

CSE3086– NoSQL Databases

J Component - Project Report

Review II

PLAGARISM CHECKER

20MIA1136
20MIA1139

Rachana Supriya Nandipati
Rishikesh Reddy Chittedi

MTech CSE with Specialization in Business Analytics

Submitted to

Dr.A. Bhuvaneswari,
Assistant Professor Senior,
SCOPE, VIT, Chennai

School of Computer Science and Engineering



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

September 2023



VIT[®]

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computing Science and Engineering

VIT Chennai

Vandalur - Kelambakkam Road, Chennai - 600 127

FALL SEM 23-24

Worklet details

Programme	M.Tech with Specialization in Business Analytics	
Course Name / Code	NoSQL Databases – CSE3086	
Slot	C2	
Faculty Name	Dr.A. Bhuvaneswari	
J Component	Review 1	
Team Members Name Reg. No	20MIA1136	Rachana Nandipati
	20MIA1139	Rishikesh Reddy

Team Members(s) Contributions – Tentatively planned for implementation:

<i>Worklet Tasks</i>	<i>Contributor's Names</i>
Dataset Collection	Rishikesh Reddy
Preprocessing	Rishikesh Reddy
Architecture/ Model/ Flow diagram	Rishikesh Reddy
Model building (suitable algorithm)	Rishikesh Reddy & Rachana Nandipati
Results – Tables, Graphs	Rachana Nandipati
Technical Report writing	Rachana Nandipati
Presentation preparation	Rachana Nandipati

ABSTRACT

Detecting and preventing plagiarism is crucial to maintaining academic integrity and ensuring the originality of content. The Plagiarism Checker Project aims to provide a robust and efficient solution for detecting plagiarism in text documents using MongoDB as the backend database. This project utilizes MongoDB's adaptable and scalable NoSQL database structure for the storage and organization of an array of text documents. MongoDB's document-oriented model allows for efficient retrieval and analysis of textual data, making it an ideal choice for plagiarism detection systems.

The key components of this project include document storage, similarity analysis, and reporting. Users can submit text documents in the user-friendly interface created for analysis, and the system will compare them against a database of previously stored documents to identify potential instances of plagiarism. The system also provides detailed reports, highlighting similarities. The Plagiarism Checker Project by MongoDB presents a comprehensive solution to address the pressing issue of plagiarism. It harnesses MongoDB's flexible and scalable database architecture, sophisticated similarity analysis algorithms, user-friendly interface, and reporting capabilities to offer an efficient and effective plagiarism detection system. This project plays a crucial role in promoting academic integrity and originality while providing valuable feedback to users to improve their content creation practices.

1. INTRODUCTION

The Plagiarism Checker Project is a solution designed to address the growing concern of plagiarism in various domains by leveraging the capabilities of MongoDB, a NoSQL database. Plagiarism detection is essential to maintain integrity and originality in academia, journalism, and content creation.

Problem Statement

Finding the plagiarized parts of the assignments is a very slow work for the professors. Even with a limited number of texts it relies on the teacher's ability to read and remember every submission. As the process of finding plagiarized parts in assignment is based on the teacher's ability to remember all that he or she has read, the results may be incomplete. Some clear cases of copy and paste may easily be overlooked. And since the workload cannot be shared between multiple assistants. Thus, we are introducing system for checking Plagiarism in assignments.

Objectives

- **Efficient Document Storage:** MongoDB's schema-less design allows for seamless storage and retrieval of text documents. This feature ensures scalability, making it well-suited for handling large and diverse collections of textual data efficiently.
- **Similarity Analysis:** The system incorporates advanced algorithms, including cosine similarity and other relevant metrics, to analyze the similarity between documents. This analysis helps identify potential instances of plagiarism, ensuring accurate results.
- **User-Friendly Interface:** The project offers an intuitive user interface that simplifies the process of submitting documents for analysis. Users receive comprehensive reports highlighting similarities and potential sources, making it user-friendly and accessible.
- **Scalability:** MongoDB's horizontal scaling capabilities enable the system to accommodate a growing number of documents and user demands. As the database accumulates data, it can efficiently manage and query large volumes of textual content.
- **Accuracy:** The primary goal of the project is to deliver highly accurate results in plagiarism detection. It aims to minimize both false positives and false negatives, ensuring that instances of plagiarism are identified correctly.

