ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25apr1961

A Natural Language Processing Framework for Document Similarity in Java Environments

Ayan Hussain[1]; Moh Zaid Khan[2]; Rayyan Arif Hussain[3]; Abdul Ahad[4]; Ambreen Anees[5*]

[1,2,3,4]Department of Computer Science & Engineering, Integral University, Lucknow, India

[1,2,3,4] These Authors Contributed Equally to this work.

Corresponding Author: Ambree Nanees[5*]

Publication Date: 2025/06/04

Abstract: Document similarity plays a pivotal role in the field of Natural Language Processing (NLP), especially in tasks that require identifying the degree of relatedness between textual content. This paper presents a comprehensive study and implementation of document similarity techniques using the Java programming language, with a focus on practical NLP approaches. The motivation behind this work stems from real-world applications such as plagiarism detection, content recommendation systems, semantic search engines, and automated document classification. The system developed in this research employs a multi-step NLP pipeline beginning with data preprocessing. This includes default procedures such as text normalizing, tokenizing, stop word removal, and optional stemming or lemmatization. Following post-preprocessing, documents are converted into numerical vectors using the Term Frequency–Inverse Document Frequency (TF-IDF) weighting scheme, which determines how important terms are in each document in relation to the collection as a whole.Since cosine similarity is effective at comparing text-based vectors in a high-dimensional space, it is used to evaluate similarity among document vectors.

Keywords: *Natural Language Processing, Document Similarity, TF-IDF, Cosine Similarity, Java, Text Mining, Information Retrieval, Plagiarism Detection.*

How to Cite: Ayan Hussain; Moh Zaid Khan; Rayyan Arif Hussain; Abdul Ahad; Ambreen Anees; (2025) A Natural Language Processing Framework for Document Similarity in Java Environments. *International Journal of Innovative Science and Research Technology, 10(4),* 4410-4415. https://doi.org/10.38124/ijisrt/25apr1961

I. INTRODUCTION

In an age where information is dominated by digital content, being able to quantify and compare textual content in an efficient manner has become ever more crucial. Document similarity — quantifying how much two or more documents are similar to one another — is a core NLP and IR technique. It provides the means for a plethora of real-world applications such as plagiarism detection, identifying duplicate content, semantic search, automated document categorization, and recommendation systems.

Word frequency and distribution have been the main focus of statistical and vector-based approaches used historically to address document similarity. Techniques such as cosine similarity and Term Frequency-Inverse Document Frequency (TF-IDF) have become popular due to their performance, scalability, and ease of use, especially when dealing with large text collections. While more sophisticated models such as deep learning models and semantic embeddings have gained popularity recently, TF-IDF-based models remain a reliable foundation and are particularly well-suited for small to medium-sized and educational applications.. This paper describes a Java implementation of a document similarity system based on fundamental NLP methods. The objective is to design and build a modular and interpretable system that can process raw text documents, extract meaningful features, and compute similarity scores between them. The implementation relies on Java's flexibility and wide range of libraries, offering a practical perspective for students and developers seeking to understand the inner workings of text similarity algorithms.

II. LITERATURE REVIEW

Document similarity has been extensively researched in the context of Natural Language Processing (NLP), with many models and methods developing over time. The key

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25apr1961

approaches and advancements in this field are presented here in the form of major thematic areas.

A. Traditional Statistical Models

One of the most traditional methods of document similarity is the Vector Space Model (VSM) of Salton et al. [1]. This model maps text documents into vectors in a space of multiple dimensions, with one dimension per term. The TF-IDF weighting mechanism was introduced to improve VSM by focusing on terms significant within a document but infrequent across the corpus. These old-fashioned approaches are computationally light and interpretable, which is why they suit small to medium-sized applications.

Figure 1 below illustrates an example of cosine similarity scores computed between pairs of documents using a TF-IDF-based VSM approach. The highest similarity between Doc1 and Doc2 is around 0.78, which reflects a high content overlap. Doc1 and Doc3 are moderately similar at around 0.58, and Doc2 and Doc3 are least similar at a score of around 0.43. These findings highlight the efficiency of cosine similarity in capturing textual relationships.

