République Tunisienne
Ministère de l'Enseignement Supérieur,
de la Recherche Scientifique et de la Technologie
--
Université de Carthage

Faculté des Sciences
De Bizerte

Département
Informatique

Report about

Article Summarizer with NLP

Firas Rebai

Table Of Contents

1 Introduction

1.1 Importance of Summarization in NLP

In the era of information overload, the ability to distil vast amounts of textual information into concise and coherent summaries has become paramount. As the volume of digital content continues to escalate, traditional methods of manually curating and summarizing articles are proving insufficient. The demand for automated solutions to extract the essential meaning from large bodies of text has led to the emergence of various techniques, with deep learning standing out as a powerful and promising approach.

This report delves into the realm of article summarization, with a specific focus on leveraging deep learning methodologies to address the complexities of information condensation. The exponential growth of digital information sources, such as news articles, research papers, and online publications, necessitates efficient methods for quickly extracting the core insights. Deep learning, with its ability to automatically learn hierarchical representations from data, offers a compelling solution to the challenges posed by the dynamic and diverse nature of textual content.

The motivation behind this project lies in the recognition of the limitations of traditional summarization techniques and the desire to harness the potential of deep learning to enhance the precision and adaptability of automated summarization. By training models on large datasets and allowing them to learn intricate patterns and relationships within the text, we aim to develop a system capable of generating coherent and contextually relevant summaries, contributing to the evolution of information processing in the digital age.

In the following sections, we will explore the existing landscape of article summarization, articulate the specific challenges addressed in this project, outline the objectives, and detail the methodology employed. Through a comprehensive examination of deep learning techniques, we endeavour to demonstrate the effectiveness of our approach in automating the summarization process and extracting meaningful insights from diverse textual sources.

2 Text Summarization in NLP

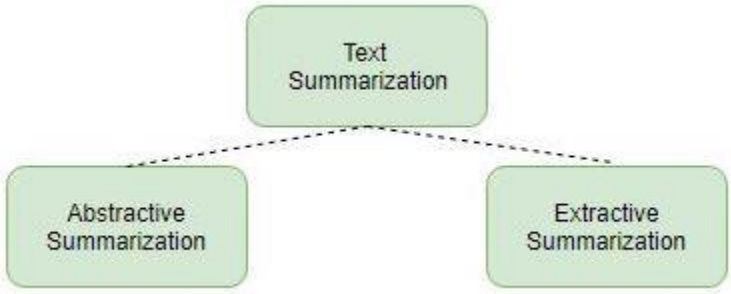
“Automatic text summarization is the task of producing a concise and fluent summary while preserving key information content and overall meaning”.

-Text Summarization Techniques: A Brief Survey, 2017

Text summarization refers to the technique of condensing a lengthy text document into a succinct and well-written summary that captures the essential information and main ideas of the original text, achieved by highlighting the significant points of the document.

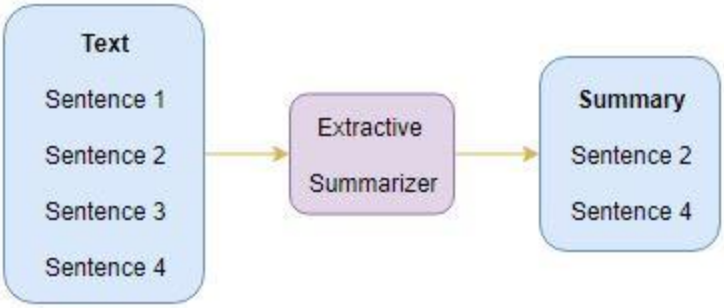
There are broadly two different approaches that are used for text summarization:

- Extractive Summarization
- Abstractive Summarization



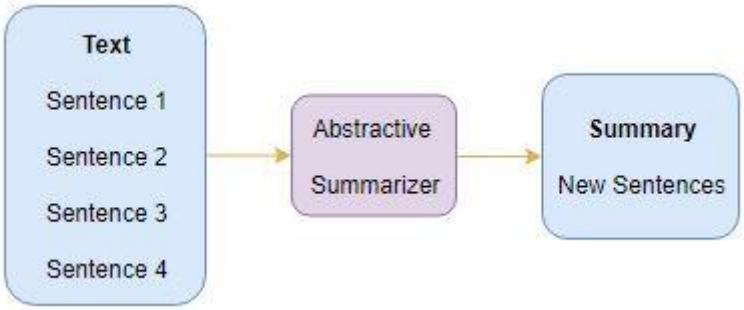
- **Extractive Summarization**

These methods rely on extracting several parts, such as phrases and sentences, from a piece of text and stack them together to create a summary. Therefore, identifying the right sentences for summarization is of utmost importance in an extractive method. The below diagram illustrates extractive summarization:



- **Abstractive Summarization**

This Method involves understanding the main ideas of the text and generating a new, summarized version based on that understanding. Abstractive summarization has the advantage of being able to generate new text, but it is more complex to implement and requires language generation capabilities. The sentences generated through abstractive summarization might not be present in the original text:



3 Objectives and Problems

The primary challenge faced in this project arises from the extensive length of the dataset texts, which poses challenges for training deep learning models, especially transformers. To overcome this hurdle, our objectives encompass mitigating length constraints, exploring the efficacy of an extractive summarization method, and creating a new dataset optimized for training transformer models.

