Proceedings of the Second International Conference on Intelligent Cyber Physical Systems and Internet of Things (ICoICI 2024)
IEEE Xplore Part Number : CFP24VG0-ART ; ISBN : 979-8-3315-4066-1

Enhancing Text Summarization through Parallelization: A TF-IDF Algorithm Approach

Diwesh Chaurasia
School of Computer Science and Engineering
Vellore Institute of Technology
Vellore, Tamil Nadu, India
diwesh.chaurasia2020@vitstudent.ac.in

Mudit Bhatta
School of Computer Science and Engineering
Vellore Institute of Technology
Vellore, Tamil Nadu, India
mudit.bhatta2020@vitstudent.ac.in

Prof. Vimala Devi K
School of Computer Science and Engineering
Vellore Institute of Technology
Vellore, Tamil Nadu, India
vimaladevi.k@vit.ac.in

2024 Second International Conference on Intelligent Cyber Physical Systems and Internet of Things (ICoICI) | 979-8-3315-4066-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICOICI62503.2024.10696641

***Abstract*— Text parallelization is a crucial aspect of natural language processing, aiming to enhance the efficiency of information retrieval and analysis. This project focuses on leveraging the Term Frequency-Inverse Document Frequency (TF-IDF) algorithm to achieve text parallelization. TF-IDF is a widely used technique for information retrieval and document similarity measurement. In this study, we propose a novel approach that harnesses the TF-IDF algorithm to identify and parallelize relevant sections of text, thereby improving the speed and scalability of text processing tasks. We present a comprehensive analysis of the proposed method, evaluating its effectiveness in comparison to traditional approaches. Our results demonstrate the potential of TF-IDF-based text parallelization in optimizing information extraction processes. This research contributes to the ongoing efforts in advancing text processing techniques, particularly in the context of large-scale document analysis.**

Keywords — Text parallelization, term frequency – inverse document frequency algorithm, Information retrieval, Natural language processing, Document similarity, Text processing, Information extraction, Scalability.

I. INTRODUCTION

In today's world of overflowing information, it's crucial to find quick and effective ways to summarize text. This research aims to make summarization faster by using a clever method: running different parts of the summarization job at the same time. We're also diving into a popular tool called TF-IDF, using it in a way that speeds up how we decide which sentences are most important for the summary. Our main goal here is to speed up how we summarize text. We're doing this smartly by getting different parts of the summarization job to work simultaneously. This is super useful, especially when dealing with a lot of text [1]. And we're not stopping there—we're also looking into a tool called TF-IDF. It's like a helper that helps us quickly decide which sentences matter the most for the summary. It's kind of like having a shortcut to find the most important information. Through this implementation, we aim to contribute to the ongoing efforts to make summarizing text quicker and more efficient. By combining the power of parallelization with the smart use of TF-IDF, we're diving into practical ways to handle the overflow of information in today's fast-paced digital world.

II. OBJECTIVES

Our primary goal is to enhance the efficiency of text summarization, making it both better and faster. The objective is to significantly reduce the time required to summarize text, particularly in situations where quick access to information is crucial. Additionally, we are exploring ways to optimize the use of the TF-IDF tool, which plays a key role in determining the sentences that are essential for the summary. Our curiosity extends to the practical applications of these ideas in real-life scenarios. The idea is that by speeding up the summarization process, we can more effectively manage substantial volumes of information. This research is dedicated to unraveling how much improvement and speed enhancement we can achieve in text summarization by concurrently addressing different components of the process.

A. Objective of the work

This research study is fundamentally driven by the overarching goal of elevating the efficiency of text summarization through the strategic implementation of parallelization techniques. The specific emphasis lies in optimizing the TF-IDF algorithm for extraction summarization, a pivotal aspect in distilling essential information from large volumes of text [2]. At the core of our mission is the imperative to significantly reduce the time investment associated with the summarization of textual resources, responding to the increasing demand for timely and responsive information processing.