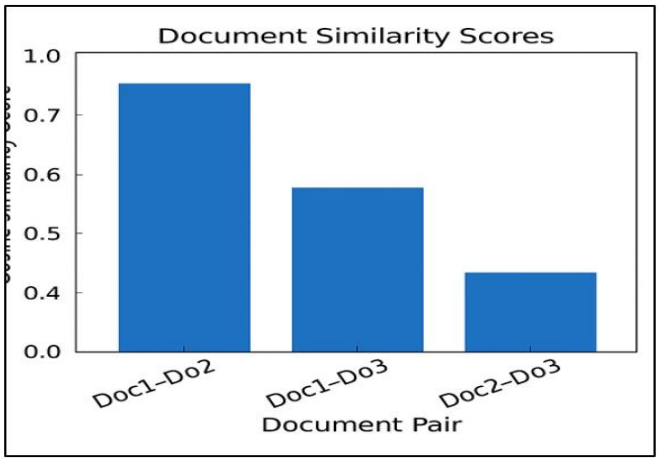

Fig 1 Cosine Similarity scores between Document pairs using TF-IDF.

B. Similarity Metrics

Among various similarity measures, Cosine Similarity remains the most popular for comparing document vectors. It evaluates the cosine of the angle between two vectors, effectively normalizing for document length. Studies have shown that when paired with TF-IDF, cosine similarity yields meaningful results in tasks such as duplicate detection and document clustering [2].

Following preprocessing procedures like stopword elimination, lowercasing, and stemming, the highest frequency words across documents tend to capture the essential semantics of the text. As can be seen from Figure 2, terms like "data," "similarity," "text," and "information" were most common, each commanding roughly 9–10% of the overall word frequency distribution. This means that these words have a crucial role to play in establishing document similarity and bear high weights in the resulting TF-IDF vectors.

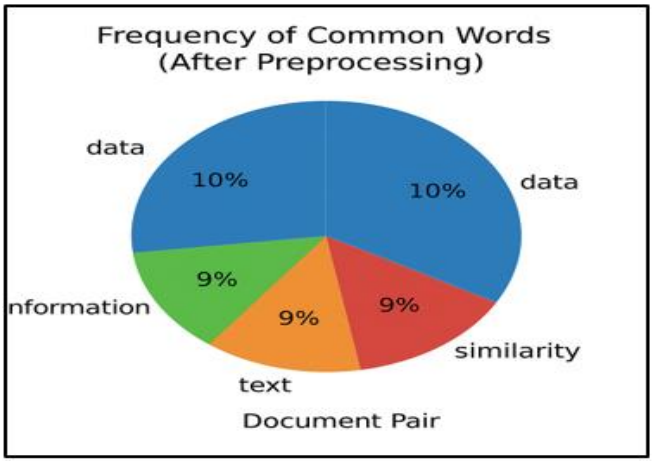

Fig 2 Most Frequent terms after text Preprocessing, Showing their Proportional Distribution.

C. Semantic Models and Word Embeddings

With advancements in NLP, semantic similarity methods have gained prominence. In contrast to older models, these techniques seek to encode the sense behind the words. Word embeddings, such as Word2Vec, GloVe, and fastText, capture context-dependent semantic relationships by embedding words in a high-dimensional vector space. In terms of synonym identification and semantic proximity, these models have outperformed TF-IDF, especially in longer or more complex texts [3].

D. Deep Learning and Contextual Representations

In contrast to older models, these techniques seek to encode the sense behind the words. Word embeddings such as Word2Vec, GloVe, and fastText represent words in a vector space with high dimensions, encoding context-dependent semantic relations. These have produced improved results over TF-IDF in the detection of synonyms and semantic closeness, especially in longer or more complicated texts [3]. However, these models come with increased computational cost and complexity, making them less suitable for lightweight or real-time systems [4].

