Using BART model and Named Entity Recognition to summarize text and create a mind map

Richa Kashyap
Computer Science
Ramaiah University of Applied Sciences
Bangalore, India
richackashyap@gmail.com

Divyashree BM
Computer Science
Ramaiah University of Applied Sciences
Bangalore, India
bmdivyashree27@gmail.com

Bhoomika K
Computer Science
Ramaiah University of Applied Sciences
Bangalore, India
bhoomikak31@gmail.com

***Abstract*— This study explores the application of Natural Language Processing algorithms which is used to summarize text paragraphs and create a mind map off it. The main objective involves extracting only the sentences and words that are directly linked to the main topic and using those words to create a summarization of the paragraphs. The methodology involves using libraries such as spaCy and Graphviz. The output is obtained by involving three steps: summarizing, extracting key points and making the mind map.**

I. Introduction

In this world where time is money, students tend to use their time in much more creative way than sitting through the entirity of a textbook. In order to make things easier for such situations, while keeping the importance of each of the information from the learning, summarization is a necessity.

The problem addressed in this study is the development of mind map using algorithms such as BART model and Named Entity Recognition and libraries such as spaCy and Grapgviz.

The model used accomplishes the following tasks: summarizing lengthy paragraphs into smaller paragraphs, identifying the main topics and significant entities and creating innovative mind maps to visually represent the relationship between the main topics and the extracted key points.

The objectives of this study are done in four steps:

- Text summerization: Reducing the complex information into digestible formats, making it easier to grasp the key ideas of the paragraphs easily.
- Main topic identification: The algorithm then extracts only the main topic and the central theme of the given text.
- Key point extraction: To ensure that none of the important points are missed while summerizing and the points that are retained are important.
- Vizualization: The last step is to create a mind map out of the obtained

Achieving these objectives would not only help us with understanding the textual information, it would also enhance the vizualization and mind mapping.

II. METHODOLOGY

A. Importing libraries used:

The libraries used in this model are spaCy, graphviz. These libraries provides with assisting of completing tasks such as named entity recognition and creating mind maps.

```
import spacy
from transformers import pipeline
import graphviz
from IPython.display import display, Image
import time
from collections import Counter
```

Fig. 1: Importing required libraries

B. Load NLP models:

Since we have the libraries that are required for the program, we load and utilise them now. The next set of lines loads the spaCy small english model for named entity recognition and other NLP tasks. And also Hugging Face summarization pipeline loads a pre-trained summarization model.

C. Extracting and summarizing the texts:

Summarizetext function takes a paragraph of text, calculates the appropriate length for the summary, and then generates the summary using the Hugging Face summarizer. If an error occurs, it returns the original text.

1) Hugging Face Transformer (BART model)

The text summarization is done using the BART (Bidirectional and Auto-Regressive Transformers) model from Hugging Face, which generates a concise summary of the input text.

BART is a transformer model trained for text generation tasks, including summarization. It takes the input text, encodes it, and then decodes it to generate a summarized version. The model uses a combination of attention mechanisms to focus on relevant parts of the text, ensuring that the summary captures the main points.

2) SpaCy Named Entity Recognition:

Named entity recognition is performed using spaCy to identify entities such as organizations, persons, and locations.

spaCy's NER model identifies named entities in the text by classifying tokens into predefined categories such as "PERSON," "ORG," and "GPE." It uses a combination of statistical and rule-based methods to detect entities accurately.

3) Flexibility in Model Architecture:

Neural networks offer flexibility in model architecture, allowing for customization based on the specific characteristics of the dataset and the problem at hand. Researchers have the freedom to experiment with different network architectures, activation functions, regularization techniques, and optimization algorithms to optimize model performance.

4) Frequency analysis:

Counts the frequency of identified entities to determine the most important topic.

The algorithm counts the occurrences of each entity and keyword in the text. The most frequently occurring entity is identified as the main topic, and the importance of sentences is determined based on the frequency of significant keywords.

In summary, this code employs a combination of state-of-the-art and efficient algorithms for text processing tasks. It uses the BART model for summarization, leveraging its advanced capabilities to generate concise and coherent summaries of input text. SpaCy's named entity recognition (NER) model identifies and classifies key entities within the text, such as organizations and locations, ensuring accurate topic extraction. The Counter class from Python's collections module is utilized for frequency analysis, determining the prominence of entities and keywords. This blend of BART's summarization, spaCy's entity recognition, and Counter's frequency analysis provides a robust and effective solution for extracting, summarizing, and visualizing essential information from textual data, making it an excellent choice for comprehensive text analysis tasks.