The strategic application of parallelization processes serves as the prerequisite in achieving the primary research goal. By assigning weights to individual sentences through parallel computation, we aim to accelerate the extraction of pivotal information crucial for creating a concise and informative summary. This strategic approach not only addresses the challenges posed by the ever-expanding volume of textual data but also aligns with the broader objective of

advancing text summarization techniques [3]. Through this focused application of parallelization, this study contributes to the ongoing evolution of efficient and scalable methods for extracting meaningful insights from diverse textual datasets.

B. Software Requirements

To realize these objectives, the project mandates the utilization of essential programming languages, such as C++. Additionally, the integration of specialized libraries dedicated to parallel processing constitutes a critical aspect of the requisite software tools. OpenMP is utilized in this project to streamline the parallelization of operations, specifically in the TF-IDF algorithm and text summarization tasks. The choice of OpenMP is motivated by its user-friendly directives, making it straightforward to integrate parallel programming into existing code. Given the iterative nature of tasks in TF-IDF and summarization, OpenMP's proficiency in parallelizing loops is advantageous. Its implicit threading model simplifies thread management, enhancing code readability and reducing development time. Moreover, OpenMP's cross-platform compatibility ensures seamless execution across diverse systems, while its scalability facilitates efficient utilization of additional processors or cores. Ultimately, OpenMP enables the project to achieve the primary goal of significantly reducing the time required for text summarization, offering simplicity, and compatibility in parallel computing implementation. For the IDE we used Visual Studio Code as the code editor because of it's user friendly environment.

C. Hardware Requirements

The successful implementation of the proposed parallelization technique demands a computer or server equipped with sufficient processing power and memory. This is crucial for managing the computational demands inherent in the parallelized algorithm. Additionally, considering the integration of a distributed computing framework, such as Apache Spark or Hadoop, is advisable to augment the summarization process's efficiency. These frameworks provide scalable and distributed computing environments, potentially accelerating the parallelization of the algorithm across multiple nodes and improving overall system performance [5]. The selection between a standalone system and a distributed computing framework should be based on the specific scalability and efficiency needs of the project.

III. System Design

III.I Algorithms for serial and parallel

execution III.I.I Serial execution

In the serial algorithm, the process begins by reading the input text file and breaking it down into individual words and sentences. The next step involves computing the term frequency (TF) for each word present in the input text. Following this, the inverse document frequency (IDF) is generated for each word by referencing a model file. This is achieved by adding the frequency of each word in the model file to the frequency of that word in the input text.

Once TF and IDF are calculated, the algorithm proceeds to determine a score for each sentence. This score is obtained by summing the TF-IDF scores of the individual words within the sentence. Subsequently, the algorithm identifies the top sentences with the highest scores, forming the summary. The generated summary is then written to an output file. Importantly, the IDF model file is updated to reflect the frequencies of the words in the input text, contributing to the refinement of the summarization process. This step-by-step process defines the serial algorithm's workflow in achieving text summarization.

III.I.II Parallel execution

The parallel execution strategy for the summarization algorithm is structured to capitalize on parallelization techniques, enhancing the efficiency of the process. The initial steps involve including necessary header files and defining regular expressions for subsequent tokenization, setting the groundwork for parallelized processing. Two essential functions, namely tokenize() and sanitize(), are defined to break down the input text into tokens and ensure uniformity by removing non-alphanumeric characters.

The algorithm proceeds with the implementation of functions critical to parallelized computation. termFrequency() is introduced to calculate the term frequency of each word in a vector of tokens, while calculateScore() plays a pivotal role in determining the sentence scores based on both term frequency and inverse document frequency [4]. These functions collectively establish the foundation for efficient parallelized computation.

In the main execution, the number of threads to be employed by OpenMP is specified, enabling parallel processing. The input text is then read from a file, and the tokenization process is parallelized using the defined functions. Subsequently, the algorithm calculates both term frequency and inverse document frequency for the words in the text, leveraging the power of parallel computation to enhance speed and efficiency.