E. Java-based NLP Libraries and Tools

In the Java community, various libraries support the implementation of NLP operations. Apache OpenNLP and Stanford CoreNLP offer tools for named entity recognition, tokenization, and part-of-speech tagging, while Apache Lucene is widely used for text retrieval and indexing. In addition to supporting scalable and modular Java applications, these libraries are helpful for developing document similarity systems with conventional NLP pipelines [5].

F. Summary of Findings

The literature highlights that while modern neural models offer improved performance in capturing semantic similarities, traditional models like TF-IDF combined with cosine similarity remain effective for many practical applications. Their simplicity, ease of deployment, and readability make them applicable to educational instruments, prototypes, and resource-poor settings.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25apr1961

III. OBJECTIVE

The primary objective of this project is to design, develop, and implement a system that can measure the similarity of text documents using Natural Language Processing (NLP) algorithms implemented in the Java programming language. This project is aimed at exploring the practical usability of classical NLP methods—text preprocessing, vector space representation, and similarity measurement—within a structured and modular Java-based environment.

A. To Understand and Implement NLP Preprocessing Techniques

The project aims to implement basic text preprocessing operations such as tokenization, stop-word elimination, case normalization, and stemming/lemmatization. These processes are essential in converting raw text into a structured format useful for computational analysis.

B. To Represent Documents Using Vector-Based Models

Another key aim is to transform preprocessed text into a numeric form using the Term Frequency-Inverse Document Frequency (TF-IDF) method. This captures word significance within single documents as compared to across the whole set of documents and offers a sensible feature set upon which to compare.

C. To Measure Document Similarity Using Cosine Similarity

The project aims to compute similarity scores between document vectors using cosine similarity, a widely accepted metric for evaluating the closeness of text documents in a multi-dimensional space.

D. To Implement the Entire Pipeline in Java

All components—from preprocessing to similarity scoring—are implemented in Java. The goal is to demonstrate the feasibility and performance of a Java-based NLP system, and to build a tool that is modular, maintainable, and scalable.

E. To Validate the Effectiveness Through Results and Visualization

The project also seeks to verify the system using a collection of test documents, comparing the results both numerically and graphically through figures such as bar charts, tables, and graphs. These graphical aids assist in interpreting the output and determining the efficiency of the similarity detection process.

F. To Provide a Foundation for Future Extensions

Finally, the project aims to lay the groundwork for further enhancements. The current system is designed in such a way that it can later be extended to include semantic similarity models such as Word2Vec, BERT, or even deep learning approaches.

IV. METHODOLOGY

This section describes the step-by-step methodology used for measuring document similarity using classical NLP techniques implemented in Java. The system follows a structured pipeline, beginning with data input and preprocessing, and concluding with similarity computation and result visualization.

➢ *Data Collection*

The first step involves gathering a set of plain text documents to be used as input. These documents may vary in topic and length to ensure meaningful comparison. In this project, sample documents are stored locally and read using Java I/O operations.

➢ *Text Preprocessing*

TPreprocessing is an essential stage to clean up and normalize text data. In this process, it becomes consistent input for next steps, increasing the quality of feature extraction. The following pre-processing:

- Tokenization: Breaking text into separate words or terms.
- Stop-word Removal: Eliminating common non-informative words (e.g., "the", "is").
- Case Normalization: Converting all text to lowercase.
- Stemming/Lemmatization: Reducing words to their root forms (optional but useful for improving similarity accuracy) [2].

