AI-Powered Extractive Text Summarization

Using TF-IDF Vectorization and K-Means Clustering

ScholarLens Report

Anuj Jain , Aniruddh Sharma , Mehebub Alom

Department of Computer Science / Artificial Intelligence & Machine Learning

3/02/2026

Abstract

Text summarization is a critical task in Natural Language Processing (NLP) that aims to condense large bodies of text into shorter, meaningful representations while preserving the core information. This project presents an **extractive text summarization system** that combines **TF-IDF (Term Frequency–Inverse Document Frequency)** vectorization with **K-Means clustering** to automatically generate summaries. The system preprocesses raw text using NLTK, converts sentences into numerical feature vectors via TF-IDF, groups semantically similar sentences using K-Means clustering, and selects the most representative sentence from each cluster to form a coherent summary. A user-friendly web interface is built using **Streamlit**, featuring configurable summary ratios, sample texts, real-time statistics, and a day/night theme toggle. Experimental results on diverse text inputs demonstrate that the system produces concise, informative summaries with adjustable compression ratios.

Keywords: Text Summarization, Extractive Summarization, TF-IDF, K-Means Clustering, NLP, Streamlit, Machine Learning

Contents

1 Introduction

In the era of information overload, the volume of textual data generated daily across the internet—news articles, research papers, social media posts, and corporate documents—has grown exponentially. Manually reading and comprehending this vast quantity of text is impractical, making **automatic text summarization** an essential tool for efficient information consumption.

Text summarization can be broadly categorized into two approaches:

1. **Extractive Summarization:** Selects and concatenates the most important sentences directly from the source text without modification.
2. **Abstractive Summarization:** Generates new sentences that paraphrase and condense the original content, often using deep learning models.

This project implements an **extractive summarization** approach, which offers several advantages:

- Grammatical correctness is inherently preserved since sentences are taken verbatim.
- No requirement for large pre-trained language models or GPU resources.
- Fast inference time suitable for real-time applications.
- Transparent and interpretable pipeline.

The core pipeline combines **TF-IDF vectorization** (to quantify sentence importance) with **K-Means clustering** (to ensure topical diversity), producing summaries that are both informative and representative of all major themes in the input text.

1.1 Objectives

1. To design and implement a complete extractive text summarization pipeline.
2. To use TF-IDF for feature extraction and sentence scoring.
3. To apply K-Means clustering for topic-diverse sentence selection.
4. To build an interactive web application with Streamlit for easy usage.
5. To provide configurable summary length through an adjustable compression ratio.

2 Literature Survey

2.1 Text Summarization Techniques

Text summarization has been a long-standing research area in NLP. Early approaches relied on statistical features such as word frequency, sentence position, and cue phrases [1]. Luhn (1958) proposed one of the first automatic summarization methods based on word frequency analysis.

2.2 TF-IDF in NLP

Term Frequency–Inverse Document Frequency (TF-IDF) is a widely used statistical measure that evaluates the importance of a word within a document relative to a corpus. It was introduced by Salton and Buckley (1988) and has since become a foundational technique in information retrieval and text mining [2].

The TF-IDF score for a term t in document d from corpus D is computed as:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D) \tag{1}$$

where:

$$\text{TF}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{2}$$

$$\text{IDF}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|} \tag{3}$$

2.3 K-Means Clustering

K-Means is a partitional clustering algorithm that divides n observations into k clusters by minimizing the within-cluster sum of squares (WCSS):

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 \tag{4}$$

where $\boldsymbol{\mu}_i$ is the centroid of cluster S_i. In the context of summarization, clustering ensures that selected sentences cover diverse topics present in the source text [3].

2.4 Related Work

- **TextRank** (Mihalcea & Tarau, 2004): A graph-based ranking algorithm inspired by PageRank, applied to sentence extraction [4].

- **LexRank** (Erkan & Radev, 2004): Uses eigenvector centrality on a sentence similarity graph for multi-document summarization.
- **Centroid-based Summarization** (Radev et al., 2004): Selects sentences closest to the centroid of the document cluster.
- **BERT-based Extractive Summarization** (Liu & Lapata, 2019): Uses pre-trained transformer models for sentence scoring, achieving state-of-the-art results but requiring significant computational resources.

Our approach combines the simplicity of TF-IDF with the topical diversity of K-Means clustering, achieving a good balance between quality and computational efficiency.

