

**SAVEETHA SCHOOL OF ENGINEERING**

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES**

# CAPSTONE PROJECT REPORT

**PROJECT TITLE**

RECOGNIZING THE SIMILAR TEXTS WITH COSINE SIMILARITY USING NLTK LIBRARY

# REPORT SUBMITTED BY

192210185 JAIKISHORE.P

192211168 THANVEER AASHIF.N

**REPORT SUBMITTED TO**

Dr. E. MONIKA

# COURSE CODE / COURSE NAME

CSA1369 / THEORY OF COMPUTATION WITH PROBLEM SOLVING

SLOT C

# DATE OF SUBMISSION

11.09.2024

# ABSTRACT

# This project focuses on recognizing similar texts using cosine similarity, a popular method in Natural Language Processing (NLP) to measure the similarity between two textual data samples. The project employs the NLTK library for preprocessing text, including tokenization, stopword removal, and lemmatization, to ensure that texts are normalized before comparison. Texts are then transformed into numerical representations using the Term Frequency-Inverse Document Frequency (TF-IDF) technique, which helps in highlighting the importance of terms in relation to the corpus. Once the text is vectorized, cosine similarity is applied to measure the degree of similarity between the vectors, producing a score between 0 (completely different) and 1 (identical). The system allows users to set a threshold to classify pairs of texts as "similar" or "not similar" based on their similarity score. This project is applicable in various domains such as plagiarism detection, document clustering, and recommendation systems. By utilizing widely accessible tools like NLTK and scikit-learn, this project demonstrates a simple yet effective way to automate text comparison and enhance the analysis of textual data.

# INTRODUCTION

# In the age of digital information, managing and analyzing textual data has become increasingly crucial. One of the fundamental challenges in text analysis is determining the similarity between different pieces of text. This project addresses this challenge by employing cosine similarity, a widely used metric in Natural Language Processing (NLP), to measure how closely related two texts are. Cosine similarity is particularly effective in capturing the contextual similarity between documents by analyzing the angle between their vector representations. By leveraging this approach, the project aims to provide a robust method for recognizing and comparing textual content.

# Text preprocessing is a vital step in text analysis. This project utilizes the Natural Language Toolkit (NLTK) library to preprocess text data, which includes steps such as tokenization, stopword removal, and lemmatization. Tokenization breaks down the text into manageable units, while stopword removal eliminates common but uninformative words. Lemmatization ensures that different forms of a word are standardized to their base form, thus improving the accuracy of the similarity measurement.

# To quantify text similarity, the Term Frequency-Inverse Document Frequency (TF-IDF) method is employed. TF-IDF transforms textual data into numerical vectors, capturing the importance of terms relative to the entire corpus. This transformation allows for a more precise comparison of texts by representing them in a high-dimensional space where cosine similarity can be computed. This method highlights key terms and their significance, making it a valuable tool for text analysis.

# Cosine similarity calculates the angle between two vectors to determine their similarity. A smaller angle indicates higher similarity, while a larger angle suggests dissimilarity. This metric provides a straightforward way to compare text documents and is especially useful in applications such as document clustering, plagiarism detection, and information retrieval. By applying cosine similarity, the project aims to offer an effective solution for comparing and categorizing text data.

# The outcomes of this project have broad implications for various fields that rely on text analysis. In academic settings, it can be used for detecting similarities between research papers and detecting potential plagiarism. In business, it can enhance content recommendation systems by matching user interests with relevant documents. Overall, this project demonstrates how combining preprocessing techniques with similarity metrics can significantly improve the analysis and management of textual data.

# LITERATURE REVIEW

# Text similarity measurement has been a pivotal area of research in Natural Language Processing (NLP). Early approaches relied on simple metrics like Jaccard similarity and Euclidean distance, which compared text based on shared terms or overall distance in a vector space. However, these methods often fell short in capturing the nuanced relationships between words and phrases. The introduction of Term Frequency-Inverse Document Frequency (TF-IDF) marked a significant advancement. TF-IDF, as discussed by Salton and McGill (1983), enhances text representation by weighing terms based on their frequency in individual documents relative to their occurrence in the entire corpus. This approach provides a more balanced representation of text, allowing for improved similarity measurement.

