Technical Report

Web-Based Text Summarization Tool

Duran Duman

January 2025

1. Introduction

In the digital age, information overload has become a significant challenge. People frequently encounter lengthy articles, research papers, and web content, making it difficult to extract key insights efficiently. This issue is particularly prevalent among students, researchers, journalists, and professionals who need to quickly grasp the essence of a document without reading it in its entirety. To address this challenge, we developed a web-based "Summary Tool" that provides users with concise and accurate summaries of textual content.

The "Summary Tool" is designed to help users summarize texts either by manually entering or pasting the content or by providing a Wikipedia URL. The tool then extracts and processes the content to generate a summary using one of three summarization techniques: TF-IDF, Latent Semantic Analysis (LSA), and TextRank.

This project is beneficial for students who need to summarize academic papers, journalists who want to extract the main points of an article, professionals who need quick insights from reports, and casual readers who want to save time while consuming information.

1.1 Project Objectives

The primary goal of this project is to develop an efficient and accessible text summarization tool that can be used by a wide range of individuals. The objectives are,

- Develop a dual-input system supporting both raw text and Wikipedia URLs.
- Implement multiple summarization algorithms for result comparison. (TF-IDF (Baseline), LSA (Semantic Analysis), TextRank (Graph-based))
- Create an intuitive web interface for users.
- Ensure robust text preprocessing for quality summarization.
- Error handling for invalid inputs

2. The Algorithms Used

In this section, we describe the core algorithms used in the application for summarization: **TF-IDF**, **Latent Semantic Analysis (LSA)**, and **TextRank**. These algorithms are designed to extract the most important sentences from the input text, producing a coherent and concise summary.

2.1 TF-IDF (Term Frequency-Inverse Document Frequency)

The **TF-IDF** algorithm is a statistical method used to evaluate how important a word is to a document in a collection or corpus. The **TF-IDF** score is calculated using two main components:

- **Term Frequency (TF)**: The number of times a word appears in the document divided by the total number of words in that document. It reflects the importance of the term within the document.

$$\mathbf{TF} = \frac{\text{Number of times term t appears in a document}}{\text{Total number of terms in the document}}$$

- **Inverse Document Frequency (IDF)**: Measures the importance of the word across all documents. If a word appears in many documents, its IDF score decreases.

$$\mathbf{IDF} = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents containing the term t}}\right)$$

- **TF-IDF Calculation**: The final TF-IDF score is the product of the two:

$$\text{TF}(t) \times \text{IDF}(t)$$

In the code, the **TF-IDF Vectorizer** from ***sklearn.feature_extraction.text*** is used to convert the preprocessed sentences into numerical vectors, which represent the importance of each term in the sentence. These vectors are then used to calculate the similarity between sentences

```
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(processed_sentences)
```

This matrix is then used to score each sentence based on its term importance, and the highest-scoring sentences are selected for the summary.

2.2 Latent Semantic Analysis (LSA)

Latent Semantic Analysis (LSA) is a technique that reduces the dimensionality of the term-document matrix, helping uncover hidden relationships between terms and sentences. The goal of LSA is to identify the most meaningful sentences in the text by analyzing the patterns in the co-occurrence of terms.

The process of LSA includes the following steps:

- **TF-IDF Vectorization**: The text is first converted into a term-document matrix using **TF-IDF**. This matrix represents the importance of terms across all sentences.
- **Singular Value Decomposition (SVD)**: **SVD** is applied to the term-document matrix to reduce its dimensionality. It helps in identifying the underlying structure of the text by decomposing the matrix into three smaller matrices. The most significant singular values (components) are kept, while less important information is discarded.
- **Sentence Selection**: The reduced matrix is used to score sentences. Sentences with higher scores (indicating more important content) are selected for the summary.

In the code, **TruncatedSVD** is used to perform the dimensionality reduction on the TF-IDF matrix.

```
svd = TruncatedSVD(n_components=1)
lsa_matrix = svd.fit_transform(tfidf_matrix)
```

The sentences with the highest importance, determined by the sum of the components in the reduced space, are chosen for inclusion in the summary.

2.3 TextRank

TextRank is a graph-based algorithm for text summarization, inspired by the PageRank algorithm used by Google. TextRank works by building a similarity graph where each node represents a sentence, and the edges between nodes represent the similarity between the sentences. The sentences that are most similar to others are ranked higher.