Further steps in the main function encompass dynamic summarization, as the size of the summary is determined based on a specified ratio. The use of a priority queue facilitates the parallelized ranking of sentences according to their scores, ensuring an optimized and speedy summarization process.

The output and update steps involve writing the top-ranked sentences to an output file, preserving the results of parallelized summarization. Additionally, the inverse document frequency in the model file is updated to maintain the accuracy of the summarization model over time.

Finally, the algorithm not only wraps up its execution but also offers valuable insights into its performance [6]. This is accomplished by measuring the time taken for the entire summarization process. This metric serves as a crucial indicator, providing a deeper understanding of the efficiency gains achieved through the deliberate application of parallelization techniques. By strategically leveraging parallelization, the algorithm contributes significantly to the development of a robust and accelerated approach to text summarization, ensuring that the summarization task is not only completed but completed with noteworthy gains in terms of speed and efficiency.

III.II Block Diagram

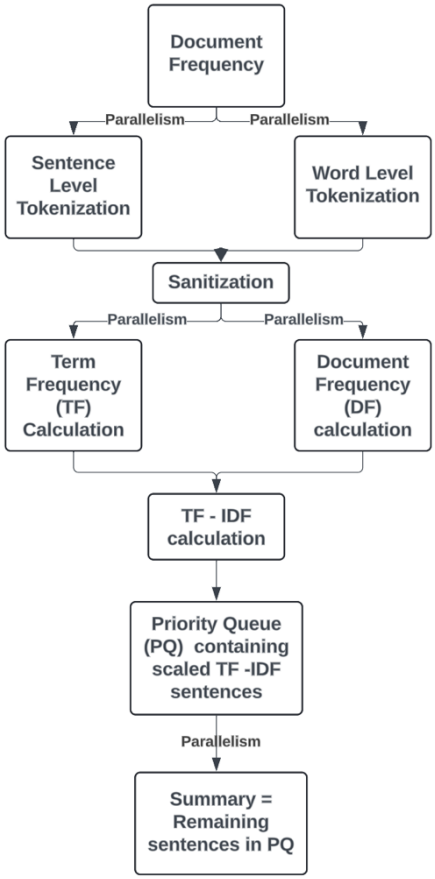

Fig. I. Design of Proposed System

The parallelized text summarization algorithm using the TF-IDF approach efficiently summarizes text documents by employing parallel processing techniques as shown in fig (1). The algorithm begins with necessary preparations, including incorporating the required header files and defining regular expressions for tokenization. Two essential functions, tokenize() and sanitize(), are defined to prepare the text for analysis. The tokenize() function breaks down the input text into individual tokens, while the sanitize() function removes non-alphanumeric characters, ensuring uniformity in the text representation.

The algorithm then introduces crucial functions for parallelized computation: termFrequency() and calculateScore(). The termFrequency() function calculates the frequency of each term in a vector of tokens, while the calculateScore() function determines the score of each sentence based on both term frequency and inverse document frequency (IDF). These functions collectively establish the foundation for efficient parallelized computation.

To identify the most relevant sentences, a dynamic summarization approach is adopted [8], where the size of the summary is determined based on a specified ratio [7]. A priority queue is utilized to efficiently prioritize sentences based on their calculated scores, ensuring an optimized and speedy summarization process.

The algorithm concludes by preserving the results of parallelized summarization by writing the top-ranked sentences to an output file. Additionally, the inverse document frequency in the model file is updated to maintain the accuracy of the summarization model over time.

Finally, the algorithm evaluates its performance by measuring the time taken for the summarization process. This metric serves as a valuable indicator of the efficiency gains achieved through the strategic application of parallelization, contributing to a robust and accelerated text summarization approach.

To achieve high accuracy in text summarization, we begin with clean, well-preprocessed data and optimize feature extraction methods called TF-IDF. Then it is ensured reliability by implementing robust scoring mechanisms and addressing text ambiguity and redundancy effectively. Then it removes non-alphanumeric characters through preprocessing steps like tokenization and sanitization. Finally, by maintaining accuracy with continuous refining the model based on performance metrics and user feedback, it can improve performance by leveraging parallel processing techniques and optimizing computational efficiency throughout the summarization process.