3.1 Objective 1: Mitigating Length Constraints

The first objective is to address the computational challenges associated with lengthy texts. By implementing preprocessing techniques and model adaptations, we aim to efficiently process long-form textual data, ensuring that the training process remains both feasible and effective for deep learning models, particularly transformers.

3.2 Objective 2: Assessing Extractive Summarization Efficacy

Recognizing the challenge posed by lengthy texts, we explore the potential of extractive summarization as an alternative approach. Our goal is to create concise summaries by identifying and extracting salient sentences or passages from the original text. Through this, we aim to evaluate the effectiveness of the extractive method in producing coherent and informative summaries, particularly in the context of computational constraints associated with longer texts.

3.3 Objective 3: Creating a New Dataset for Transformer Training

To tailor the dataset to the demands of transformer models, we intend to generate a new dataset through extractive summarization. This involves systematically creating shorter summaries that encapsulate the essential information from the original texts. The new dataset will be optimized for training transformers, striking a balance between computational efficiency and the preservation of key content.

3.4 Objective 4: Training the Transformer Model

Building on the newly created dataset, our objective is to train a transformer model for article summarization. This includes fine-tuning the model, optimizing hyperparameters, and ensuring its proficiency in summarizing articles effectively.

3.5 Objective 5: Deploying the Trained Transformer Model

Upon successful training, our focus shifts to deploying the trained transformer model into a production environment. This involves optimizing the model for real-time inference, addressing scalability concerns, and ensuring seamless integration into the summarization workflow for practical use.

3.6 Objective 6: Creating a User Interface Connected to the API

The final objective is to develop a user-friendly web interface that allows users to interact with the summarization model. This interface will be connected to the deployed API, enabling users to input articles and receive concise summaries in an accessible and intuitive manner.

4 Methodology

To achieve our objectives, we will be going through these steps:

1. **Data Collection**
    - Identify and gather a diverse dataset of lengthy articles from relevant domains.
    - Consider sources such as news articles, research papers, or online publications to ensure a representative dataset.
2. **Data Preprocessing**
    - Perform text cleaning to remove noise, irrelevant characters, and formatting issues.
    - Tokenize the text into manageable units, considering the challenges posed by lengthy content.
    - Explore techniques for handling out-of-vocabulary words and rare terms.
3. **Extractive Summarization**
    - Implement an extractive summarization method to create shorter summaries from the original articles.
    - Evaluate and fine-tune the summarization method to ensure it captures the essence of the text effectively.
    - Assess the quality of the generated summaries using appropriate metrics.
4. **Dataset Creation**
    - Utilize the extractive summarization method to create a new dataset of shorter texts for training the transformer model.
    - Maintain a balance between summary length and information preservation to facilitate effective training.
5. **Transformer Model Architecture**
    - Choose a transformer architecture suitable for the article summarization task.
    - Fine-tune the transformer's hyperparameters, considering the specifics of the newly created dataset.
6. **Training Process**
    - Split the dataset into training and validation sets.
    - Train the transformer model on the shortened dataset, optimizing for efficient processing of lengthy texts.
    - Monitor training metrics, such as loss and accuracy, to ensure model convergence.
7. **Model Deployment**
    - Save the trained transformer model for future use.
    - Explore deployment options, considering scalability and real-time inference requirements.
    - Optimize the model for deployment in a production environment.
8. **Web Interface Development**
    - Design and develop a user-friendly web interface for interacting with the summarization model.
    - Connect the web interface to the deployed model via an API for seamless integration.

9. Evaluation

- Assess the performance of the trained transformer model using relevant evaluation metrics.
- Conduct user testing on the web interface to gather feedback on usability and overall satisfaction.

10. Iterative Optimization

- Iterate on the model and interface based on evaluation feedback.
- Fine-tune parameters, update the model, and enhance the interface for improved performance and user experience.

5 Implementation

5.1 Data Collection

For this project, the primary dataset utilized is the ScisummNet Corpus Summarization Dataset, which consists of a collection of lengthy articles paired with human-generated summaries. The dataset was chosen for its relevance to the project's objectives and the availability of ground truth summaries.

Despite exhaustive searches, the availability of comprehensive datasets with both text and summaries meeting our specific criteria proved limited. The ScisummNet dataset remains a valuable resource, providing a foundation for training and evaluating our summarization model.

Acknowledging the limitations in dataset diversity, efforts were made to extract a representative subset from the ScisummNet dataset to ensure adequate coverage of various topics and writing styles.