Challenges

- **Data Preprocessing:** Effective data preprocessing is crucial to remove noise from textual data, such as stop words and formatting inconsistencies. This step is essential to ensure accurate similarity analysis.
- **Data Volume and Scalability:** As the database grows with an increasing number of documents, managing and querying the data efficiently becomes a challenge. Ensuring that MongoDB can scale seamlessly is vital to handle the expanding data volume.
- **User Interface Design:** Designing an intuitive and user-friendly interface is essential for users to interact seamlessly with the plagiarism checker. This requires a deep understanding of user needs and workflows to create an interface that enhances the user experience.

The Plagiarism Checker Project, by MongoDB, offers a robust solution to address plagiarism concerns efficiently. It focuses on accurate similarity analysis, scalability, and a user-friendly interface to provide an effective tool for plagiarism detection across various domains.

2. LITERATURE SURVEY

Sl. no	Title	Author / Journal name / Year	Technique	Result
1	An automated system for plagiarism detection using the internet	Allan Knight, Kevin Almeroth, Bruce Bimber Spring Conference, 2009	Intelligent sentence selection approach	Aiming to reduce query numbers for plagiarism detection. It suggests utilizing pattern recognition techniques to improve identification and further reduce queries.
2	Detecting Text Similarity on a Scalable No-SQL Database Platform	Sergey Butakov, Stepan Murzintsev, Aleksandr Tskhai International Conference on Platform Technology and Service (PlatCon), 2016	MongoDB, scalable "key-value" data structure	It achieves a processing speed of 80ms per KB, suitable for similarity detection tasks, with the potential to process 85,000 documents in 24 hours.

3	Using Natural Language Processing Techniques and Fuzzy-Semantic Similarity for Plagiarism Detection	Deepa Gupta, Vani K, Charan Kamal Singh International Conference on Advances in Computing, Communications and Informatics, 2014	NLP techniques, lemmatization, stop word removal, POS tagging, and fuzzy-semantic similarity measures.	The proposed method, POSPIFS, outperforms others in terms of accuracy and efficiency.
4	High Performance Plagiarism Detection Using Rabin's fingerprint and Adaptive N-gram Methodologies	Karuma Chege, Dr George Okeyo, Dr Richard Rimiru International Journal of Scientific & Engineering Research, 2016	Rabin's fingerprinting scheme	The research explores Rabin's fingerprinting scheme as an efficient replacement for MD5 and SHA-1, with Rabin's fingerprint. The effective n-gram size for plagiarism detection is 4, and the addition of n-gram rolling improves effectiveness by a factor of 3.02.

5	A Plagiarism Detection Technique for Java Program Using Bytecode Analysis	Jeong-Hoon Ji, Gyun Woo, and Hwan-Gue Cho Third International Conference on Convergence and Hybrid Information Technology, 2008	Detection technique for Java programs using bytecode	Achieved meaningful correlations between bytecode and source code similarities. The technique employs a local alignment algorithm for bytecode comparison.
6	Using Microsoft SQL Server platform for plagiarism detection	Vladislav Shcherbinin, Sergey Butakov PAN'09 workshop, 2009	Fingerprinting-based algorithm	The paper presents a plagiarism detection approach using Microsoft SQL Server platform. It employs a fingerprinting-based algorithm and Levenshtein's metric for marking plagiarism.

7	Work in Progress: Developing Arabic Plagiarism Detection Tool for E-Learning Systems	Salha M. Alzahrani, Naomie Salim, Mohammed M. Alsofyani International Association of Computer Science and Information Technology - Spring Conference, 2009	Arabic Plagiarism Detection (APD) Tool	The paper introduces an Arabic Plagiarism Detection (APD) Tool for e-learning systems, enabling automatic detection and highlighting of plagiarism in Arabic documents.
8	An IR-based Approach Utilising Query Expansion for Plagiarism Detection in MEDLINE	Rao Muhammad Adeel Nawab, Mark Stevenson and Paul Clough Journal of computational biology and bioinformatics, april 2015	Information Retrieval (IR)-based approach	(IR) based approach for plagiarism detection, using query expansion. The proposed method outperforms Kullback-Leibler Distance.