III Proposed System

The summarization method adopted in this research paper utilizes Term Frequency (TF) and Document Frequency (DF) to extract key sentences. The DF data, encompassing document and stop word frequencies, is stored in a dedicated model file. To facilitate sentence extraction, we employ parallelism during tokenization, using multiple threads to tokenize both at the sentence and word levels simultaneously.

Following tokenization, a sanitization process removes punctuation, and we compute the TF for each word using a dictionary or map. Concurrently, DF values are sourced from our pre-existing model file. The TF-IDF score for each word is then calculated as the ratio of TF to DF.

To generate the summary, we employ a priority queue. Each sentence's importance score is determined by aggregating the TF-IDF values of its constituent words, scaled by 100, and added to the queue. Less relevant sentences are progressively removed from the queue, resulting in a final selection of the top k sentences that form the summary.

IV. IMPLEMENTATION

IV.I Description of Modules/Programs

IV.I.I Serial module

The given code is an implementation of a text summarization program using a serial approach. It reads an input file containing a text document, tokenizes the text into words and sentences, calculates the term frequency and inverse document frequency of each word, ranks the sentences based on their score computed by combining the term frequency and inverse document frequency, and outputs

the top-ranked sentences as a summary to an output file. The code consists of the following modules/programs:

a. tokenize(): A function that takes a string and a regular expression as input and returns a vector of tokens obtained by splitting the string using the regular expression.
b. sanitize(): A function that takes a token and removes any non- alphanumeric characters from the end of the token and converts it to lowercase.
c. termFrequency(): A function that takes a vector of tokens and returns a map of each token and its frequency in the document.
d. calculateScore(): A function that takes a sentence, the term frequency map, and the inverse document frequency map as input and calculates the score of the sentence by combining the term frequency and inverse document frequency of each word in the sentence.
e. main(): The main program that reads the input file, tokenizes the text into words and sentences, calculates the term frequency and inverse document frequency of each word, ranks the sentences based on their score, and outputs the top-ranked sentences as a summary to an output file. The program also updates the inverse document frequency map and calculates the time taken for summarization.

IV.I.II Parallel module

The program is a parallel summarization tool that takes an input text file, tokenizes it into words and sentences, calculates the term frequency (tf) and inverse document frequency (idf) of each word, ranks the sentences based on their score, and generates a summary of the input text file. The program is written in C++ and uses the OpenMP library for parallelization. The program consists of the following modules:

a. tokenize: This module takes a string and a regular expression as input and tokenizes the string into a vector of strings based on the regular expression.
b. sanitize: This module takes a string and removes any non- alphanumeric characters from the end of the string and converts it to lowercase.
c. termFrequency: This module takes a vector of strings (words) and calculates the term frequency of each word in the vector. It uses OpenMP to parallelize the loop that iterates over the vector.
d. calculateScore: This module takes a string (sentence) and the tf and idf maps and calculates the score of the sentence based on the tf-idf algorithm. It uses OpenMP to parallelize the loop that iterates over the words in the sentence.
e. main: This module is the main function of the program. It reads the input file, tokenizes the input text into words and sentences, calculates the tf and idf of each word, ranks the sentences based on their score, generates a summary of the input text file, and updates the idf file. It uses OpenMP to parallelize the loops that iterate over the sentences and update the idf file.

IV.II Test cases

The given program is very large for text summarization, it is difficult to provide a set of basic test cases that cover all possible scenarios. However, here are some test cases that can be used to test specific parts/module of the code:

- Test tokenize() function:

Input: "This is a test sentence, which contains some punctuation marks." Expected Output: ["This", "is", "a", "test", "sentence", "which", "contains", "some", "punctuation", "marks"]

- Test sanitize() function:

Input: "This is a test sentence, with some extra characters!@#" Expected Output: "this is a test sentence with some extra characters"

- Test termFrequency() function:

Input: ["this", "is", "a", "test", "sentence", "this", "is", "another", "test",
"sentence"] Expected Output: {"this": 2, "is": 2, "a": 1, "test": 2, "sentence": 2,
"another": 1}

- Test calculateScore() function:

Input: "This is a test sentence.", {"this": 2, "is": 2, "a": 1, "test": 2, "sentence":
2, "another": 1}, {"this": 1, "is": 1, "a": 1, "test": 2, "sentence": 2, "another":
1}
Expected Output: A floating point number representing the score of the sentence.

