RAGNOVEL: Retrieval Augmented Summarization and Question Answering of Literary Texts

Devcharan Krishna Naik, Yashika Patil, Mousami Biswas

May 2024

Abstract

This project addresses the challenge of efficiently summarizing extensive literary and academic texts using state-of-the-art language models and data mining algorithms. In our project, we aim to enhance the accessibility and navigation through large documents such as novels and academic theses, through two primary mechanisms: a summarization system tailored for narrative texts and an interactive information retrieval system that responds to user queries. The integration of extractive and abstractive summarization methods. We are also using advanced language processing techniques like TextRank and Retrieval Augmented Generation (RAG) which allows for dynamic interaction with texts. This approach not only conserves the theory of the original content but also significantly reduces time and effort required to extract key insights from the text.

1 Introduction

Text summarization and Question and Answering (QnA) are increasingly growing domains in text data mining, dealing with the challenge of considering large volumes of text into concise summaries without losing the essence and meaning of the original content. Using natural language processing (NLP) is crucial for addressing the large volumes of text in the digital age. This project develops an advanced system leveraging models like LLM and data mining algorithms to effectively summarize such texts in a meaningful manner.

The approach we follow is to initially create a summarization system tailored for narrative texts, enhancing reader engagement and accessibility. Secondly, to develop an interactive information retreival system that enables users to query and navigate through large textual data easily through a user-friendly interface. Such an approach combines the benefits of extractive and abstractive summarization methodologies. Extractive summarization involves selecting significant portions of the original text based on their thematic and informational importance, maintaining the text's original wording. In contrast, abstractive summarization

uses advanced language models to reformulate the essence of texts, generating new, concise versions that convey the same information in fewer words.

In addition to text summarization, another key aspect of this project involves developing an interactive question answering system. With the exponential growth of digital information, users often struggle to find relevant answers to their queries within large volumes of textual data. By integrating RAG and natural language processing (NLP) techniques, our system aims to address this challenge by providing users with the ability to ask questions and receive accurate and relevant answers in real-time. Leveraging advanced language models and data mining algorithms, our question answering system enables users to navigate through complex textual data with ease, facilitating a more efficient and intuitive information retrieval process.

2 Background

The field of text summarization is broadly divided into extractive and abstractive approaches. Extractive methods, such as the TextRank algorithm, identify and compile significant parts of the text based on their relevance and connectivity. These methods preserve the original text's wording, making them reliable but sometimes lacking in cohesion.

On the other hand, abstractive summarization uses language models to generate new text that conveys the same information in a condensed form, allowing for more natural and cohesive summaries. Our project also involves Retrieval Augmented Generation (RAG), enhancing language models capability by integrating external data during the generation process. This method supports more accurate and contextually relevant summaries and interactive query responses, thereby setting our approach apart from traditional text summarization techniques.

2.1 Retrieval Augmented Generation (RAG)

Recent advancements in natural language processing (NLP) have introduced innovative techniques for text summarization and question answering, among which Retrieval Augmented Generate (RAG) models stand out. RAG models signify a significant shift in text summarization strategies, blending the strengths of retrieval-based and generation-based approaches. Language models utilize the parametrized knowledge gained from their training to answer general questions that may be asked to chatbots like ChatGPT. However, for more focused tasks like specific document summarization and question answering, we need a way to provide additional information to the language model.

The RAG framework consists of two main components: a retriever and a generator. The retriever module is responsible for efficiently retrieving relevant passages or documents from a large corpus of text based on a given query or context. This retrieval process is often facilitated by techniques such as dense vector retrieval or sparse vector retrieval, which enable

the model to identify salient information effectively.

Once the retriever has identified relevant text passages, the generator module (LLM) synthesizes these passages into coherent and concise summaries. Unlike traditional extractive summarization methods that select and concatenate existing text segments, the generator in RAG models employs neural language generation techniques to produce human-like summaries. This generation process enables the model to capture the essence of the input text while ensuring coherence and fluency in the generated summary. RAG models offer flexibility in summarization tasks, allowing users to specify different types of prompts or queries to tailor the summary output according to their preferences making RAG models suitable for a wide range of applications.