The TextRank process involves the following steps:

- **Similarity Matrix**: A **cosine similarity** matrix is created by calculating the similarity between all pairs of sentences in the document. Cosine similarity is a measure of similarity between two vectors, where a value closer to 1 indicates high similarity.

  Cosine Similarity = A.B / |A||B|
- **Graph Construction**: The similarity matrix is converted into a graph, where sentences are connected by weighted edges that represent the degree of similarity between them.
- **TextRank Algorithm**: The **TextRank** algorithm is applied to the similarity graph. It assigns a score to each sentence based on its centrality in the graph.

Sentences that are more central (i.e., connected to more important sentences) receive higher scores.

- **Sentence Ranking**: Sentences are ranked based on their scores, and the top-ranking sentences are selected for the summary.

In the code, **NetworkX** is used to build the graph and apply the TextRank algorithm.

```
similarity_matrix = cosine_similarity(tfidf_matrix, tfidf_matrix)
similarity_graph = nx.from_numpy_array(similarity_matrix)
scores = nx.pagerank(similarity_graph)
```

The sentences with the highest scores are selected for inclusion in the final summary.

3. System Architecture

The application follows a client-server architecture with three main components:

- **Web Interface**: Built with Flask using templates (HTML/CSS)
- **Processing Engine**: Python-based NLP pipeline
- **Wikipedia Integration**: API connectivity for content retrieval

3.1 Technology Stack

Layer	Technologies
Frontend	HTML5, CSS3
Backend	Python, Flask
NLP Libraries	NLTK, scikit-learn
Graph Analysis	NetworkX
Wikipedia Integration	Wikipedia-api

3.2 Details of Technologies Used

The project utilizes several technologies and libraries to efficiently process and summarize text data. Below is a breakdown of the key technologies used:

- **Flask**: A lightweight Python web framework used to build the web interface. Flask handles incoming requests, renders templates, and serves the summarization tool to users.
- **HTML5/CSS3**: Standard technologies for building the structure and design of the web pages. HTML5 is used to define the structure of the input fields, and CSS3 is employed for styling the page to ensure it is user-friendly and responsive.

- **Python**: The backend logic is implemented using Python. Python's robust ecosystem of libraries makes it well-suited for text processing and natural language tasks. It forms the backbone of the text summarization process.
- **NLTK (Natural Language Toolkit)**: Used for various text preprocessing tasks, such as tokenization (breaking text into words or sentences), removing stopwords, and stemming (reducing words to their root forms). NLTK plays a critical role in ensuring that the input text is cleaned and processed correctly before summarization.
- **scikit-learn**: This library provides the **TF-IDF Vectorizer**, which is used to convert text data into numerical vectors based on term frequency and inverse document frequency. It is essential for the **TF-IDF** summarization algorithm. Additionally, also used to implement **Latent Semantic Analysis (LSA).**
- **NetworkX**: Used for building the **TextRank** algorithm, which constructs a similarity graph of sentences and applies the PageRank algorithm to rank them based on importance. This library helps in analyzing the relationships between sentences to determine which are the most crucial for the summary.
- **Wikipedia API**: Used to retrieve content directly from Wikipedia pages. This allows users to input a Wikipedia URL, and the tool fetches the relevant text from the page for summarization.

4. How the System Works

The web-based "Summary Tool" offers a streamlined and efficient way for users to summarize lengthy texts into concise summaries. Let's break down the process into a step-by-step explanation of how the system works, from the user's interaction with the web interface to the backend processing and summary generation.

4.1 User Interaction

Once the user navigates to the web interface, they are presented with two main options to input text:

1. **Text Input**: The user can manually type or paste the content they want to summarize into a text area.
2. **Wikipedia URL**: Alternatively, the user can provide the URL of a Wikipedia page, and the system will fetch the content from that page automatically.

The user can then choose which summarization technique they want to use: **TF-IDF**, **LSA**, or **TextRank**. They can specify the number of sentences they want in the

summary. The process flow is simple and intuitive, ensuring that even users with minimal technical knowledge can easily use the tool.