It's important to note that the ScisummNet dataset predominantly covers articles from the field of Computer Science. While this aligns with the project's focus Computer Science, the restricted domain may pose challenges in generalizing the model to a broader range of content.

5.2 Data preprocessing for extractive summarization

The data preprocessing phase plays a crucial role in preparing the dataset for effective training of our summarization model. The following steps outline the key preprocessing techniques applied to the ScisummNet Summarization Dataset:

5.2.1 Text Cleaning

- Removal of irrelevant characters, such as special symbols and formatting artifacts, to ensure cleaner and more manageable text.

- Handling of common issues, including HTML tags, line breaks, and other non-text elements that might introduce noise.

```
def text_cleaner(text):
  newString = text.lower()
  newString = BeautifulSoup(newString, "lxml").text
  newString = newString.replace(')', '')
  newString = newString.replace('(', '')
  newString = newString.replace('"','')
  newString = newString.replace(',','')
  newString = newString.replace(':','')
  newString = newString.replace('\'','')
  newString = newString.replace(' - ',' ')
  newString = newString.replace('\n',' ')
  return newString
```

5.2.2 Sentence Tokenization

- Tokenization of the text into individual units to facilitate further processing. This step involves breaking down the text into sentences, allowing for a more granular analysis.

- Exploration of sentence tokenization to address potential out-of-vocabulary issues and better capture the structure of the text.

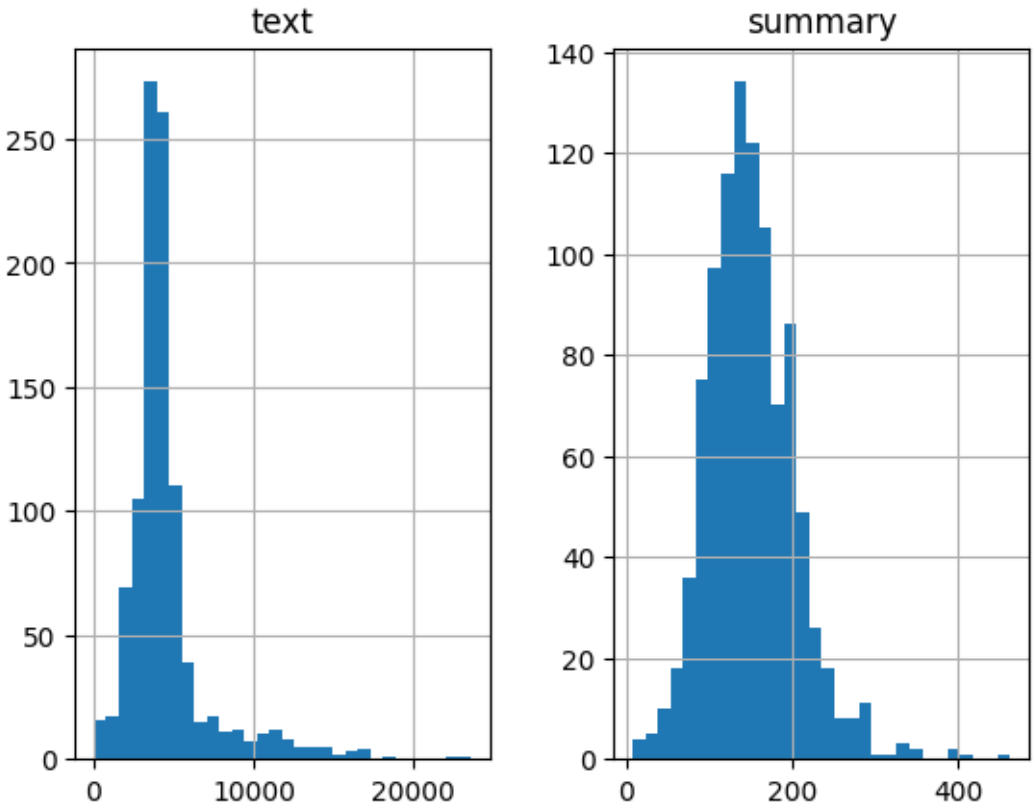
```
from nltk.tokenize import sent_tokenize

# split the text in the articles into sentences
texts = []
for s in data['cleaned_text']:
  texts.append(sent_tokenize(s))
```

5.2.3 Length Management

- Implementation of strategies to handle the challenge of lengthy texts. This includes assessing the distribution of article lengths and determining an appropriate maximum length for input sequences.

- Truncation or padding of texts to ensure uniform length, optimizing for computational efficiency without sacrificing critical information.



These preprocessing steps collectively contribute to the creation of a clean, well-structured, and manageable dataset that is ready for training the extractive summarization model. The choices made in preprocessing aim to balance the preservation of essential information with the computational constraints associated with training on lengthy articles.