# The application of cosine similarity, as described by Deerwester et al. (1990), offers a robust method for comparing text vectors. Cosine similarity computes the angle between two vectors in a high-dimensional space, thus capturing the contextual similarity between texts. This metric is particularly valuable in information retrieval and document clustering, where understanding the relative closeness of documents is crucial. Cosine similarity’s effectiveness in various NLP tasks has been well-documented, highlighting its utility in scenarios where traditional methods may struggle.

# Recent advancements in NLP have introduced more sophisticated techniques, such as word embeddings and deep learning models. Word2Vec and GloVe, for instance, generate dense vector representations of words based on their context, providing a richer understanding of text semantics. These models, as explored by Mikolov et al. (2013) and Pennington et al. (2014), capture complex relationships between words, surpassing the limitations of TF-IDF. Despite these advancements, cosine similarity remains a fundamental technique due to its simplicity and effectiveness in various applications. The continued relevance of cosine similarity in combination with modern methods underscores its importance in the evolving field of text analysis.

# RESEARCH PLAN

The research plan for "Recognizing the Similar Texts with Cosine Similarity using NLTK Library" is structured into five key phases. The first phase, Project Initiation and Planning, involves defining the project’s scope and objectives, identifying stakeholders, and establishing a comprehensive project plan with timelines and resource allocation. Next, in the Data Collection and Preprocessing phase, relevant textual data is gathered from various sources and processed using NLTK, including steps such as tokenization, stopword removal, and lemmatization. This ensures that the text data is clean and suitable for analysis. The third phase, Development and Implementation, focuses on implementing text vectorization using TF-IDF and calculating cosine similarity scores to measure text similarity. This phase also ensures that the system is scalable and efficient. In the Testing and Evaluation phase, the system undergoes rigorous testing, including unit tests, integration tests, and user acceptance testing, to ensure accuracy and reliability. Finally, the Documentation, Deployment, and Feedback phase involves documenting the development process, preparing for deployment, and gathering user feedback to refine and enhance the system. This structured approach ensures a thorough and effective implementation of text similarity analysis using cosine similarity and NLTK.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| S.NO | DESCRIPTION | 04.09.2024  DAY-01 | 05.09.2024  DAY-02 | 06.09.2024  DAY-03 | 09.09.2024  DAY-04 | 10.09.2024  DAY-05 |
| 1. | Project Initiation and Planning |  |  |  |  |  |
| 2. | Requirement Analysis and Design |  |  |  |  |  |
| 3. | Development and Implementation |  |  |  |  |  |
| 4. | Testing and Refinement |  |  |  |  |  |
| 5. | Documentation, Deployment, and Feedback |  |  |  |  |  |

**Fig. 1 Timeline chart**

**Day 1: Project Initiation and Planning (1 day)**

* Define the scope and objectives of the project, focusing on using cosine similarity to recognize similar texts. Identify key goals such as improving text analysis accuracy and developing a user-friendly comparison tool.
* Conduct initial research on text similarity metrics, including cosine similarity and TF-IDF, and review best practices for text preprocessing and vectorization using NLTK and scikit-learn.
* Identify key stakeholders, including researchers, data scientists, and potential end-users, and establish effective communication channels. Develop a detailed project plan with tasks and milestones for each subsequent phase of the project.

**Day 2: Requirement Analysis and Design (1 day)**

* Gather relevant textual data for analysis, which may include sample texts or datasets from available repositories.
* Implement preprocessing steps using NLTK, including tokenization, stopword removal, and lemmatization, to ensure that text data is clean and ready for similarity analysis.
* Finalize the preprocessing pipeline and verify its effectiveness in standardizing text data for further analysis.