4.2 Backend Processing

After the user submits their input, the following steps are performed by the backend to generate the summary:

1. **Text Retrieval**:

    - If the user provides a Wikipedia URL, the system fetches the page content using the `get_wikipedia_content` function, which uses the Wikipedia API to retrieve the article's title and text. This step involves checking the URL's validity and extracting the relevant page information.
    - If the user inputs raw text, the system proceeds directly to text preprocessing.

2. **Text Preprocessing**: The `preprocess_text` function is called to clean and prepare the text for summarization:

    - **ISBN/ISSN Removal:** Identifies and removes ISBN/ISSN patterns from the text to eliminate unnecessary metadata.
    - **Bracketed Number Removal:** Deletes reference numbers enclosed in brackets (e.g., `[5]`) to ensure cleaner text.
    - **URL, Email, and Special Character Removal:** Strips out URLs, email addresses, and non-alphanumeric characters (except common punctuation) to maintain readability.
    - **Whitespace Normalization:** Reduces multiple spaces to a single space for a consistent text structure.
    - **Sentence Tokenization:** Splits text into individual sentences using NLTK's `sent_tokenize` function.
    - **Short Sentence Removal:** Eliminates sentences with fewer than three words, which are often uninformative.
    - **Stopword Removal:** Filters out common stopwords (e.g., "the", "and", "is") along with custom stopwords (`etc`, `eg`, `ie`, `vs`) to retain meaningful words.
    - **Stemming:** Uses Porter Stemmer to reduce words to their base form, treating variations (e.g., "running" and "ran") as the same root word.
    - **Number and Date Normalization:** Replaces numeric values with `"NUM"` and month names with `"MONTH"` to ensure consistency in text analysis.
    - **Contraction Handling:** Expands common contractions (e.g., `"can't"` → `"can not"`, `"I'm"` → `"I am"`) for improved processing.

These preprocessing steps help structure the text, making it more suitable for summarization and further natural language processing tasks.

3. **Summarization Algorithms**: Depending on the user's selection, one of the following summarization techniques is applied:

    - **TF-IDF (Term Frequency-Inverse Document Frequency)**: The system calculates the TF-IDF score for each sentence to determine its importance within the text. Sentences with higher TF-IDF scores are considered more important and are selected for the summary.
    - **LSA (Latent Semantic Analysis)**: LSA reduces the dimensionality of the term-document matrix using Singular Value Decomposition (SVD), uncovering hidden patterns in the co-occurrence of terms. Sentences with higher importance in the reduced space are selected for the summary.
    - **TextRank**: Inspired by the PageRank algorithm, TextRank builds a similarity graph of sentences. The system calculates cosine similarity between all pairs of sentences to determine their relevance to each other. A PageRank algorithm is applied to this graph, and sentences that are more central (i.e., highly connected to other important sentences) are ranked higher.

4. **Summary Generation**: Once the relevant sentences are selected based on the chosen summarization algorithm, they are combined to create the final summary. If the number of selected sentences exceeds the user's desired summary length, the sentences are ranked, and the top sentences are presented in the summary.

5. Future Improvements

Future improvements for the tool include:

- **Support for Additional Content Sources**: The tool could be enhanced to handle other document types, such as PDFs, Word documents, and scanned images (OCR), to offer greater versatility for different types of input.
- **Incorporating More Advanced Summarization Algorithms**: Adding both **extractive** and **abstractive** summarization methods would improve the tool’s capability to provide more coherent and meaningful summaries. Extractive methods would select key sentences from the text, while abstractive methods could generate entirely new sentences based on the content.
- **Customizable Summarization Parameters**: The inclusion of customizable options would allow users to set preferences such as summary length, content focus (e.g., prioritizing specific sections or topics), or different summarization styles, providing more control over the output.
- **Multilingual Support**: Implementing support for multiple languages would expand the tool’s usability to a global audience, helping users summarize

content in various languages, making it adaptable for a wider range of applications.

6. Conclusion

This web-based summarization tool effectively condenses lengthy texts into concise summaries, allowing users to quickly extract key information. With its current use of techniques like TF-IDF, LSA, and TextRank, it provides flexibility and accuracy in summarization. As the tool evolves, integrating additional algorithms, support for other document types, and features like customizable parameters will further enhance its functionality. By addressing these future improvements, the tool has the potential to meet the diverse needs of users, providing a comprehensive solution for summarizing a wide variety of content.