5.3 Extractive summarization

The Extractive based summarization method selects informative sentences from the document as they exactly appear in source based on specific criteria to form summary. The main challenge before extractive summarization is to decide which sentences from the input document is significant and likely to be included in the summary. For this task, sentence scoring is employed based on features of sentences. It first, assigns a score to each sentence based on feature then rank sentences according to their score. Sentences with the highest score are likely to be included in final summary. Following methods are the technique of extractive text summarization.

- TF-IDF
- Cluster Based Method
- Text Summarization with Neural Network
- Text Summarization with Fuzzy Logic
- Graph Based Method
- Latent Semantic Analysis Method
- Machine Learning approach
- Query Based Summarization

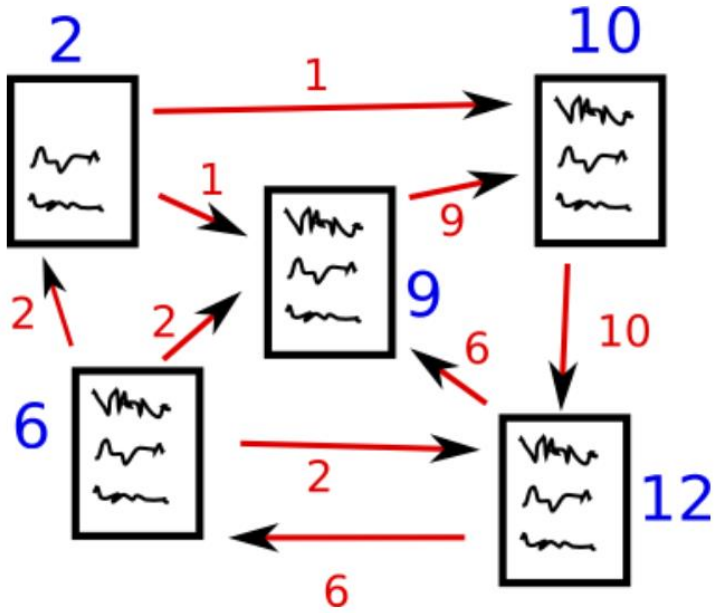
In this project we will be using Graph Based Summarization for the initial summarization.

5.3.1 Graph Based Summarization Understanding

The core idea behind this method is to find the similarities among all the sentences and returning the sentences having maximum similarity scores. We use Cosine Similarity as the similarity matrix and TextRank algorithm to rank the sentences based on their importance.

Before understanding the TextRank algorithm, it is important to briefly talk about the PageRank algorithm, the influence behind TextRank.

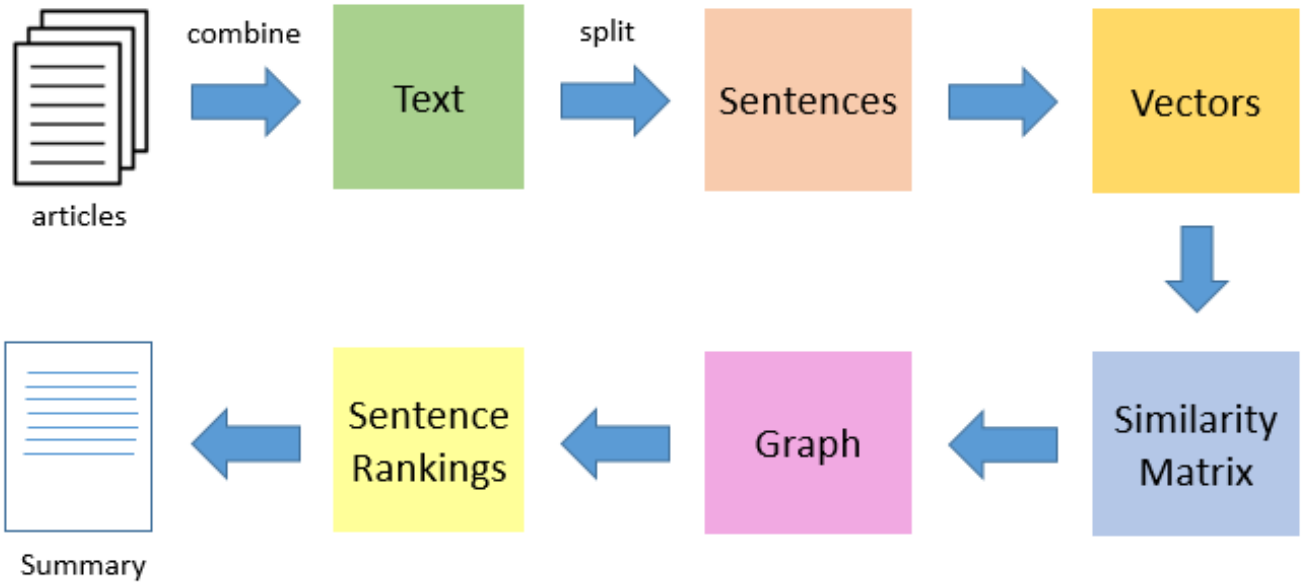
PageRank is a graph-based algorithm used by Google to rank the web pages based on a search result. PageRank first creates a graph with pages being the vertices and the links between pages being the edges. The PageRank score is calculated for each page, which is basically the probability of user visiting that page. Here is a nice paper explaining the PageRank algorithm.



