as part of the CAP6640 course. Additionally, we extend our thanks to Kaggle for providing us with the dataset necessary to conduct this study. The Extractive models were implemented by Navya Sree Sagi, and the Abstractive models were implemented by Priyanka Gopi. Both team members contributed equally to the project report.

REFERENCES

[1] Liu, Yang (2019). “Fine-tune BERT for extractive summarization”. In: arXiv preprint arXiv:1903.10318.

[2] Gupta, Som and SK Gupta (2019). “Abstractive summarization: An overview of the state of the art”. In: Expert Systems with Applications 121, pp. 49–65

[3] CNN Daily Mail News Text Summarization Kaggle Dataset (n.d.). https://www.kaggle.com/datasets/gowrishankarp/newspaper-text-summarization-cnn-dailymail.

[4] Shubham Kumar Shukla. 2022. Text Summarization in NLP. https://www.topcoder.com/thrive/articles/text-summarization-in-nlp

[5] https://medium.com/luisfredgs/automatic-text-summarization-with-machine-learning-an-overview-68ded5717a25

[6] https://medium.com/nlplanet/two-minutes-nlp-learn-the-rouge-metric-by-examples-f179cc285499

[7] https://towardsdatascience.com/the-ultimate-performance-metric-in-nlp-111df6c64460

[8] https://www.projectpro.io/article/transformers-bart-model-explained/553

[9] Gupta, Vishal and Gurpreet Singh Lehal (2010). “A survey of text summarization extractive techniques”. In: Journal of emerging technologies in web intelligence 2.3, pp. 258–268.

[10] Akira Takezawa. 2019. How to implement Seq2Seq LSTM Model in Keras. https://towardsdatascience.com/how-to-implement-seq2seq-lstm-model-in-keras-shortcutnlp-6f355f3e5639

[11] https://medium.com/data-science-in-your-pocket/text-summarization-using-textrank-in-nlp-4bce52c5b390