- Test file input and output:

Input file: "input.txt" containing a long text with multiple sentences. Expected Output: A summary of the input text in the "output.txt" file, with a specific length based on the summary size ratio defined in the code.

The output file should contain coherent sentences that make sense and summarize the input text accurately which is shown in fig (II) and fig(III).

V. Output And Discussion

V.I Execution Snapshots

- *Full text*

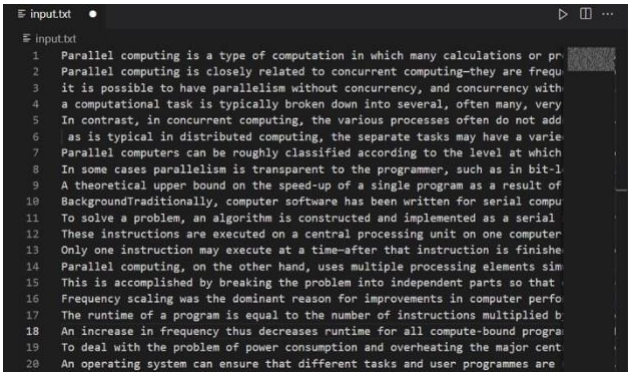

Fig. II. Input text given by the user

- *Summarized Text*

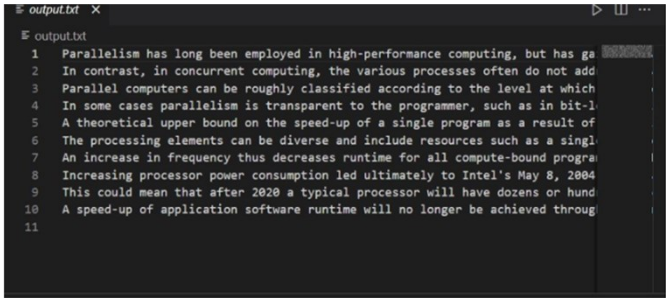

Fig. III. Output after running the program

V.II Output – in terms of performance metrics

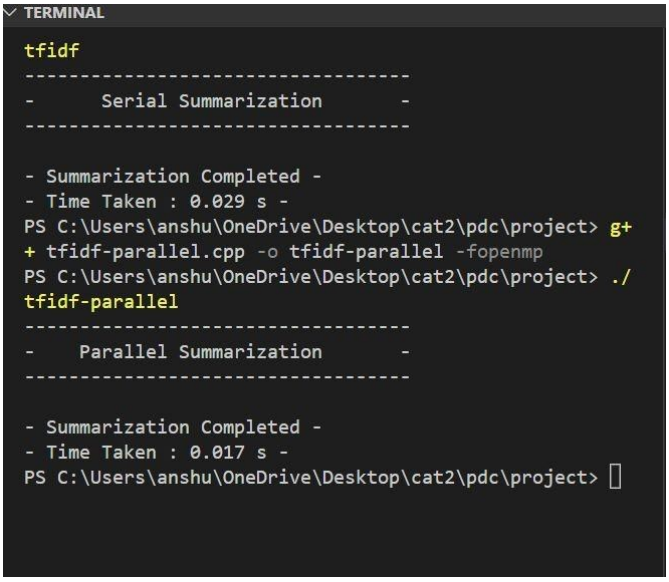

Fig. IV. Time Comparison

	Serial	Parallel
Time (in seconds)	0.029	0.017

Table I. Time Comparison

VI. Comparison Of Obtained Results In Graph And Table Form

VI. I Serial execution

The plots illustrate the relationship between execution time and memory usage for serial and parallel execution units which is shown in fig(V) and fig (VI). Serial execution typically exhibits longer execution times as memory usage increases due to sequential processing, while parallel execution generally results in shorter execution times but higher memory usage due to parallelization overhead and potential data duplication. These observations highlight the trade-off between execution efficiency and memory consumption, emphasizing the importance of selecting the appropriate execution strategy based on performance requirements and resource constraints [9]. Similarly, the