9	Automatic Generation of Plagiarism Detection Among Student Programs	<p>Edita Roxas, Nathalie Rose Lim and Natasja Bautista</p> <p>7th International Conference on Information Technology Based Higher Education and Training, 2016</p>	Transformations	<p>The paper presents a plagiarism detection system for programming languages using transformations, providing flexibility and accuracy. Future work includes handling additional programming paradigms and further transformations.</p>
10	Reliable plagiarism detection system based on deep learning approaches	<p>Mohamed A. El-Rashidy, Ramy G. Mohamed, Nawal A. El-Fishawy¹, Marwa A. Shouman¹</p> <p>Springer, 2022</p>	LSTM	<p>An intelligent deep learning system for text plagiarism detection using a new database with 42 features reflecting various text similarities.</p>

3. DATASET AND DATABASE SPECIFIC TOOL TO BE USED (DETAILS)

A plagiarism checker is a tool used to detect similarities between documents, and in your case, you want to use text files as the dataset for plagiarism detection.

1. Data Collection and Storage:

- ✓ Begin by gathering the text documents that need to be checked for plagiarism. These documents can be in the form of text files, each representing a piece of content authored by students or any other source.
- ✓ Next, set up a MongoDB database to store these documents. MongoDB is a NoSQL database that can efficiently store and retrieve text data. You'll create collections within the database to organize your documents.

2. MongoDB Database Setup:

- ✓ Create a new MongoDB database to hold your plagiarism checking data.
- ✓ Within this database, create one or more collections, each representing a group of documents. For instance, you could have a "student_documents" collection to store student essays, papers, or assignments.

3. Importing Data:

- ✓ To populate your MongoDB collection(s), you can write a script or use a MongoDB client tool to insert the text data from your text files into the appropriate collections. Each document in the collection should include a unique identifier (e.g., a document ID) and the text content itself.

4. Integration with Python:

- ✓ Use the PyMongo library to connect your Python application to the MongoDB database. PyMongo provides a Pythonic way to interact with MongoDB.
- ✓ Retrieve the documents from the MongoDB collection(s) as needed for plagiarism checking.

The text content retrieved from MongoDB will be used as the input for your plagiarism detection algorithm. This way we can set up a plagiarism checker that integrates with MongoDB to efficiently manage and analyze a dataset of text documents for plagiarism detection. This system will allow you to maintain a secure and organized repository of documents while performing plagiarism checks as needed.

4. ALGORITHMS / TECHNIQUES DESCRIPTION

Pseudocode

1. Import necessary libraries and modules
2. Define MongoDB connection settings (URI, database name, collection name)
3. Initialize MongoDB client and select the database and collection
4. Attempt to establish a connection to MongoDB
5. Initialize the MongoDB client and select the database and collection
6. Print an error message and exit the program if there's a connection issue
7. Define a function to perform text vectorization using TF-IDF
8. Use scikit-learn's TfidfVectorizer to convert text into TF-IDF vectors
9. Return the TF-IDF vectors as NumPy arrays
10. Define a function to calculate cosine similarity between two documents
11. Use scikit-learn's cosine_similarity to calculate cosine similarity
12. Return the similarity score as a float
13. Initialize an empty dictionary to store plagiarism results
14. Attempt to retrieve student documents from the MongoDB collection
15. Use the collection.find() method to query MongoDB and retrieve student documents
16. Store the retrieved documents in a list
17. Vectorize the text content of student documents
18. Use the vectorize () function to convert text into TF-IDF vectors
19. Iterate through pairs of student documents
20. Avoid comparing a document with itself
21. Calculate the similarity score between the TF-IDF vectors of two documents

22. Use the similarity () function to calculate cosine similarity
23. Convert the plagiarism_results dictionary to a JSON-formatted string
24. Return the JSON string containing the plagiarism detection results
25. If the script is executed directly (not imported), call the check_plagiarism function
26. Print the results

Techniques used

The plagiarism checker will be using Cosine similarity algorithm to find the similarity between the documents from the extracted keywords, and etc.