**Day 3: Development and Implementation (2 days)**

* Develop the core functionality of the text similarity system, focusing on implementing TF-IDF vectorization and cosine similarity calculations. Utilize Python libraries such as NLTK and scikit-learn for efficient text processing and similarity measurement.
* Code and test the core functionalities, including text vectorization and similarity scoring. Ensure that the system can handle various text inputs and accurately compute similarity scores.
* Integrate additional features such as threshold-based classification to label text pairs as "similar" or "not similar."

**Day 4: Testing and Refinement (1 day)**

* Conduct comprehensive testing, including unit tests for individual components, integration tests to ensure proper functionality across the system, and user acceptance testing to validate the system’s effectiveness.
* Identify and resolve any bugs or issues discovered during testing, and gather feedback from initial users to address usability concerns and performance issues.
* Refine the system based on feedback and testing results to ensure that it meets the project's objectives and performs reliably.

**Day 5: Documentation, Deployment, and Feedback (1 day)**

# Document the entire development process, including methodologies for text preprocessing, vectorization, and similarity calculation, as well as the testing and evaluation procedures.

# Prepare the system for deployment, ensuring that it is configured for optimal performance and usability.

# Deploy the system in a testing environment for final validation and gather feedback from stakeholders and end-users to assess its effectiveness in recognizing similar texts. Evaluate the project’s success in achieving its objectives and identify areas for future improvement.

# METHODOLOGY

# Step 1: Download and install Python from the official website <https://www.python.org/downloads/>. Choose the version compatible with your operating system.

# Step2: Open the command prompt or terminal and install the necessary Python libraries for the project. Execute the following commands:

**pip install nltk**

**pip install scikit-learn**

# Step 3: Open a Python interpreter or script and import the NLTK library. Run the following commands to download the necessary NLTK data:

**import nltk**

**nltk.download('punkt')**

**nltk.download('stopwords')**

**nltk.download('wordnet')**

# Step 4: Preprocess Text Data

* Collect the text data you want to analyze and store it in a text file or a Python list.
* Create a Python script to preprocess the text using NLTK. Implement functions for tokenization, stopword removal, and lemmatization. Ensure your script saves the preprocessed text for further analysis.

# Step 5: Implement Cosine Similarity Calculation

* In your Python script, implement TF-IDF vectorization using scikit-learn. Write functions to convert the preprocessed text into TF-IDF vectors.
* Calculate cosine similarity between text vectors using scikit-learn's cosine\_similarity function. Ensure your script includes functionalities to compute and display similarity scores.

# Step 6: Test and Validate the System

* Run your Python script to preprocess the text data, compute similarity scores, and display the results.
* Verify the correctness of the similarity calculations by comparing the output with expected results. Ensure that the system handles various text inputs effectively.

# Step 7: Refine and Optimize

* Based on testing results, make any necessary adjustments to improve the accuracy and efficiency of the text similarity analysis.
* Optimize preprocessing and vectorization steps to handle larger datasets and improve performance.

# Step 8: Document and Review

* Document the development process, including the implementation details for preprocessing, vectorization, and similarity calculation.
* Review the results and gather feedback from potential users or stakeholders to assess the effectiveness of the system in recognizing similar texts.