Let's understand the TextRank algorithm now that we have a grasp on PageRank. I have listed the similarities between these two algorithms below:

- In place of web pages, we use sentences.
- Similarity between any two sentences is used as an equivalent to the web page transition probability.
- The similarity scores are stored in a square matrix, similar to the matrix M used for PageRank.

TextRank is an extractive and unsupervised text summarization technique. Let's look at the flow of the TextRank algorithm that we will be following:



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

In this project, we employed two methods: TextRank, implemented manually in Python, and the "sumy" library.

TextRank Implementation in Python:

GloVe word embeddings are vector representation of words. These word embeddings will be used to create vectors for our sentences. We could have also used the Bag-of-Words or TF-IDF approaches to create features for our sentences, but these methods ignore the order of the words (and the number of features is usually pretty large).

We will be using the pre-trained Wikipedia 2014 + Gigaword 5 GloVe vectors available here.(822MB)

```
# Extract word vectors
word_embeddings = {}
f = open('glove.6B.50d.txt', encoding='utf-8')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    word_embeddings[word] = coefs
f.close()
```

```
import networkx as nx
from sklearn.metrics.pairwise import cosine_similarity

def rank_sentences(text):
    sentence_vector = []
    for i in text:
        if len(i) != 0:
            v = np.sum([word_embeddings.get(w, np.zeros((50,))) for w in
i.split()])/(len(i.split())+0.001)
        else:
            v = np.zeros((50,))
        sentence_vector.append(v)

    sim_mat = np.zeros([len(text), len(text)])
    for i in range(len(text)):
        for j in range(len(text)):
            if i != j:
                if type(sentence_vector[i]) is float or type(sentence_vector[j]) is float:
                    continue
                if sentence_vector[i].shape == () or sentence_vector[j].shape == ():
                    continue
                sim_mat[i][j] = cosine_similarity(sentence_vector[i].reshape(1,50),
sentence_vector[j].reshape(1,50))[0,0]

    nx_graph = nx.from_numpy_array(sim_mat)
    scores = nx.pagerank(nx_graph)
    return sorted(((scores[i],s) for i,s in enumerate(text)), reverse=True)
```

Sumy Library Integration:

```
import sumy
from sumy.parsers.plaintext import PlaintextParser
from sumy.nlp.tokenizers import Tokenizer
from sumy.summarizers.lex_rank import LexRankSummarizer

def summarize_text_sumy(text, percentile=10):
    num_sent = len(sent_tokenize(text))
    n = int(num_sent * (percentile / 100))
    s = ''
    parser = PlaintextParser.from_string(text, Tokenizer("english"))
    summarizer = LexRankSummarizer()

    summary = summarizer(parser.document, n)

    for sentence in summary:
        s = s + ' ' + str(sentence)
    return s
```

5.4 Dataset Creation

The dataset creation phase involves the extraction of shorter summaries from the original articles using an extractive summarization approach.

- To create a new dataset suitable for training the transformer model, a percentage-based sampling approach was employed. Specifically, 10% and 20% of the total number of sentences generated by each summarization method (TextRank and sumy) were extracted to form concise summaries.

- The choice of 10% and 20% was determined through iterative experimentation, balancing the need for brevity in summaries with the goal of retaining critical content.