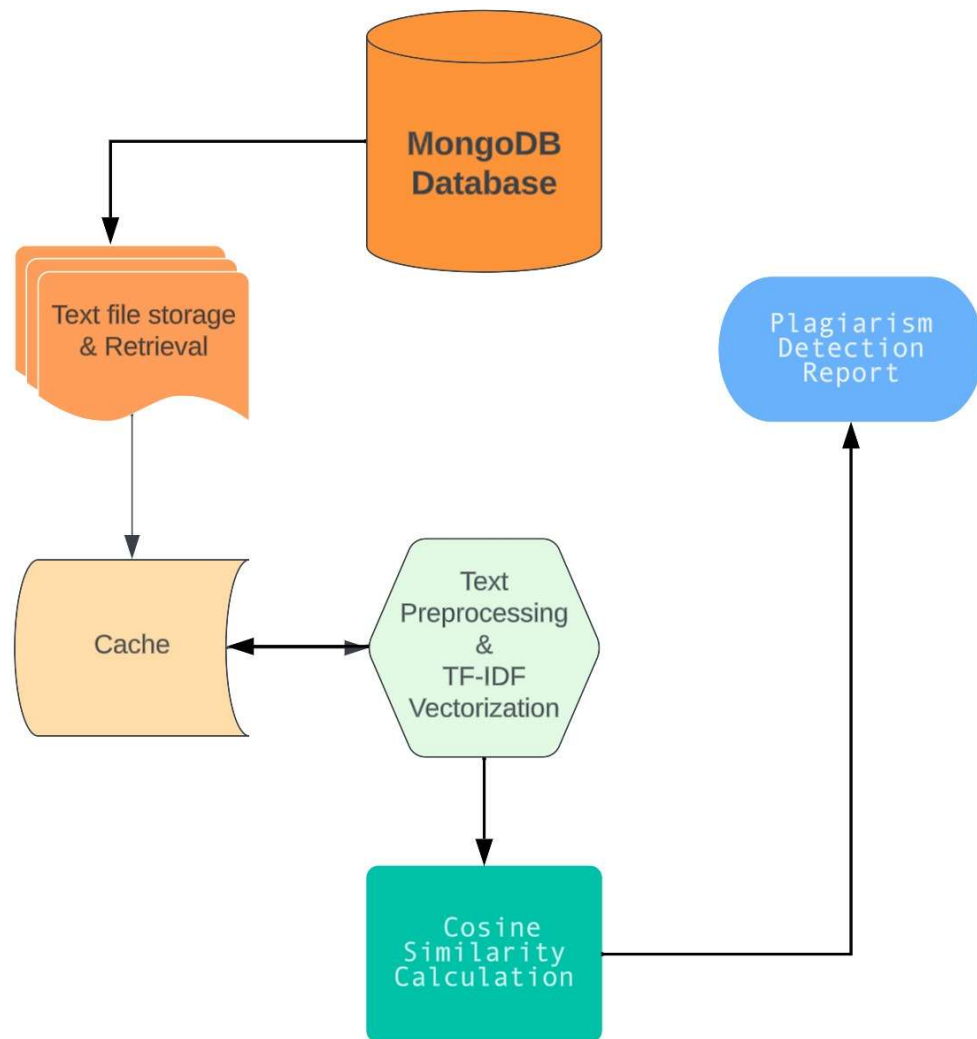
Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. Let x and y be two vectors for comparison. Using the cosine measure as a similarity function, we have

$$\text{sim}(x,y) = (x \cdot y) / (\|x\| \|y\|)$$

where $\|x\|$ is the Euclidean norm of vector $x=(x_1,x_2,\dots,x_p)$, defined as $x_1^2+x_2^2+\dots+x_p^2$. Conceptually, it is the length of the vector. Similarly, $\|y\|$ is the Euclidean norm of vector y . The measure computes the cosine of the angle between vectors x and y . A cosine value of 0 means that the two vectors are at 90 degrees to each other (orthogonal) and have no match. The closer the cosine value to 1, the smaller the angle and the greater the match between vectors.