2.2 TextRank

TextRank is a graph-based ranking algorithm that has gained prominence in the field of natural language processing, particularly in the domain of text summarization and keyword extraction. Developed by Mihalcea and Tarau in 2004, TextRank was inspired by Google's PageRank algorithm, which revolutionized web search by ranking web pages based on their importance and relevance.

At its core, TextRank treats text documents as graphs, where nodes represent entities such as words or sentences, and edges denote relationships or similarities between these entities. The algorithm assigns importance scores to nodes based on their connections within the graph, similar to PageRank's approach of assigning importance to web pages based on their inbound links. The importance scores derived from TextRank reflect the centrality of nodes within the graph, capturing the significance of each word or sentence in the context of the entire document. By iteratively updating these scores, TextRank identifies key entities that play pivotal roles in conveying the main ideas or themes of the text. Finally, the algorithm selects the highest-scoring nodes to compose a concise summary that captures the most relevant information from the original document.

One of the key strengths of TextRank lies in its simplicity and effectiveness in capturing the semantic relationships between textual units. By leveraging techniques such as cosine similarity or word embeddings, TextRank can identify important keywords, phrases, or sentences within a document, enabling tasks such as automatic summarization or keyword extraction.

Text summarization, in particular, has emerged as a crucial application of TextRank, allowing users to generate concise and informative summaries from large volumes of text. By employing graph-based ranking techniques, TextRank can identify the most salient sentences in a document, which can then be used to construct a coherent summary that preserves the essential information.

Moreover, TextRank's unsupervised nature makes it highly adaptable to various domains and

languages, as it relies solely on the inherent structure of the text rather than external training data. This makes it particularly valuable for tasks such as multi-document summarization, where traditional supervised approaches may face challenges due to the lack of labeled data.

3 Problem Definition

The problem definition for the project includes conducting a comprehensive comparative analysis of different techniques and methodologies for text summarization and question answering. Specifically, the project aims to evaluate the performance and effectiveness of two primary approaches:

1. **Question Answering (QnA):** The project seeks to compare and contrast the performance of two distinct methodologies for question answering: **Retrieval Augmented Generation (RAG) vs. GloVe and TF-IDF embeddings:** This comparison involves assessing the effectiveness of RAG models in generating accurate and relevant responses to user queries, as opposed to traditional techniques such as GloVe and TF-IDF embeddings. RAG models leverage advanced language generation capabilities and retrieval-based mechanisms to provide dynamic and contextually relevant answers, while GloVe and TF-IDF embeddings rely on pre-trained word embeddings and statistical measures to estimate semantic similarities and extract relevant information.

2. **Text Summarization:** The project also aims to evaluate the performance of different summarization techniques: **Retrieval Augmented Generation (RAG) vs. TextRank with and without hashing techniques:** This comparison involves analyzing the efficacy of RAG models in generating concise and informative summaries of textual data, compared to the traditional TextRank algorithm with and without hashing techniques. RAG models utilize a combination of retrieval-based and generation-based approaches to produce summaries that capture the essence of the original content, while TextRank relies on graph-based ranking algorithms to extract key sentences and phrases from the text.

By conducting this comparative analysis, the project seeks to identify the strengths and weaknesses of each approach. This information will not only contribute to advancing the field of natural language processing and text mining but also provide valuable insights for developing more effective and robust systems for text summarization and question answering.

4 Methodology

4.1 RAG for Text Summarization and Question Answering

A Retrieval Augmented Generation (RAG) application was developed using Large Language Models (LLM) within the Python Lang-chain framework. This framework facilitates the

integration of LLMs into applications and provides a structured environment for experimentation and development.