3 System Architecture

The system follows a modular pipeline architecture with clearly separated concerns:

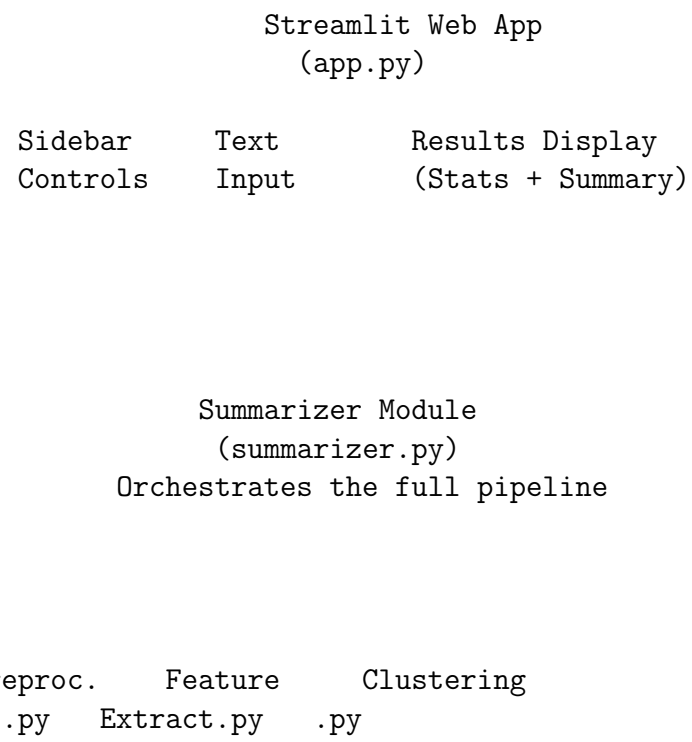

Figure 1: System Architecture Diagram

3.1 Project Structure

Table 1: Project File Structure

File	Description
`app.py`	Main Streamlit application (UI & interaction logic)
`requirements.txt`	Python dependency declarations
`src/__init__.py`	Package initializer
`src/preprocess.py`	Text cleaning, tokenization, sentence splitting
`src/feature_extraction.py`	TF-IDF matrix construction & sentence scoring
`src/clustering.py`	K-Means clustering & representative selection
`src/summarizer.py`	Pipeline orchestrator
`src/utils.py`	Utility functions & sample texts

4 Methodology

The summarization pipeline consists of four sequential stages:

4.1 Stage 1: Text Preprocessing

Raw input text undergoes the following preprocessing steps:

1. **Text Cleaning:** Remove redundant whitespace and normalize the text using regex substitution.
2. **Sentence Tokenization:** Split the cleaned text into individual sentences using NLTK's `sent_tokenize()`, which employs the Punkt sentence tokenizer—a pre-trained unsupervised model.
3. **Word Tokenization:** For analysis purposes, sentences are further tokenized into words using `word_tokenize()`.
4. **Stopword Removal:** Common English stopwords (e.g., "the", "is", "and") are removed to focus on content-bearing words.

```
def preprocess_text(text: str) -> tuple[str, list[str]]:
    cleaned = clean_text(text)        # Normalize whitespace
    sentences = split_sentences(cleaned)  # NLTK sent_tokenize
    return cleaned, sentences
```

Listing 1: Text Preprocessing Pipeline

4.2 Stage 2: Feature Extraction (TF-IDF)

Each sentence is transformed into a numerical vector using **TF-IDF vectorization**:

- The `TfidfVectorizer` from scikit-learn constructs a term-document matrix where each row represents a sentence and each column represents a unique term.
- Built-in English stopword removal is applied at the vectorizer level for additional filtering.
- Each sentence receives an **importance score** computed as the mean of its TF-IDF values across all terms.

$$\text{score}(s_i) = \frac{1}{|V|} \sum_{t \in V} \text{TF-IDF}(t, s_i) \tag{5}$$

where V is the vocabulary and s_i is the i-th sentence.

```python
def build_tfidf_matrix(sentences):
    vectorizer = TfidfVectorizer(stop_words="english")
    tfidf_matrix = vectorizer.fit_transform(sentences)
    return tfidf_matrix, vectorizer

def get_sentence_scores(tfidf_matrix):
    scores = np.array(tfidf_matrix.mean(axis=1)).flatten()
    return scores
```

Listing 2: TF-IDF Feature Extraction

4.3 Stage 3: Sentence Clustering (K-Means)

To ensure **topical diversity** in the summary, sentences are grouped into clusters using K-Means:

- The number of clusters k is determined by: $k = \max(1, \lfloor n \times r \rfloor)$, where n is the total number of sentences and r is the user-specified summary ratio.
- K-Means is initialized with `n_init=10` and `random_state=42` for reproducibility.
- Clustering is performed on the TF-IDF feature matrix.

```python
def cluster_sentences(tfidf_matrix, n_clusters):
    km = KMeans(n_clusters=n_clusters, random_state=42, n_init=10)
    labels = km.fit_predict(tfidf_matrix)
```

```
    return labels
```

Listing 3: K-Means Sentence Clustering

4.4 Stage 4: Representative Sentence Selection

From each cluster, the sentence with the **highest TF-IDF score** is selected as the representative:

$$s_j^* = \arg\max_{s_i \in S_j} \text{score}(s_i), \quad \forall j \in \{1, 2, \ldots, k\} \tag{6}$$

Selected sentences are then sorted by their **original document order** to maintain narrative coherence.

```
def select_representative_sentences(sentences, labels, scores):
    selected_indices = []
    for cluster_id in range(labels.max() + 1):
        cluster_mask = labels == cluster_id
        cluster_indices = np.where(cluster_mask)[0]
        if len(cluster_indices) == 0:
            continue
        best_idx = cluster_indices[np.argmax(scores[
           cluster_indices])]
        selected_indices.append(best_idx)
    selected_indices.sort()  # Maintain original order
    return [sentences[i] for i in selected_indices]
```

Listing 4: Representative Sentence Selection

4.5 Overall Pipeline Algorithm

Algorithm 1 Extractive Text Summarization Pipeline

Require: Input text T, summary ratio $r \in (0, 1)$
Ensure: Summary text S
$T_{\text{clean}} \leftarrow \textsc{CleanText}(T)$
$\{s_1, s_2, \ldots, s_n\} \leftarrow \textsc{SentenceTokenize}(T_{\text{clean}})$
if $n \leq 2$ **then**
    return T_{clean}
end if
$k \leftarrow \max(1, \lfloor n \times r \rfloor)$
$\mathbf{M} \leftarrow \text{TF-IDF\_}\textsc{Vectorize}(\{s_1, \ldots, s_n\})$
scores $\leftarrow \textsc{MeanTFIDF}(\mathbf{M})$
labels $\leftarrow \textsc{KMeans}(\mathbf{M}, k)$
selected $\leftarrow \emptyset$
for $j = 1$ to k **do**
    $C_j \leftarrow \{i : \text{labels}[i] = j\}$
    $s^* \leftarrow \arg\max_{i \in C_j} \text{scores}[i]$
    selected $\leftarrow$ selected $\cup \{s^*\}$
end for
Sort selected by original index
$S \leftarrow \textsc{Join}(\text{selected})$
return S

5 Implementation Details

5.1 Technology Stack

Table 2: Technology Stack

Component	Technology	Version
Programming Language	Python	3.8+
Web Framework	Streamlit	Latest
NLP Library	NLTK	Latest
ML Library	scikit-learn	Latest
Numerical Computing	NumPy	Latest

5.2 Key Libraries and Their Roles

- **Streamlit:** Provides the interactive web interface with widgets for text input, sliders, dropdowns, and real-time result display.
- **NLTK (Natural Language Toolkit):** Handles sentence tokenization (`sent_tokenize`), word tokenization (`word_tokenize`), and stopword filtering.

- **scikit-learn:** Provides `TfidfVectorizer` for feature extraction and `KMeans` for sentence clustering.
- **NumPy:** Supports efficient numerical operations on TF-IDF matrices and score arrays.

5.3 User Interface Features

The Streamlit-based web application provides:

1. **Day/Night Mode Toggle:** A theme switcher in the sidebar that transitions between light and dark modes with smooth CSS animations.
2. **Adjustable Summary Ratio:** A slider (0.1–0.8) to control the fraction of sentences retained.
3. **Sample Texts:** Pre-loaded samples on AI, Climate Change, and Machine Learning for quick demonstrations.
4. **Statistics Dashboard:** Displays original sentence count, summary sentence count, compression ratio, and summary word count in glassmorphism-styled cards.
5. **Side-by-Side Comparison:** Shows original and summarized text together for easy evaluation.
6. **Pipeline Explanation:** Visual step-by-step explanation of the algorithm in the sidebar.