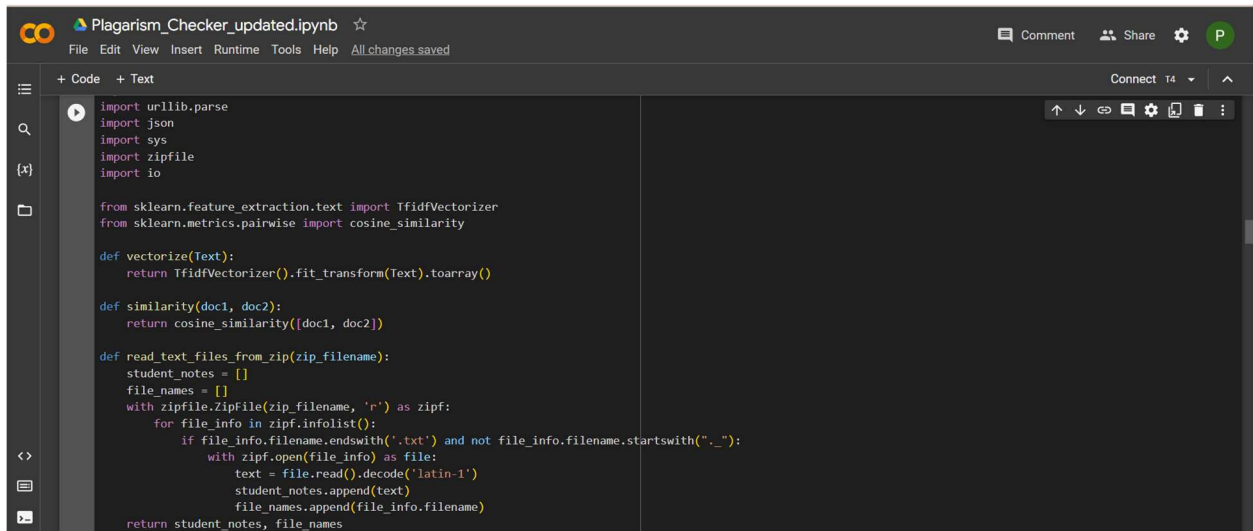
TF-IDF Vectorization (TfidfVectorizer): The program uses the TF-IDF (Term Frequency-Inverse Document Frequency) vectorization technique to convert the text data into numerical feature vectors. This technique assigns weights to words in a document based on their frequency and importance relative to the entire corpus of documents. The TfidfVectorizer from scikit-learn is used for this purpose.

5. ARCHITECTURE DIAGRAM OF THE PROPOSED WORK / SYSTEM DESIGN



6. IMPLEMENTATION

- Retrieve all files
- Get the content for each file
- Check cosine similarity value among each and every file contents
- Conversion of dictionary into JSON format using API
- Display the results