# PYTHON CODE:

# # Import necessary libraries

# import nltk

# from nltk.corpus import stopwords

# from nltk.tokenize import word\_tokenize

# from nltk.stem import WordNetLemmatizer

# import string

# from sklearn.feature\_extraction.text import TfidfVectorizer

# from sklearn.metrics.pairwise import cosine\_similarity

# # Download necessary NLTK data (run this once)

# nltk.download('punkt')

# nltk.download('stopwords')

# nltk.download('wordnet')

# # Initialize lemmatizer

# lemmatizer = WordNetLemmatizer()

# # Function to preprocess text

# def preprocess(text):

# # Lowercasing

# text = text.lower()

# # Remove punctuation

# text = text.translate(str.maketrans('', '', string.punctuation))

# # Tokenization

# tokens = word\_tokenize(text)

# # Remove stopwords

# tokens = [word for word in tokens if word not in stopwords.words('english')]

# # Lemmatization

# tokens = [lemmatizer.lemmatize(word) for word in tokens]

# # Join tokens back into a single string

# return ' '.join(tokens)

# # Function to check similarity and mark texts as "similar" or "not similar"

# def check\_similarity(similarity\_matrix, threshold=0.8):

# n = len(similarity\_matrix)

# for i in range(n):

# for j in range(i + 1, n):

# similarity\_score = similarity\_matrix[i][j]

# if similarity\_score >= threshold:

# print(f"Text {i+1} and Text {j+1} are SIMILAR (Score: {similarity\_score:.2f})")

# else:

# print(f"Text {i+1} and Text {j+1} are NOT SIMILAR (Score: {similarity\_score:.2f})")

# # Sample texts for comparison

# texts = [

# "This is a sample text",

# "Text comparison example",

# "Another text that is somewhat different",

# "This is a sample text again"

# ]

# # Preprocess each text

# preprocessed\_texts = [preprocess(text) for text in texts]

# # Initialize TF-IDF Vectorizer

# vectorizer = TfidfVectorizer()

# # Fit and transform the texts into TF-IDF vectors

# tfidf\_matrix = vectorizer.fit\_transform(preprocessed\_texts)

# # Compute cosine similarity between text vectors

# similarity\_matrix = cosine\_similarity(tfidf\_matrix)

# # Display the similarity matrix

# print("Similarity Matrix:")

# print(similarity\_matrix)

# # Check similarity and mark texts as "similar" or "not similar"

# threshold = 0.8 # You can adjust this threshold value

# check\_similarity(similarity\_matrix, threshold)}

**OUTPUT**

# CONCLUSION

# This project successfully demonstrated how cosine similarity can be utilized to recognize similar texts using the NLTK library and Python. By implementing a systematic approach to text preprocessing, including tokenization, stopword removal, and lemmatization, the project ensured that textual data was accurately prepared for analysis. The use of TF-IDF for vectorization allowed for effective measurement of text similarity, highlighting the relationships between different text samples based on their content.

# The testing and refinement phases confirmed the reliability of the implemented system, with cosine similarity providing accurate and meaningful similarity scores. Feedback from initial users validated the system’s effectiveness in identifying similar texts, confirming that the project met its primary objective. The ability to adjust similarity thresholds and analyze various text inputs further enhanced the system's versatility and applicability.

# In conclusion, this project not only achieved its goal of recognizing similar texts but also established a robust framework for future enhancements. The integration of NLTK and scikit-learn libraries proved effective for text analysis, and the documented methodologies provide a solid foundation for further development. Future work may explore advanced models and techniques to refine and expand the capabilities of text similarity analysis.

# REFERENCES

1. **Salton, G., & McGill, M. J. (1983).** *Introduction to Modern Information Retrieval*. McGraw-Hill Education.
2. **Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990).** *Indexing by Latent Semantic Analysis*. Journal of the American Society for Information Science, 41(6), 391-407.
3. **Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013).** *Distributed Representations of Words and Phrases and their Compositionality*. Proceedings of the 26th International Conference on Neural Information Processing Systems (NIPS 2013), 3111-3119.
4. **Pennington, J., Socher, R., & Manning, C. D. (2014).** *GloVe: Global Vectors for Word Representation*. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014), 1532-1543.
5. **Bird, S., Klein, E., & Loper, E. (2009).** *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly Media.
6. **Scikit-learn Documentation. (n.d.).** *Scikit-learn: Machine Learning in Python*. Retrieved from <https://scikit-learn.org/>
7. **Jurafsky, D., & Martin, J. H. (2021).** *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall.
8. **Sebesta, R. W. (2012).** *Concepts of Programming Languages*. Pearson Education.
9. **Russell, S., & Norvig, P. (2020).** *Artificial Intelligence: A Modern Approach*. Pearson Education.
10. **Chomsky, N. (1957).** *Syntactic Structures*. Mouton de Gruyter