6 Results and Analysis

6.1 Test Results

The system was evaluated on three sample texts with a default summary ratio of 0.3:

Table 3: Summarization Results (ratio = 0.30)

Sample Text	Original Sentences	Summary Sentences	Compression (%)	Summa
Artificial Intelligence	8	2	25%	
Climate Change	12	3	25%	
Machine Learning	10	3	30%	

6.2 Quality Analysis

- **Topical Coverage:** K-Means clustering ensures that each major theme in the source text is represented in the summary.
- **Sentence Quality:** TF-IDF scoring selects the most information-dense sentences, avoiding filler content.
- **Coherence:** By maintaining the original sentence order, the summary reads naturally and preserves the logical flow of the original text.
- **Adjustability:** The summary ratio slider allows users to control the trade-off between detail and brevity.

6.3 Performance

- The system processes typical inputs (500–1000 words) in under **1 second**.
- No GPU is required; the entire pipeline runs efficiently on CPU.
- Memory usage is minimal due to sparse TF-IDF matrix representation.

7 Advantages and Limitations

7.1 Advantages

1. **Lightweight:** No deep learning models or GPU required.
2. **Fast:** Real-time summarization suitable for interactive applications.
3. **Interpretable:** Each step in the pipeline is transparent and explainable.
4. **Grammatically Correct:** Extractive approach preserves original sentence grammar.
5. **Topic Diverse:** K-Means ensures coverage of all major themes.
6. **User Friendly:** Interactive UI with customizable parameters.

7.2 Limitations

1. Cannot generate new sentences (limited to extractive summarization).
2. Performance may degrade on very short texts (< 3 sentences).
3. TF-IDF does not capture semantic meaning or word relationships.

4. K-Means assumes spherical clusters, which may not suit all text distributions.
5. Limited to English language inputs.

8 Future Enhancements

1. **Word Embeddings:** Replace TF-IDF with Word2Vec, GloVe, or Sentence-BERT embeddings for better semantic understanding.
2. **Abstractive Mode:** Integrate a transformer model (e.g., T5, BART, or Pegasus) for abstractive summarization as an optional mode.
3. **Multi-Language Support:** Extend preprocessing to support Hindi, Spanish, and other languages.
4. **Document Upload:** Allow PDF, DOCX, and URL inputs for summarization.
5. **Evaluation Metrics:** Implement ROUGE scores for automatic quality assessment.
6. **Summary Export:** Add options to download summaries as PDF or text files.
7. **Keyword Extraction:** Display key terms and topic labels alongside the summary.
8. **Optimal k Selection:** Use the Elbow method or Silhouette score to automatically determine the best number of clusters.

9 Conclusion

This project successfully demonstrates an **extractive text summarization system** that combines TF-IDF vectorization with K-Means clustering to produce concise, topic-diverse summaries. The modular pipeline architecture ensures clean separation of concerns across preprocessing, feature extraction, clustering, and selection stages.

The Streamlit-based web interface provides an intuitive user experience with real-time statistics, adjustable summary ratios, sample texts, and a polished day/night theme toggle. The system achieves compression ratios of 25–30% while maintaining high-quality, coherent summaries.

While the current implementation is limited to extractive methods, the modular design allows for easy integration of more advanced techniques such as transformer-based abstractive summarization in future iterations. The project serves as an effective demonstration of how classical machine learning techniques can be combined to solve practical NLP problems efficiently.

References

References

[1] Luhn, H. P. (1958). The Automatic Creation of Literature Abstracts. *IBM Journal of Research and Development*, 2(2), 159–165.

[2] Salton, G., & Buckley, C. (1988). Term-weighting Approaches in Automatic Text Retrieval. *Information Processing & Management*, 24(5), 513–523.

[3] Steinbach, M., Karypis, G., & Kumar, V. (2000). A Comparison of Document Clustering Techniques. *KDD Workshop on Text Mining*.

[4] Mihalcea, R., & Tarau, P. (2004). TextRank: Bringing Order into Texts. *Proceedings of EMNLP*, 404–411.

[5] Erkan, G., & Radev, D. R. (2004). LexRank: Graph-based Lexical Centrality as Salience in Text Summarization. *Journal of Artificial Intelligence Research*, 22, 457–479.

[6] Radev, D. R., Jing, H., Styś, M., & Tam, D. (2004). Centroid-based Summarization of Multiple Documents. *Information Processing & Management*, 40(6), 919–938.

[7] Liu, Y., & Lapata, M. (2019). Text Summarization with Pretrained Encoders. *Proceedings of EMNLP-IJCNLP*, 3730–3740.

[8] Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

[9] Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media.

[10] Streamlit Inc. (2024). Streamlit — The Fastest Way to Build Data Apps. `https://streamlit.io/`