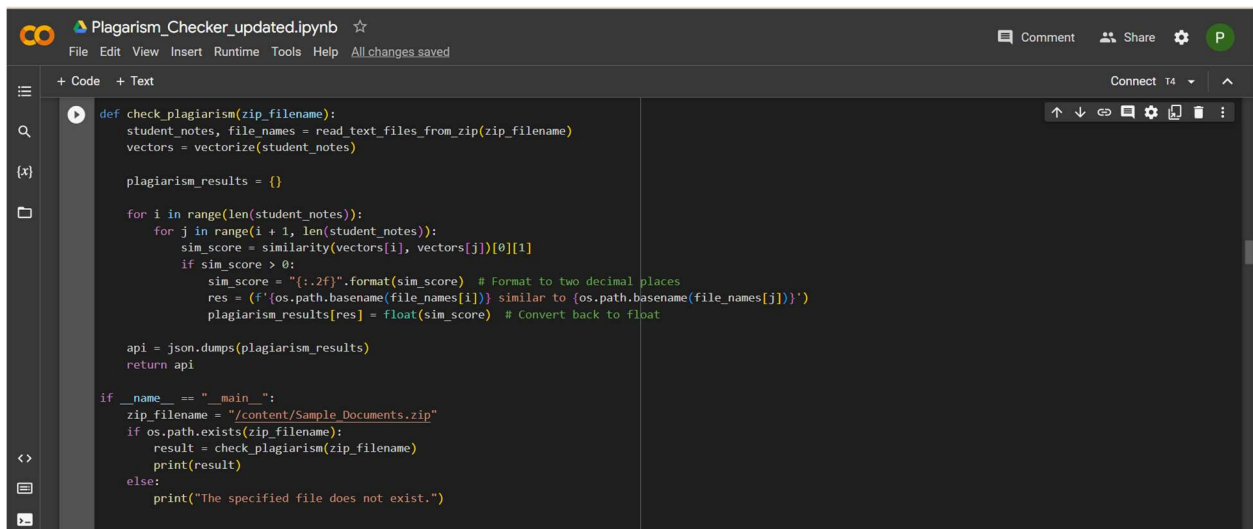
```
Plagarism_Checker_updated.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
import urllib.parse
import json
import sys
import zipfile
import io

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

def vectorize(Text):
    return TfidfVectorizer().fit_transform(Text).toarray()

def similarity(doc1, doc2):
    return cosine_similarity([doc1, doc2])

def read_text_files_from_zip(zip_filename):
    student_notes = []
    file_names = []
    with zipfile.ZipFile(zip_filename, 'r') as zipf:
        for file_info in zipf.infolist():
            if file_info.filename.endswith('.txt') and not file_info.filename.startswith("__"):
                with zipf.open(file_info) as file:
                    text = file.read().decode('latin-1')
                    student_notes.append(text)
                    file_names.append(file_info.filename)
    return student_notes, file_names
```



```
Plagarism_Checker_updated.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
def check_plagiarism(zip_filename):
    student_notes, file_names = read_text_files_from_zip(zip_filename)
    vectors = vectorize(student_notes)

    plagiarism_results = {}

    for i in range(len(student_notes)):
        for j in range(i + 1, len(student_notes)):
            sim_score = similarity(vectors[i], vectors[j])[0][1]
            if sim_score > 0:
                sim_score = "{:.2f}".format(sim_score) # Format to two decimal places
                res = ('{os.path.basename(file_names[i])} similar to {os.path.basename(file_names[j])}')
                plagiarism_results[res] = float(sim_score) # Convert back to float

    api = json.dumps(plagiarism_results)
    return api

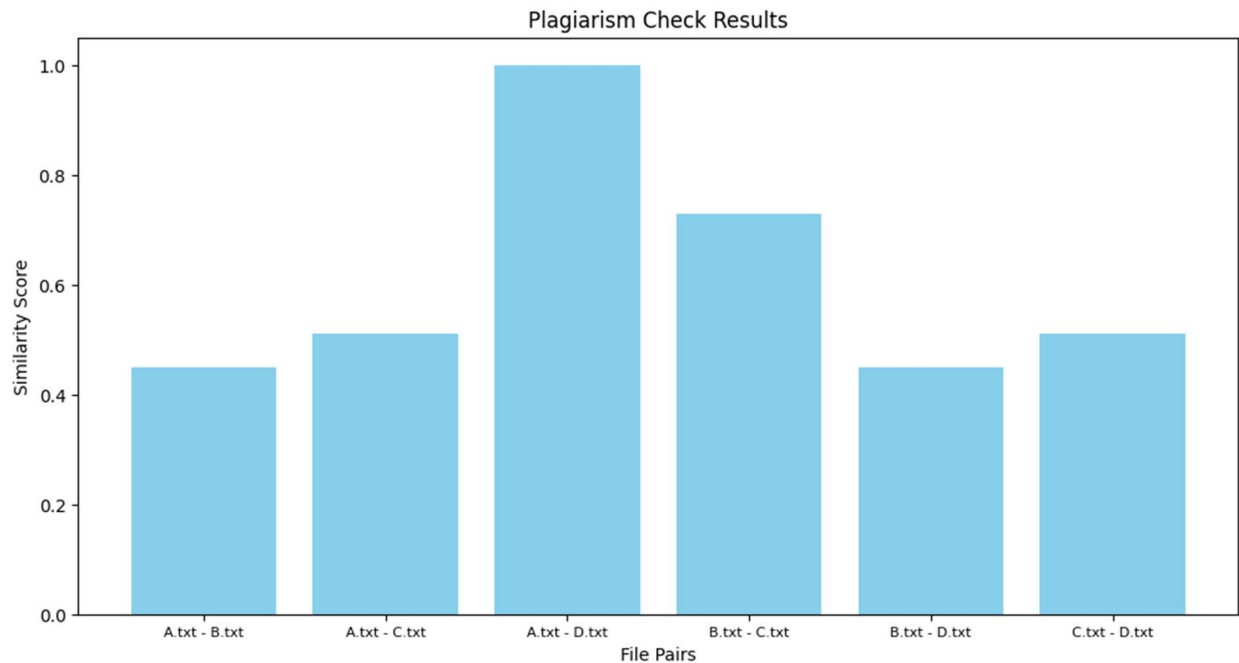
if __name__ == "__main__":
    zip_filename = "/content/Sample Documents.zip"
    if os.path.exists(zip_filename):
        result = check_plagiarism(zip_filename)
        print(result)
    else:
        print("The specified file does not exist.")
```