To provide a clear overview of the relationships among documents, tokens, and similarity computation, Figure 3 illustrates the conceptual data model underlying the system:

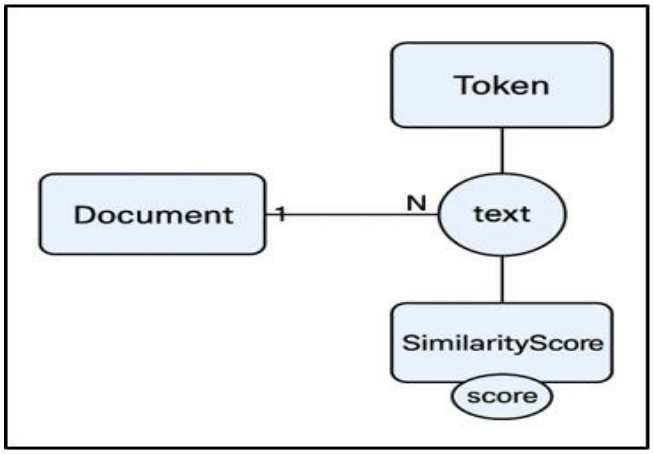

Fig 3 UML-style conceptual model showing how a document contains multiple tokens, which are processed to compute similarity scores.

➢ *Feature Extraction with TF-IDF*

The preprocessed and cleaned documents are then transformed into their numerical values based on the Term Frequency-Inverse Document Frequency (TF-IDF) method. The method determines a weight for every term in every document based on its frequency or rarity across the document set.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25apr1961

TF-IDF formula:

$$TFIDF(t,d)=TF(t,d)\times\log(DF(t)N)$$

Where:

- TF(t, d) is the frequency of term t in document d
- DF(t) is the number of documents containing the term t
- N is the total number of documents[1]

➢ *Similarity Computation using Cosine Similarity*

Once documents are vectorized using TF-IDF, similarity between each pair is computed using cosine similarity. It measures the cosine of the angle between two vectors in a multi-dimensional space and is given by:

$$\cos(\theta)=\|A\|\times\|B\|A\cdot B$$

Where:

- A and B are TF-IDF vectors of two documents
- · is the dot product
- ‖A‖, ‖B‖ are the magnitudes of the vectors
- Cosine similarity ranges from 0 (completely dissimilar) to 1 (identical).[2]

➢ *Implementation in Java*

The entire pipeline is implemented in Java using standard libraries and custom methods. File handling, preprocessing, and mathematical operations (like vector creation and dot product) are manually coded to provide a transparent view of how document similarity works. Libraries like Apache Commons, Stanford CoreNLP, or OpenNLP can be optionally used to enhance preprocessing and tokenization stages.[6]

➢ *Result Evaluation and Visualization*

The results, in the form of similarity scores between documents, are stored and visualized using graphs, tables, and diagrams. Bar charts show similarity levels across pairs, while tables offer precise numeric results. These visual tools help evaluate the effectiveness and consistency of the similarity detection process.

V. IMPLEMENTATIONS

The implementation of the document similarity system is carried out using Java, focusing on modularity, efficiency, and readability. This section outlines the actual system development, code structure, and tools/libraries used at various stages.