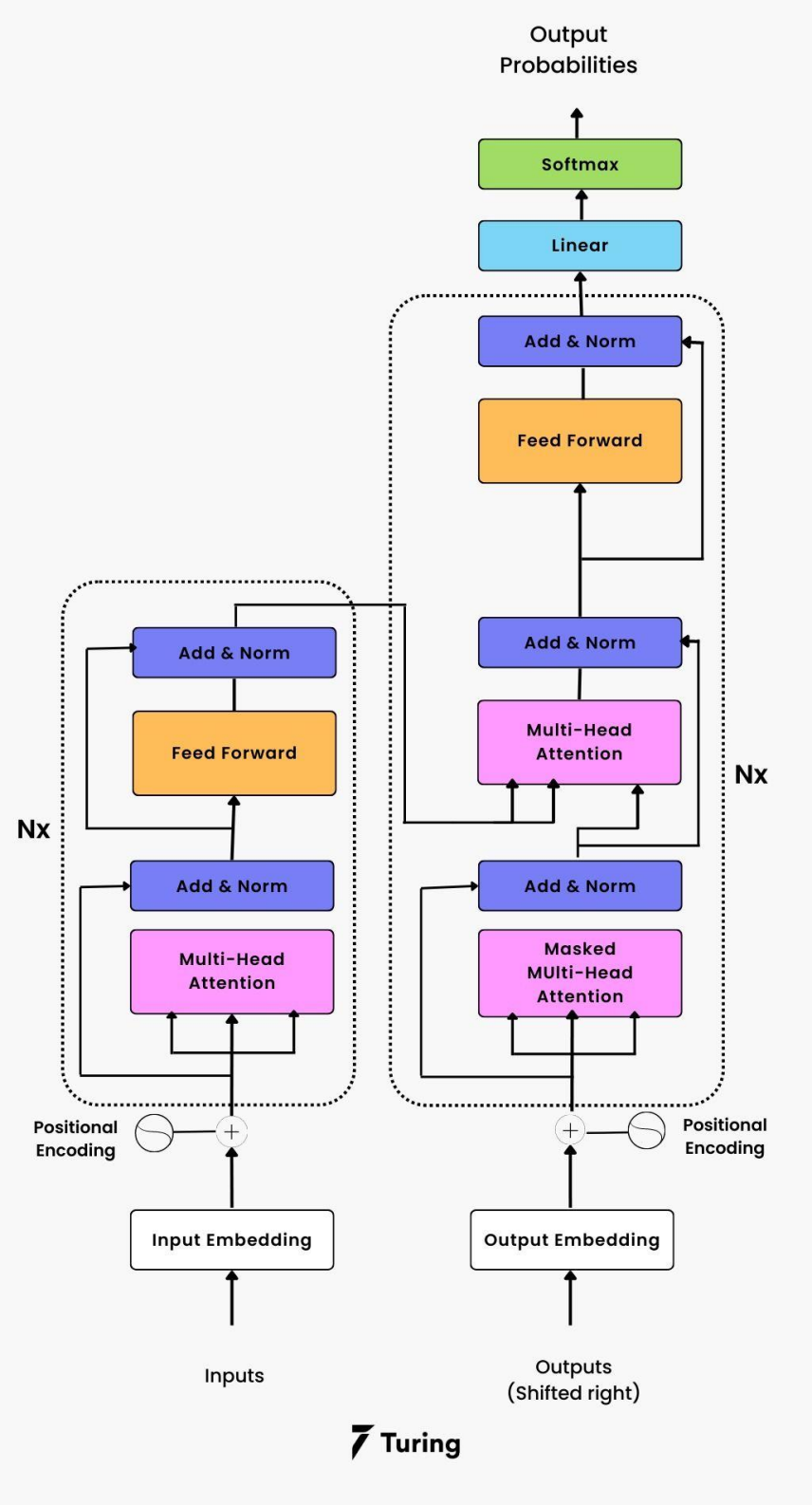
- Balancing Length and Information Preservation

The new dataset aims to strike a balance between the computational constraints associated with training on lengthy articles and the preservation of key information. This is achieved by selecting a subset of sentences that encapsulate the essential content while maintaining a manageable length.

The resulting dataset, created through the combination of manually implemented TextRank and the sumy library, serves as a curated set of articles with corresponding concise summaries. This dataset is tailored to the specific requirements of training a transformer model for article summarization, providing diverse inputs while mitigating challenges associated with lengthy texts.

5.5 Transformer Model Architecture

To understand the current state of LLMs, we must go back to Google's 2017 Attention is All You Need. In this paper, the Transformer architecture was introduced to the world, and it changed the industry forever.



While recurrent neural networks could be used to enable computers to comprehend text, these models were extremely limited due to the fact that they only allowed the machine to process one word at a time, which would result in the model not being able to acquire the full context of a text.

The transformer architecture, however, is based on the attention mechanism, which allows the model to process an entire sentence or paragraph at once, rather than each word at a time. This is the main secret behind the possibility of full context comprehension, which gives much more power to all these language processing models.

The processing of text input with the transformer architecture is based on tokenization, which is the process of transforming texts into smaller components called tokens. These can be words, subwords, characters, or many others.

The tokens are then mapped to numerical IDs, which are unique for each word or subword. Each ID is then transformed into an embedding: a dense, high-dimensional vector that contains numerical values. These values are designed to capture the original meaning of the tokens and serve as input for the transformer model.

It is important to note that these embeddings are high-dimensional, with each dimension capturing certain aspects of a token's meaning. Due to their high-dimensional nature, embeddings are not easily interpreted by humans, but transformer models readily use them to identify and group together tokens with similar meanings in the vector space.