OUTPUT

```
{"A.txt similar to B.txt": 0.45, "A.txt similar to C.txt": 0.51, "A.txt similar to D.txt": 1.0, "B.txt similar to C.txt": 0.73, "B.txt similar to D.txt": 0.45, "C.txt si
```

```
plt.figure(figsize=(12, 6))
plt.bar(x, similarity_scores, color='skyblue')
plt.xticks(x, [f'{file1} - {file2}' for file1, file2 in zip(file1_names, file2_names)], rotation=0) # Horizontal labels
plt.xlabel('File Pairs')
plt.ylabel('Similarity Score')
plt.title('Plagiarism Check Results')
plt.gca().xaxis.set_major_locator(plt.MultipleLocator(1)) # Set the x-axis tick spacing
plt.xticks(fontsize=8) # Adjust font size for labels

plt.show()
else:
    print("No plagiarism found.")
else:
    print("The specified file does not exist.")
```



7. GITHUB REPOSITORY LINK

https://github.com/RachanaNandipati/Plagiarism_Checker

8. CITATIONS

- [1] Knight, Allan & Almeroth, Kevin & Bimber, Bruce. (2004). An automated system for plagiarism detection using the internet.
- [2] S. Butakov, S. Murzintsev and A. Tskhai, "Detecting Text Similarity on a Scalable No-SQL Database Platform," 2016 International Conference on Platform Technology and Service (PlatCon), Jeju, Korea (South), 2016, pp. 1-5, doi: 10.1109/PlatCon.2016.7456789.
- [3] D. Gupta, K. Vani and C. K. Singh, "Using Natural Language Processing techniques and fuzzy-semantic similarity for automatic external plagiarism detection," 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Delhi, India, 2014, pp. 2694-2699, doi: 10.1109/ICACCI.2014.6968314.
- [4] High Performance Plagiarism Detection Using Rabin's fingerprint and Adaptive N-gram Methodologies Karuma Chege, Dr George Okeyo PhD, Dr Richard Rimiru PhD.
- [5] J. -H. Ji, G. Woo and H. -G. Cho, "A Plagiarism Detection Technique for Java Program Using Bytecode Analysis," 2008 Third International Conference on Convergence and Hybrid Information Technology, Busan, Korea (South), 2008, pp. 1092-1098, doi: 10.1109/ICCIT.2008.267.
- [6] Shcherbinin, Vladislav and Sergey Butakov. "Using Microsoft SQL Server platform for plagiarism detection." (2009).
- [7] S. M. Alzahrani, N. Salim and M. M. Alsofyani, "Work in Progress: Developing Arabic Plagiarism Detection Tool for E-Learning Systems," 2009 International Association of Computer Science and Information Technology - Spring Conference, Singapore, 2009, pp. 105-109, doi: 10.1109/IACSIT-SC.2009.22.
- [8] R. M. A. Nawab, M. Stevenson and P. Clough, "An IR-Based Approach Utilizing Query Expansion for Plagiarism Detection in MEDLINE," in IEEE/ACM Transactions on Computational Biology and Bioinformatics, vol. 14, no. 4, pp. 796-804, 1 July-Aug. 2017, doi: 10.1109/TCBB.2016.2542803.
- [9] R. E. Roxas, N. R. Lim and N. Bautista, "Automatic Generation of Plagiarism Detection Among Student Programs," 2006 7th International Conference on Information Technology Based Higher Education and Training, Ultimo, Australia, 2006, pp. 226-235, doi: 10.1109/ITHET.2006.339768.
- [10] El-Rashidy, M.A., Mohamed, R.G., El-Fishawy, N.A. et al. Reliable plagiarism detection system based on deep learning approaches. Neural Comput & Applic 34, 18837–18858 (2022).