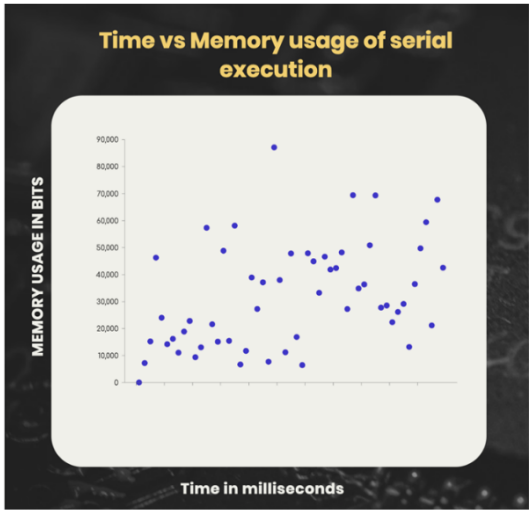

Fig. V. Memory vs time plot of serial execution

VI. II Parallel execution

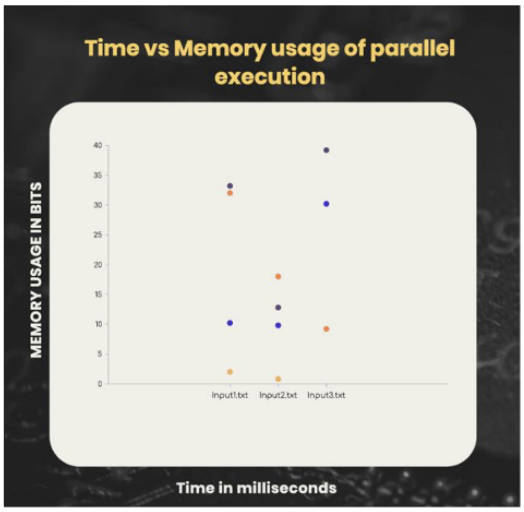

Fig. VI. Memory vs time plot of parallel execution

time caparision is shown in the table (2) which explicitly shows the time taken by each of the execution.

VII. Research Gap

Existing text summarization methods often lack parallel processing, use simplistic scoring, and do not fully utilize document frequency, leading to inefficiencies and less accurate summaries. Our TF-IDF algorithm addresses these issues by integrating parallel tokenization, utilizing a comprehensive model for document frequency, and employing a priority queue for sentence scoring, while also allowing dynamic summary length for better flexibility. This approach utilizes parallel processing for efficiency, incorporates a robust document frequency model for accuracy, and adapts summary lengths flexibly. These innovations aim to overcome existing limitations in text summarization methods, offering a more effective and versatile solution.

VII.I Serial vs Parallel Execution Time for TF-IDF Text Summarization

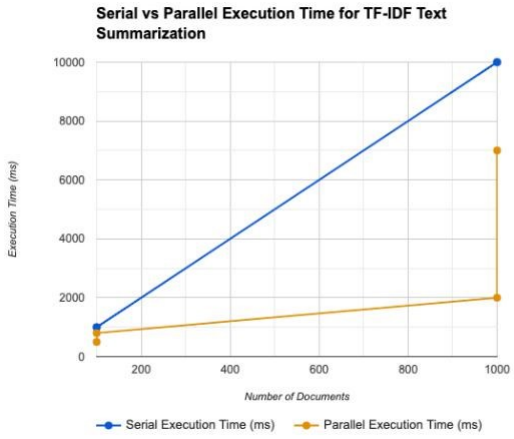

Fig. VII. Serial vs Parallel executoion time comparison

VIII. Limitations

While the strategic application of parallelization has substantially improved the efficiency of text summarization, particularly in reducing the time investment, it is important to note that the benefits may not be as pronounced for smaller text datasets. In instances where the dataset size is limited, the overhead associated with parallelization processes outweigh the gains achieved as shown in fig (VII).