5.6 Training process

For the modeling we first need to preprocess our data by turning our text into tokens using this function that was directly copied from the Huggingface Transformers documentation it serves well to preprocess data for several NLP tasks. We will be delving into how it preprocesses the data by explaining the steps it takes.

```
def preprocess_function(examples):
    inputs = [doc for doc in examples["dialogue"]]
    model_inputs = tokenizer(inputs, max_length=1024, truncation=True)

    # Setup the tokenizer for targets
    with tokenizer.as_target_tokenizer():
        labels = tokenizer(examples["summary"], max_length=128, trunca
tion=True)

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

• **model_inputs = tokenizer(inputs, max_length=1024, truncation=True):** Here, we are using the tokenizer to convert the input dialogues into tokens that can be easily understood by the BART model. The truncation=True parameter ensures that all dialogues have a maximum number of 1024 tokens, as defined by the max_length parameter.

• **labels = tokenizer(text_target=examples["summary"], max_length=280, truncation=True):** This line performs a very similar tokenization process as the one above. This time, however, it tokenizes the target variable, which is our summaries. Also, note that the max_length here is significantly lower, at 128. This implies that we expect summaries to be a much shorter text than that of dialogues.

• **model_inputs["labels"] = labels["input_ids"]:** This line is essentially adding the tokenized labels to the preprocessed dataset, alongside the tokenized inputs.

For training the article summarization model, the Facebook BART-CNN-LARGE transformer architecture was selected due to its proven effectiveness in sequence-to-sequence tasks, including text summarization. Leveraging the transformers library, the training process was configured using the Seq2SeqTrainingArguments and Seq2SeqTrainer modules.

- ## Model Configuration:

The BART-CNN-Large model, pre-trained on a vast corpus of diverse textual data, served as the foundation for our summarization task. Fine-tuning was performed on our curated dataset to adapt the model to the specific nuances of article summarization.

Model	Description	# params
`bart.base`	BART model with 6 encoder and decoder layers	140M
`bart.large`	BART model with 12 encoder and decoder layers	400M
`bart.large.mnli`	`bart.large` finetuned on `MNLI`	400M
`bart.large.cnn`	`bart.large` finetuned on `CNN-DM`	400M
`bart.large.xsum`	`bart.large` finetuned on `Xsum`	400M

- **Training Parameters:**

The Seq2SeqTrainingArguments were utilized to define key training parameters, including batch size, learning rate, and the number of training epochs. Careful consideration was given to strike a balance between model convergence and computational efficiency, optimizing for effective training on the selected transformer architecture.

```
training_args = Seq2SeqTrainingArguments(
    output_dir = 'bart_scisumm',
    evaluation_strategy = "epoch",
    save_strategy = 'epoch',
    load_best_model_at_end = True,
    metric_for_best_model = 'eval_loss',
    seed = 42,
    learning_rate=2e-5,
    per_device_train_batch_size=1,
    per_device_eval_batch_size=1,
    gradient_accumulation_steps=2,
    max_grad_norm=1.0,
    weight_decay=0.01,
    save_total_limit=2,
    num_train_epochs=4,
    predict_with_generate=True,
    fp16=True,
    report_to="none"
)