Lang-chain, as the name suggests, is a framework for building chains of modules which perform different tasks. These modules are concatenated based on their input and outputs to perform more complex tasks as a unit. A chain is created to process the input data through various modules. The input is initially processed by the prompt module, followed by the LLaMa 2 model module. Subsequently, the output from the model is passed through a parser module, which returns the results as a string output. This modular approach allows for flexibility in customizing the application's functionality.

Setting up the prompt

The first step involves configuring a prompt to be sent to the language model. A prompt that includes both the context, representing the text data to be queried or summarized, and a question, which specifies the query to be made from the context is constructed. Additional instructions are incorporated into the prompt to guide the LLM in performing the desired tasks.

Handling Text Data

The next step involves processing the text data, which can be a text file or a PDF document. A PDF reader is used to extract the text from each page of the document, which is then stored in a list. The text data is converted into vector representations using Ollama Embeddings, a process that involves encoding each token and sentence into embeddings based on preexisting vector stores.

Creating the Vectorstore

The vectorstore, containing the embeddings for the text data, is generated after incorporating the text data into the Ollama Embeddings. This process can be time-consuming, especially for large documents, as every token and sentence must be converted into embeddings. Once generated, a retriever function is defined to facilitate querying the vectorstore.

Handling Token limit

The Language models being used have a token limit of 4096 tokens which control how much information the model can assess at a time. So, the entire document cannot be incorporated into the chain as "context" for the summarization or question answering task. Larger documents need to be split into smaller documents based on chapters or pages to reduce the number of tokens in each document. To optimize the processing of relevant text data for each question, similarity measures such as cosine similarity are utilized to compare the similarity between the question and the document pages. Only the text data with the highest similarity to the question is sent to the model for processing, enhancing efficiency and accuracy.

Model Execution

With the individual modules in place, the entire chain of the application is constructed. The input, consisting of the PDF document as context and a question or list of questions, is passed through the prompt template, model, and parser modules.

4.2 TF-IDF and GloVe embeddings for Question Answering

The combined use of TF-IDF (Term Frequency-Inverse Document Frequency) vectors and GloVe word embeddings for question answering ensures that both textual content and semantic similarities are considered to yield high quality responses to user queries.

Preprocessing

Before applying TF-IDF and GloVe embeddings, the text data is preprocessed by tokenizing the text into sentences using NLTK's sent_tokenize() function and further tokenizing each sentence into words. Stop words are removed, and words are lemmatized to standardize their form.

Vectorization with TF-IDF

TF-IDF vectors are generated to represent the importance of words in the document relative to a corpus. The TfidfVectorizer from scikit-learn is utilized for this purpose, producing TF-IDF matrices for both the document and query sentences. The TF-IDF vectors capture the significance of each word in the context of the document.

Embedding with GloVe

GloVe word embeddings are employed to represent words in a continuous vector space, capturing their semantic meanings. The pre-trained GloVe embeddings are loaded and used to calculate sentence vectors for both the document and query sentences. This step involves averaging the GloVe embeddings of individual words in each sentence to obtain sentence-level representations.

Selecting Top Relevant Sentences

Cosine similarity is computed between the TF-IDF vectors of the query and document sentences to measure their semantic similarity. Additionally, cosine similarity is calculated between the GloVe sentence vectors of the query and document sentences. The final similarity score for each sentence is a weighted combination of the cosine similarities obtained from TF-IDF and GloVe embeddings, ensuring a balanced consideration of textual content and semantic relationships.
The sentences with the highest similarity scores are identified as the top relevant sentences for each query. These sentences encapsulate the most pertinent information related to the query topic and serve as the basis for generating summaries or answering questions.

4.3 TextRank for Text Summarization

Text summarization using the TextRank algorithm involves several key steps, including data preprocessing, word embeddings, sentence representation, similarity computation, and the TextRank algorithm implementation. Each step plays a crucial role in generating concise and informative summaries from the input text data.