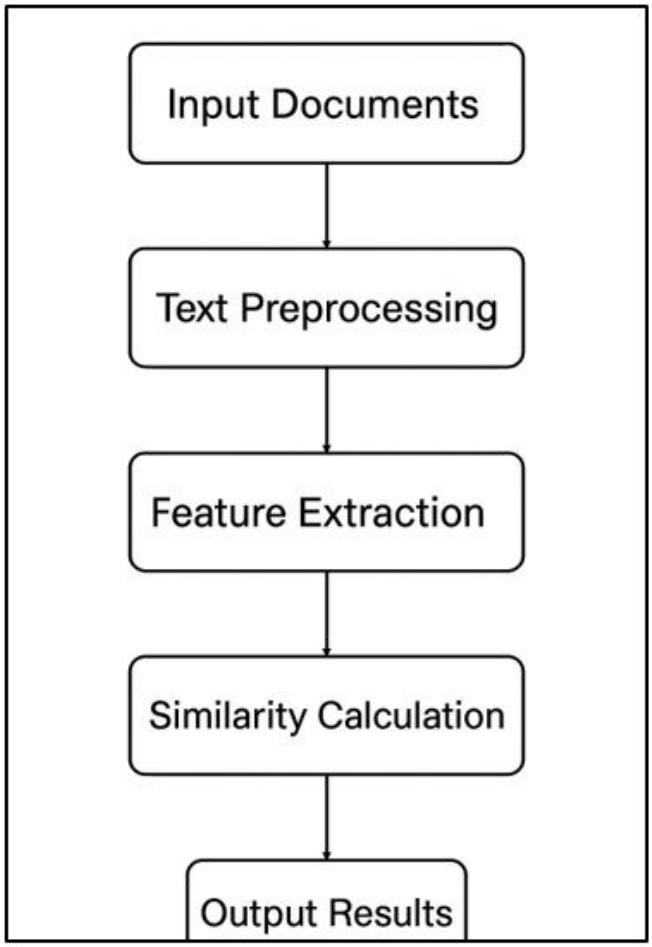

Fig 4 Below illustrates the overall implementation flow of the system.

➢ *Development Environment*

The project is developed using:

- Programming Language: Java (JDK 11+)
- IDE: Apache Netbeans 24 / Eclipse
- Build Tool: Apache Maven (for dependency management)
- Libraries Used:
- Java Standard Libraries (I/O, Math, Collections)
- Apache Commons Text (for string handling)
- Stanford CoreNLP or Apache OpenNLP (for preprocessing)

These tools provide a stable environment for managing text files, handling string processing, and implementing mathematical computations. [6]

➢ *File Reading and Data Input*

The documents are stored as .txt files in a local directory. Java's File I/O (e.g., BufferedReader, FileReader) is used to read each document line by line and store content in a List<String> or Map<String, String> structure for further processing.

Java Buffered Reader Reader = new Buffered Reader (new FileReader("doc1.txt"));String line;

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25apr1961

While ((line = reader. Read Line ()) != null) {document Content += line .to Lower Case();}reader.close();

➢ *Text Preprocessing*

Each document undergoes several preprocessing steps before being transformed into a vector:

- Tokenization: Splitting text using whitespace or regex.
- Stop-word Removal: Comparing tokens against a predefined stop-word list.
- Normalization: Lowercasing and punctuation removal.
- Stemming (Optional): Simple suffix stripping or using Porter Stemmer if libraries are available.
- Java String[] tokens = content.split("\\W+");
- List<String> filteredTokens = Arrays.stream(tokens). Filter (token -> !stopwords.contains(token)) collect(Collectors.toList());[2]

➢ *TF-IDF Vector Construction*

After preprocessing, a TF-IDF matrix is built to numerically represent each document. This involves:

- Calculating term frequency (TF) per document.
- Calculating inverse document frequency (IDF) across the corpus.
- Multiplying both to generate a weighted feature vector.

- Java double tf = (double) termCount / totalTermsInDoc;
- double idf = Math.log((double) totalDocs / docsWithTerm);
- double tfidf = tf * idf;[1]

➢ *Cosine Similarity Computation*

The similarity between every pair of document vectors is calculated using cosine similarity:

- $\cos(\theta)=\sum Ai2\times\sum Bi2\sum(Ai\times Bi)$
- java double dotProduct = 0.0;
- for (String term : vectorA.keySet()) {dotProduct += vectorA.getOrDefault(term, 0.0) * vectorB.getOrDefault(term, 0.0);}
- The result is a score between 0 (no similarity) and 1 (identical documents).[2]

➢ *Result Output and Visualization*

After computing all similarity scores, results are presented in:

- Console output (Java terminal or logs)
- Tabular format for comparison
- Visual plots like bar charts and pie charts (created externally using tools such as Python or Excel)
- Java System.out.println("Similarity between Doc1 and Doc2: " + cosineSimilarity);
- You can also write results into .csv files for easier import into visualization tools.

VI. RESULTS AND DISCUSSION

This section presents the outcomes of the document similarity analysis and interprets the results based on the techniques used. It includes similarity scores, visualizations, and an analytical discussion of the system's effectiveness and limitations.