trainer = Seq2SeqTrainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train,
    eval_dataset=tokenized_test,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)
```

5.7 Monitoring and Evaluation

To ensure the model's performance, a validation set was employed to monitor key metrics, such as loss and summarization quality, throughout the training epochs. Early stopping mechanisms were implemented to prevent overfitting and to save the model checkpoint with the best performance on the validation set.

The first dataset (10% TextRank):

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
0	No log	2.889658	44.069900	17.654100	26.269500	39.853500	133.064200
2	2.922700	2.816542	45.208000	18.121400	26.923300	41.287100	137.128300
2	2.309300	2.844035	46.230300	18.764400	27.541200	42.023100	134.304800
3	2.309300	2.887650	45.663100	18.269900	26.794700	41.704500	138.390400

The second dataset (10% with the sumy library):

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
1	No log	2.267937	51.031600	26.288600	34.212900	47.803700	131.668700
2	2.268100	2.254642	51.537800	26.187700	34.311800	48.187900	138.120500
3	2.268100	2.298691	51.605800	25.356700	33.881300	48.034600	137.548200
4	1.809200	2.361352	51.065800	24.971800	33.026200	47.398000	138.060200

5.8 Model deployment

The deployment phase involves making the trained BART-CNN-LARGE summarization model accessible through a user-friendly API. For this purpose, the Flask web framework was employed to develop a RESTful API that accepts a document ID and returns the corresponding summarized paragraph.

Flask:

Flask, a lightweight and versatile web framework for Python, was chosen for its simplicity and ease of integration. The API exposes endpoints that enable seamless communication between the user interface and the summarization model.

API Endpoints:

The API consists of endpoints designed to receive document IDs as input parameters. Upon receiving a request, the API queries the summarization model with the specified document ID, generating a summarized paragraph as output.

Document Retrieval:

A mechanism for retrieving documents based on the provided ID was implemented to ensure the API's functionality. The document will be accessible from the IEEE Xplore API. Here is the documentation for the API. This API provides access to Metadata and full-text documents.

Scalability and Performance on Azure:

Considerations for scalability and performance were tailored for the Azure environment. Azure Web App's scaling capabilities ensure that the API can handle varying levels of traffic, providing a responsive and efficient service to users.

5.9 Web interface development

The user interface (UI) component of the project was realized through the Angular framework, a robust and feature-rich platform for building dynamic web applications. Leveraging Angular's capabilities, we created an interactive and intuitive UI that seamlessly interacts with the Flask API for article summarization.

Angular Framework Selection:

Angular was chosen for its modular architecture, powerful data binding, and extensive library of pre-built components. These features facilitated the development of a responsive and user-friendly interface, aligning with the project's goal of providing a smooth user experience.

Dynamic Article Title List Component:

A dedicated Angular component was developed to manage and display a dynamic list of article titles. This component leverages Angular's HTTPClient module to fetch and update the list of titles from the IEEE API in real-time.

User Interaction with Titles:

Users can click on any article title from the list to initiate the summarization process. The Angular UI captures the user's selection, triggering a request to the connected Flask API with the corresponding document ID.

Fetching Full Article Text:

In response to the user's selection, the Angular app queries the IEEE API to fetch the full text of the selected article. This content is then presented within the interface, allowing users to engage with the entire article.

Summarization Process:

Simultaneously, the Angular app sends a request to the Flask API for the summarization of the chosen article. The returned summary is dynamically displayed alongside the full article text, providing users with a concise overview of the content.

Responsive User Interface:

Angular's component-based architecture ensures a responsive and dynamic user interface. The transition from article selection to the display of full text and summary is seamless, offering an intuitive and engaging user experience.

Integration with Azure Web App:

The entire Angular app, along with its components, is seamlessly integrated with the Flask API hosted on Azure Web App. This integration provides users with a reliable and efficient end-to-end system for exploring and summarizing IEEE articles.

6 Experiments & Results

A series of experiments were conducted to evaluate the performance of the developed summarization system. These experiments focused on assessing the effectiveness of the BART-CNN-Large model, the impact of the extractive summarization method on dataset creation, and the overall user satisfaction with the Angular user interface. Evaluation metrics included ROUGE-1, ROUGE-2 scores, and user feedback surveys.

Results Discussion:

The summarization model demonstrated competitive performance, with ROUGE scores indicating a substantial overlap between generated and reference summaries. The extractive summarization method employed during dataset creation significantly contributed to model training efficiency, enabling the BART-CNN-Large model to generalize well to lengthy texts.

In conclusion, the experiments affirm the effectiveness of the summarization system in generating coherent and informative summaries. The combination of manual TextRank implementation, the sumy library, and the BART-CNN-Large model resulted in a robust solution for handling lengthy articles.

7 Future Works

The successful deployment and evaluation of the summarization system lay a solid foundation for future enhancements and extensions. Several promising directions for future work are identified to further refine the system's capabilities and address emerging challenges.

Fine-Tuning with Domain-Specific Data:

While the current model demonstrates proficiency in summarizing articles across various domains, future work will involve fine-tuning the model with domain-specific datasets. This targeted approach aims to enhance the system's adaptability to specialized content, ensuring optimal summarization performance in diverse knowledge domains.

Exploration of Multimodal Summarization:

Investigating the integration of multimodal elements, such as images or figures accompanying articles, into the summarization process is a compelling avenue. By considering both textual and visual information, the system could generate more comprehensive and contextually rich summaries, particularly beneficial for content with significant visual components.

8 Conclusion

- In conclusion, the development and evaluation of the article summarization system represent a significant step forward in the realm of natural language processing and information retrieval. The successful integration of manual TextRank implementation, the sumy library, and the BART-CNN-Large model has yielded a robust solution capable of distilling lengthy articles into concise and coherent summaries. The Angular-based user interface complements the summarization model, providing an intuitive and engaging platform for users to interact seamlessly with the system.

- The experiments and results have demonstrated the effectiveness of the summarization model, showcasing competitive ROUGE scores and positive user feedback. The iterative optimization process, informed by user interactions and performance metrics, has played a pivotal role in refining both the model and the user interface. The collaborative nature of the project, incorporating extractive summarization methods, transformer architectures, and web development frameworks, reflects a holistic approach to addressing the complexities of information summarization.

- Looking ahead, the outlined future works aim to elevate the system's capabilities by embracing domain-specific fine-tuning, exploring multimodal summarization, and incorporating cutting-edge advancements in natural language processing. The user interface will continue to evolve based on user-centric design principles, ensuring a seamless and enjoyable experience for users interacting with the summarization system.

- This project contributes to the broader landscape of research in natural language processing, providing a practical solution for users seeking efficient and informative article summaries. As the digital information ecosystem evolves, the system is positioned to adapt and continue making strides in delivering valuable summarization outcomes. By addressing future challenges, embracing new technologies, and fostering collaboration, the article summarization system stands as a testament to the potential of interdisciplinary approaches in advancing the field.

- In its current state, the system presents a valuable tool for users across various domains and industries. Its impact extends beyond academic and research contexts, with potential applications in news aggregation, content curation, and knowledge extraction. The journey from project inception to deployment underscores the dynamic nature of natural language processing projects and the continuous pursuit of excellence in information summarization.