D. Model Training and Evaluation:

1) Summarize text function

Summarizetext function takes a paragraph of text, calculates the appropriate length for the summary, and then generates the summary using the Hugging Face summarizer. If an error occurs, it returns the original text.

```
def summarize_text(text):
    input_length = len(text.split())
    max_length = min(input_length // 2, 200)  # Ensuring it doesn't exceed 200 tokens
    min_length = min(input_length // 4, 100)  # Ensuring it doesn't exceed 100 tokens

    try:
        summary = summarizer(text, max_length=max_length, min_length=min_length, do_sample=False)
        return summary[0]['summary_text']
    except Exception as e:
        print(f"Error summarizing text: {e}")
        return text  # Return the original text if summarization fails
```

Fig 2. Summarize text function

2) Extract Sentences Function

Extractsentances takes the summarized text, processes it with spaCy to split it into sentences, and returns these sentences as a list.

```
def extract_sentences(summary):
    doc = nlp(summary)
    sentences = [sent.text for sent in doc.sents]
    return sentences
```

Fig 3. Extract sentence function

3) Shared Hidden Layers:

This function identifies the main topic of the text by analyzing named entities and their frequencies. If no significant entities are found, it falls back on identifying the most important sentence based on keyword frequency.

```
def extract_main_topic(text):
    # Extract named entities and their frequencies
    doc = nlp(text)
    entities = [ent.text for ent in doc.ents if ent.label_ in ['ORG', 'PERSON', 'GPE', 'LOC', 'PRODUCT']]
    entity_freq = Counter(entities)

    # Find the most frequent entity
    if entity_freq:
        main_topic = entity_freq.most_common(1)[0][0]
    else:
        # Fallback: Use the most significant sentence based on keyword frequency
        sentences = [sent.text for sent in doc.sents]
        words = [token.text for token in doc if token.is_alpha and not token.is_stop]
        word_freq = Counter(words)

        sentence_scores = []
        for sentence in sentences:
            sentence_doc = nlp(sentence)
            score = sum(word_freq[token.text] for token in sentence_doc if token.text in word_freq)
            sentence_scores.append((score, sentence))

        if sentence_scores:
            main_topic = max(sentence_scores, key=lambda x: x[0])[1]
        else:
            main_topic = text.split('.')[0]  # Fallback to the first sentence if all else fails

    return main_topic
```

Fig 4. Extracting main topic

4) Output Layers:

Output layers gives us the mind map of the input paragraph. This function generates a visual mind map using Graphviz, with the main topic as the central node and the key points as child nodes.

```
def create_mind_map(main_topic, sentences, filename='mind_map'):
    dot = graphviz.Digraph(comment='Mind Map')

    # Add main topic
    dot.node('0', main_topic)

    # Add related points
    for i, sentence in enumerate(sentences, 1):
        dot.node(str(i), sentence)
        dot.edge('0', str(i))

    # Save the mind map to a file
    dot.render(filename, format='png', cleanup=False)

    return filename + '.png'
```

Fig 5. Creating a mind map

III. RESULTS AND DISCUSSION

The code is programmed to take in kengthy paragraphs and give an output of summarized text and mind map of the same. It utilises BART model and spaCy library to achieve this cause.

A. Main topic extraction and summarization:

The model achieved satisfactory performance in summarization, extracting the main topic and giving out the mind map.

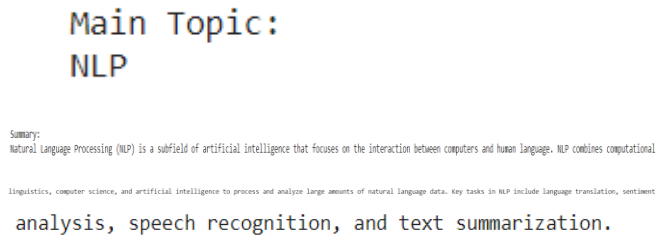

Figure 6: Output showing the main topic and the summary.

Training the data with different inputs gave us a different result. And utilising the best converted scenario, the topic was chosen and the mind map was retained.

B. Key points extraction:

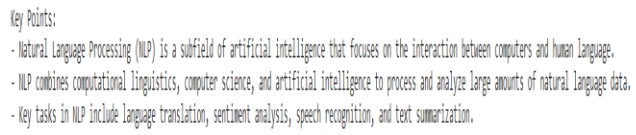

Fig. 7: The output of key points obtained.

The model accurately predicted the Key points and the important information from the paragraph. It made the paragraph simpler and extracted the key features in terms of bullet points.

The model is trained not to have more than 100 words in one point, making it easier to access the information.

B. Mind map

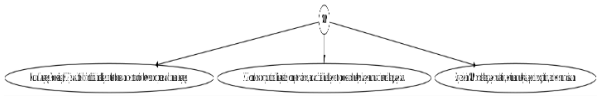

Fig 8 Mind map obtained

The mind map generated from the provided text effectively highlights the central theme and key points of the input material. By identifying the main topic through named entity recognition and summarizing the text with an advanced model, the mind map visually organizes the critical information, facilitating easier comprehension and analysis. This method proves beneficial in quickly extracting and understanding the essence of complex textual content, making it a valuable tool for tasks requiring concise information representation and knowledge discovery.