Data Preprocessing

The text is tokenized into sentences and then into words using NLTK. These steps include techniques such as removing punctuation, converting text to lowercase, and handling special characters.

Sentence Representation

The pre-trained GloVe embeddings are loaded. These embeddings provide semantic representations of words, capturing nuanced relationships between them. By associating each word in the vocabulary with a high-dimensional vector, the semantic meaning of words is effectively encoded. With words represented as vectors, the next step involves aggregating these embeddings to represent entire sentences. Each sentence is transformed into a vector representation by combining or averaging the embeddings of its constituent words. This process captures the semantic meaning of the sentence in a continuous vector space, facilitating the comparison and ranking of sentences based on their semantic similarity.

Similarity Computation

Semantic similarity between pairs of sentences is computed using cosine similarity, a metric that quantifies the similarity between two vectors by measuring the cosine of the angle between them. By computing the cosine similarity between pairs of sentence vectors, a similarity matrix is constructed. Each element of this matrix represents the semantic similarity score between two sentences, forming the basis for subsequent steps in the TextRank algorithm.

TextRank Algorithm

TextRank algorithm operates on the constructed similarity matrix to rank sentences based on their importance within the text. This matrix serves as the foundation for building the sentence graph, with sentences as nodes and edges weighed by their corresponding similarity scores. Inspired by the iterative process of PageRank, TextRank iteratively updates the importance scores of sentences based on the scores of their neighboring sentences, converging towards a stable solution.

The final step of the TextRank algorithm involves ranking sentences based on their importance scores, derived from the iterative process. By selecting the top-ranked sentences, TextRank effectively generates a summary that encapsulates the most salient information

from the original text. This extractive summarization approach ensures that the summary retains the essential content while condensing the text into a concise form.

Hashing Techniques

MinHash: MinHash is used to estimate the Jaccard similarity between sets. In this context, it is applied to create a signature matrix representing the similarity between sentences. Each row in the signature matrix corresponds to a hash function, and each column represents a sentence. Using MinHash, the similarity between sentences is computed based on the cosine similarity of their corresponding sentence vectors.

Algorithm 1 MinHash Algorithm

```
function MinHash(S, k)
    H ← initialize empty hash array of size k
    for all h ∈ H do
        h ← ∞                                   ▷ Initialize hash values to infinity
    end for
    for all s ∈ S do                            ▷ Iterate over sets in collection S
        for all i ∈ {1, 2, ..., k} do           ▷ For each hash function
            h_i(s) ← compute hash value of s using hash function i
            if h_i(s) < H[i] then
                H[i] ← h_i(s)                   ▷ Update minimum hash value
            end if
        end for
    end for
    return H
end function
```

SimHash: SimHash is used to estimate similarity between data items. It utilizes random projection vectors to generate a signature for each data item. By mapping each feature of the data item onto a high-dimensional space and then determining the sign of the dot product between the projection vectors and the data's features, SimHash produces a signature that captures the essence of the data's content. In the context of text summarization, SimHash facilitated the computation of the similarity matrix, contributing to the subsequent TextRank algorithm's performance.

Algorithm 2 SimHash Algorithm

```
function SimHash(T, k)
    V ← initialize vector of size 64                ▷ Initialize vector with zeros
    for all t ∈ T do                                ▷ Iterate over tokens in text T
        h ← hash(t)                                 ▷ Compute hash value of token
        b ← binary representation of h
        for i ← 1 to 64 do                          ▷ Iterate over bits in hash
            if b_i == 1 then
                V[i] ← V[i] + 1                     ▷ Increment corresponding vector element
            else
                V[i] ← V[i] − 1                     ▷ Decrement corresponding vector element
            end if
        end for
    end for
    S ← empty string
    for i ← 1 to 64 do                              ▷ Iterate over elements in vector
        if V[i] > 0 then
            S ← S + 1
        else
            S ← S + 0
        end if
    end for
    return S
end function
```

5 Experimental Results

5.1 TextRank Algorithm Performance

The TextRank algorithm efficiently generates extractive summaries from novel data. However, as it focuses solely on the top ten ranked sentences based on their importance and links, the resulting summary may lack a coherent flow, given its extractive nature. Consequently, the ROUGE scores obtained are relatively low, indicating room for improvement in capturing the essence of the original text while maintaining coherence.