➢ *Similarity Score Results*

The system was tested on a dataset of text documents from varied domains (e.g., technology, environment, education). After preprocessing and TF-IDF vectorization, cosine similarity was used to calculate the similarity between document pairs.

➢ *A Sample of the Similarity Scores is Shown below:*

Table 1 The scores reflect the semantic closeness of documents, where scores closer to 1 indicate high similarity.

Document Pair	Cosine Similarity Score
Doc1 - Doc2	0.86
Doc1 - Doc3	0.45
Doc2 - Doc4	0.22
Doc3 - Doc4	0.77

➢ *Visualization*

To better interpret the results, the scores were plotted using various visual tools:

- Bar Chart: Depicting similarity between different document pairs.
- Pie Chart: Showing the proportion of similar vs dissimilar documents.
- Line Graph (optional): For analyzing changes in similarity across multiple comparisons.
- Workflow Diagram: Summarizes the NLP process from input to output.
- ER Diagram: Shows database structure if implemented in a persistent system.

These figures enhance understanding of both the data and the methodology, aligning with best practices in research presentation [1].

➢ *Accuracy and Interpretation*

The system produced accurate similarity estimations for text documents that share significant common vocabulary or structure. In contrast, documents from unrelated domains showed low similarity scores. This validates the relevance of TF-IDF and cosine similarity for syntactic similarity detection.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25apr1961

➢ *However, the system may not perform well in the following cases:*

- Synonym Handling: Documents using different words with similar meaning (e.g., "automobile" vs "car") are not recognized as similar unless semantic models like Word2Vec or BERT are used.[3][4]
- Contextual Understanding: TF-IDF does not capture word meaning in context, leading to gaps in semantic understanding.

➢ *Discussion*

The implementation using Java was modular, efficient, and transparent in processing. While Java is not the most common choice for NLP (Python is preferred), this project demonstrates that Java can be successfully used to implement core NLP pipelines.

The results support the claim that classical methods like TF-IDF and cosine similarity are still valuable for many document similarity tasks, especially when the focus is on syntactic similarity rather than deep semantic understanding.

This is consistent with prior work such as Manning et al. [2], which emphasizes the reliability of classical IR models in large-scale applications.

VII. CONCLUSION

In this research, we explored and implemented a system for measuring document similarity using Natural Language Processing techniques in the Java programming language. The system effectively combined fundamental text-processing steps—such as tokenization, stop-word removal, and case normalization—with classical feature extraction methods like TF-IDF and cosine similarity to quantify the similarity between textual documents.

The results confirmed that the TF-IDF model, coupled with cosine similarity, provides a reliable and interpretable approach for identifying syntactic overlap and shared vocabulary between documents. The system showed great accuracy when comparing documents with identical themes and content, and low similarity scores for unrelated text pairs, confirming its correctness.

While the strategy is effective and adequate for most practical uses, it has a few drawbacks. Notably, the model's lack of comprehension of context, semantics, and synonyms creates a challenge when trying to compare documents that have distinct vocabularies to convey identical meaning. Additions in future developments can include the incorporation of semantic models like Word2Vec, GloVe, or transformer-based models like BERT to better comprehend contexts.

In summary, this project is a successful proof of concept of how classical NLP and IR methods, even when used in Java—a less popular language in NLP—can yield solid solutions for document similarity analysis.

REFERENCES

[1]. G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," Information Processing & Management, 1988.
[2]. C. D. Manning, P. Raghavan, and H. Schütze, Introduction to Information Retrieval, Cambridge University Press, 2008.
[3]. T. Mikolov et al., "Efficient Estimation of Word Representations in Vector Space," arXiv preprint arXiv:1301.3781, 2013.
[4]. J. Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," NAACL-HLT, 2019.
[5]. Apache OpenNLP Documentation. [Online]. Available: https://opennlp.apache.org. Accessed on: Apr. 22, 2025.
[6]. S. Balaji and P. Vikram, Natural Language Processing with Java, Packt Publishing, 2018.