IV. CONCLUSION

In conclusion, This code employs a combination of state-of-the-art and efficient algorithms for text processing tasks. It uses the BART model for summarization, leveraging its advanced capabilities to generate concise and coherent summaries of input text. SpaCy's named entity recognition (NER) model identifies and classifies key entities within the text, such as organizations and locations, ensuring accurate topic extraction. The Counter class from Python's collections module is utilized for frequency analysis, determining the prominence of entities and keywords. This blend of BART's summarization, spaCy's entity recognition, and Counter's frequency analysis provides a robust and effective solution for extracting, summarizing, and visualizing essential information from textual data, making it an excellent choice for comprehensive text analysis tasks.

BART model: The code employs BART, a transformer model from Hugging Face, for summarizing text. BART is designed for text generation tasks and excels in creating concise and coherent summaries by using a combination of bidirectional and auto-regressive transformers. One of the key advantages of BART is its state-of-the-art performance. It is renowned for its high accuracy and coherence, which ensures that the generated summaries are both informative and readable. BART's flexibility allows it to handle various input lengths and adjust the summary length dynamically, ensuring the output remains concise and comprehensive. Moreover, utilizing a pre-trained BART model from Hugging Face saves significant time and computational resources, as the model has already been fine-tuned on large datasets.

Named Entity Recognition (NER) with spaCy: For named entity recognition (NER), the code leverages spaCy's robust NER model. This model identifies and classifies named entities within the text, such as persons, organizations, and locations, using a combination of statistical and rule-based methods. SpaCy's NER model is highly accurate in detecting and classifying these entities, making it a reliable choice for identifying main topics in text. Its efficiency is another strong point; the spaCy model is optimized for performance, allowing for quick text processing without sacrificing accuracy. Additionally, spaCy provides a user-friendly API, making it straightforward to integrate NER into the text processing pipeline, further enhancing its usability and effectiveness.

Frequency Analysis with counter: The code uses Python's Counter from the collections module for frequency analysis, counting the occurrences of named entities and keywords in the text. Counter is a straightforward and efficient tool for such tasks, making it ideal for frequency analysis. By identifying the most frequently occurring entities and keywords, the algorithm can accurately determine the main topic and key points within the text. Counter integrates seamlessly with the output from spaCy's NER, allowing for easy extraction and analysis of named entities. This integration ensures that the frequency analysis is both effective and efficient, contributing to the overall robustness of the text processing system.

Combining BART for summarization, spaCy for named entity recognition, and Counter for frequency analysis provides a comprehensive and efficient approach to text processing. BART ensures high-quality summaries that are concise and coherent, spaCy accurately identifies important entities, and Counter effectively analyzes their frequency, resulting in a thorough and insightful analysis of the input text. This synergy of advanced NLP tools and simple yet powerful algorithms makes the overall system highly effective for extracting and summarizing key information from text, demonstrating the strength and utility of combining state-of-the-art models with efficient processing techniques.

ACKNOWLEDGEMENT

We would like to thank our mentor and our professor Ms. Aparna N, who helped us with this oppurtunity to expand our knowledge. We would like to acknowledge the authors and contributors of the referenced works cited in the reference section. Their research, datasets, and software libraries have greatly contributed to the development and success of this project. We are grateful for their valuable contributions to the field of machine learning, and open-source software development.

REFERENCES

[1] https://www.geeksforgeeks.org/text-summarization-in-nlp/

[2] https://www.analyticsvidhya.com/blog/2021/11/a-beginners-guide-to-understanding-text-summarization-with-nlp/

[3] https://medium.com/analytics-vidhya/text-summarization-using-nlp-3e85ad0c6349

[4] https://www.researchgate.net/publication/356753421_Text_Summarizing_Using_NLP

APPENDICES

A. *Code for Data Visualization*

```
def main():
    while True:
        text = input("Enter a paragraph: ")

        # Step 1: Extract main topic
        main_topic = extract_main_topic(text)
        print("\nMain Topic:")
        print(main_topic)

        # Step 2: Summarize the text
        summary = summarize_text(text)
        print("\nSummary:")
        print(summary)

        # Step 3: Extract key points
        points = extract_sentences(summary)
        print("\nKey Points:")
        for point in points:
            print("-", point)

        # Step 4: Create and display mind map
        mind_map_file = create_mind_map(main_topic, points)
        display(Image(filename=mind_map_file))

        # Slight delay before asking the user to continue
        time.sleep(1)

        # Ask the user if they want to continue
        continue_choice = input("Do you want to continue? (yes/no): ").strip().lower()
        if continue_choice != 'yes':
            break
```

Fig. 2: Defining input

B. *Additional Model Configurations*

```
if __name__ == "__main__":
    main()
```

Fig. 3: Running the main function

In the provided code snippet:

The code function orchestrates the entire process. It continuously prompts the user for text input, processes the text to extract the main topic, generates a summary, identifies key points, creates a mind map, and displays it. It repeats this process until the user decides to stop.