5.2 Impact of Hashing Techniques

The incorporation of MinHash and SimHash techniques slightly enhanced the effectiveness of the summarization process. By utilizing MinHash to create a signature matrix for sentence similarity and SimHash for estimating similarity between data items, the summarization algorithm achieved higher accuracy and computational efficiency. These hashing techniques proved instrumental in improving the scalability and performance of the TextRank-based summarization approach.

5.3 ROUGE Scores Analysis

Evaluation of generated summaries (TextRank) against reference summaries (llama2) using ROUGE metrics provided valuable insights into the summarization quality. Comparative analysis revealed that the summaries produced by TextRank, with the integration of MinHash and SimHash, consistently outperformed the simple implementation of the algorithm without hashing. Notably, SimHash demonstrated superior performance in capturing relevant information and preserving text coherence, as indicated by higher Rouge-1 Precision, Recall, and F1-Score compared to Cosine Similarity and MinHash.

Table 1: ROUGE Scores for Each Experiment

Metric	**Cosine Similarity**	**MinHash**	**SimHash**
Rouge-1 Precision	0.218	0.266	0.229
Rouge-1 Recall	0.485	0.286	0.488
Rouge-1 F1-Score	0.301	0.276	0.312
Rouge-L Precision	0.102	0.122	0.111
Rouge-L Recall	0.226	0.131	0.237
Rouge-L F1-Score	0.140	0.126	0.152

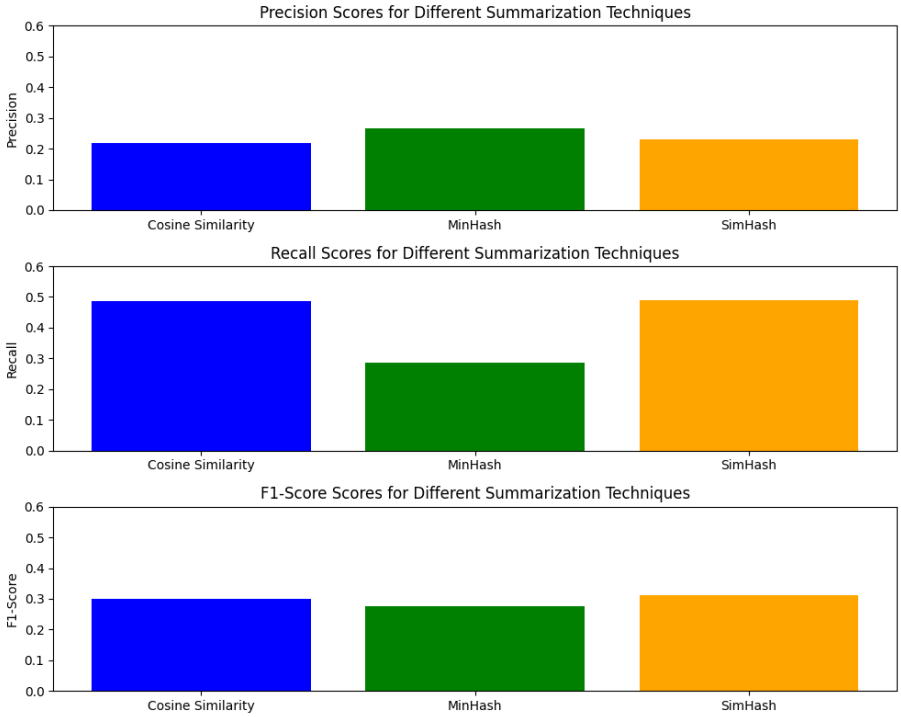

Figure 1: ROUGE Scores for different summarization techniques

6 Challenges

Generating vector embeddings presents a significant computational challenge due to the intricate nature of the process. The transformation of textual data into vector representations using techniques like Ollama Embeddings requires substantial computational resources owing to the complexity of the algorithms involved. Additionally, the process demands extensive memory and processing power, leading to longer execution times and resource-intensive operations.