Therefore, a consideration of the trade-off between efficiency gains and computational overhead is vital, highlighting a limitation in the universal applicability of our parallelization approach across all text summarization scenarios. Addressing this challenge may require an adjustment of the parallelization strategy based on the specific characteristics and size of the textual data under consideration.

IX. Conclusion

As observed, the parallel algorithm emerges as the superior choice, particularly evident in scenarios involving larger datasets where its notable speed advantage becomes pronounced. While the time difference is subtle for smaller text datasets, the scalability and efficiency gains of the parallel approach become increasingly evident as the dataset size expands, which. Parallelization offers a more efficient solution compared to the traditional serial implementation. The present algorithm, focusing on extractive summarization, showcased its efficacy in constructing summaries by selecting pivotal sentences from the original text. However, there exists an intriguing avenue for further improvement—the exploration of abstractive summarization techniques. Unlike extractive summarization, abstractive summarization involves generating summaries by paraphrasing and rewording the original text.

Future research should focus on abstractive summarization techniques to generate more context-rich summaries. Exploring advanced NLP models like transformer-based architectures could enhance our understanding of abstractive summarization's feasibility and effectiveness. Integrating state-of-the-art machine learning models and methodologies, such as transfer learning, may advance text summarization. Transfer learning involves fine-tuning pre-trained models on specific tasks, potentially improving summarization capabilities.

References

[1] Khabibulla Madatov, Shukurla Bekchanov, Jernej Vičič (2023). Uzbek text summarization based on TF-IDF.

[2] Dar, Rayees, and A. D. Dileep. "Small, narrow, and parallel recurrent neural networks for sentence representation in extractive text summarization." Journal of Ambient Intelligence and Humanized Computing 13.9 (2022): 4151-4157.

[3] Rohit Shelar; Yash Gujar; Niranjan Pawal; Pratiksha Londhe; Sonali Rangdale, NEWSIFY: - Article Summarization using Natural Language Processing and News Authentication using TF-IDF Vectorizer and Passive Aggressive Classifier (2023)

[4] Krutika Badiger; Sakshi Sonagaj; Sindurani Giraddi; B.M. Reshmi (2021). Kannada Text Summarization.

[6] Wenjun Liu a b, Yuyan Sun b, Bao Yu b, Hailan Wang b, Qingcheng Peng b, Mengshu Hou a c, Huan Guo b, Hai Wang b, Cheng Liu a d (2024). Automatic Text Summarization Method Based on Improved TextRank Algorithm and K-Means Clustering.

[7] Evi Yulianti, Nicholas Pangestu, Meganingrum Arista Jiwanggi (2023). Enhanced TextRank using weighted word embedding for text summarization.

[8] Mian Muhammad Danyal, Sarwar Shah Khan, Muzammil Khan, Subhan Ullah, Muhammad Bilal Ghaffar & Wahab Khan (2024). Sentiment analysis of movie reviews based on NB approaches using TF–IDF and count vectorizer

[9] Nitish Pandey; Sanjog Kumar; Vishwa Ranjan; Masood Ahamed; Abhaya Kumar Sahoo (2020). Analyzing Extractive Text Summarization Techniques and Classification Algorithms: A Comparative Study

[10] Yanni Yang, Yiting Tan, Jintao Min & Zhengwei Huang (2023). Automatic text summarization for government news reports based on multiple features

[11] Al-Amin, Sikder Tahsin, and Carlos Ordonez. "Scalable machine learning on popular analytic languages with parallel data summarization." Big Data Analytics and Knowledge Discovery: 22nd International Conference, DaWaK 2020, Bratislava, Slovakia, September 14–17, 2020, Proceedings 22. Springer International Publishing, 2020.