Running Large Language Models (LLMs) locally poses a formidable challenge owing to the extensive computational requirements involved. LLMs, such as LLaMa2 (7b) and LLaMa3 (8b), undertake numerous computation tasks to calculate the probabilities of generating the next word in a sequence. This computational heavy lifting necessitates access to large-scale GPUs to ensure swift execution and optimal performance. The resource-intensive nature of local LLM execution further exacerbates the challenge, often leading to prolonged processing times and scalability concerns. These runtimes can be reduced by utilizing smaller models or a model with lower quantization (this controls the precision of computations made by the model); however, this will come at the expense of worse performance.

Novel text poses unique challenges for summarization tasks due to its inherent complexity and stylistic characteristics. Novels often contain extensive dialogue between characters, making

it more challenging to discern the semantic meaning of the text. As a result, conventional summarization techniques may struggle to capture the essence of novel text accurately.

7 Conclusion

7.1 Question and Answering

The question answering system provides the top sentences that are related to the question based on similarities between the question statement and the text data. This forms a rudimentary question answering system which performs better when the question is more specific. The end output from the program is sentences from the text as it is, so this system is extremely potent in querying large text documents like text books or transcripts. However, if the question is too vague, the model fails to procure accurate results.

7.2 Text Summarization

The summary generated using Retrieval Augmented Generation with LLaMa 2(7b), captures the events and overall theme of the novels. Large Language models are extremely powerful at understanding semantic relationships within the text provided and creating summaries based on it. The application uses vector embeddings from Ollama through the langchain framework to process the text documents.

The summary generated by TextRank (with and without hashing) captures some key moments and dialogues from the novel chapters, providing a glimpse into the main events in the input text. However, the summary lacks coherence and context. It jumps between different scenes and dialogues without establishing a clear narrative thread. Additionally, some important details and events from the chapter are missing, which could lead to a fragmented understanding of the story for someone unfamiliar with the text.

Overall the summary captures individual moments from the chapter, it falls short of providing a cohesive and comprehensive overview of the narrative. It serves as a starting point for understanding the chapter's content but would benefit from further refinement to improve clarity and coherence.

8 Future Work

There is potential to further enhance the performance of both question answering and text summarization tasks by exploring the integration of more advanced architectures and mechanisms. One promising direction is the incorporation of transformer-based encoder-decoder architectures, which have demonstrated remarkable effectiveness in various natural language processing tasks. By leveraging transformer models and attention mechanisms, it is possible to improve the quality and accuracy of generated summaries and responses. Additionally,

exploring techniques for fine-tuning LLMs could lead to significant advancements in the field, enabling more understanding of textual data.

References

1. llama2 (ollama.com). Retrieved from https://ollama.com/library/llama2
2. Langchain. Retrieveed from https://python.langchain.com/docs/use_cases/question_answering/
3. Automatic Text Summarization Using TextRank Algorithm (analyticsvidhya.com). Retrieved from https://www.analyticsvidhya.com/blog/2018/11/introduction-text-summarizat
4. ROUGE (metric) - Wikipedia. Retrieved from https://en.wikipedia.org/wiki/ROUGE_(metric)
5. GloVe Embeddings. Retrieved from https://nlp.stanford.edu/projects/glove/
6. Carroll, L. (Year). *Title of the book*. Retrieved from https://www.gutenberg.org/cache/epub/11/pg11-images.html
7. Embedding models · Ollama Blog. Retrieved from https://ollama.com/blog/embedding-models
8. MinHash, SimHash algorithms. Retrieved from https://github.com/vicentinileonardo/query-recommendation-system