DAYANANDA SAGAR UNIVERSITY

Devarakaggalahalli, Harohalli
Kanakapura Road, Ramanagara - 562112, Karnataka, India

SCHOOL OF ENGINEERING

Bachelor of Technology
in
COMPUTER SCIENCE AND ENGINEERING (ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

Mini Project

PLAGARISM CHECKER USING N-GRAM AND JACCARD SIMILARITY

By
Kasala Bhavana – ENG22AM0153
Pihul Waradkar – ENG22AM0159
Kapu Prathik – ENG22AM0177
Nishat N Shahu – ENG22AM0184

Under the supervision of
Prof. Pradeep Kumar K
Dr. Mary Jasmine
Prof. Mitha Guru
Assistant Professor, Artificial Intelligence &Machine Learning, SOE

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING (ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

SCHOOL OF ENGINEERING,
DAYANANDA SAGAR UNIVERSITY,
BANGALORE

SCHOOL OF ENGINEERING

School of Engineering
Department of Computer Science & Engineering
(Artificial Intelligence and Machine Learning)

Devarakaggalahalli, Harohalli, Kanakapura Road, Ramanagara – 562112

Karnataka, India

CERTIFICATE

This is to certify that the Mini-Project titled **"Plagiarism Checker using n-gram and jaccard similarity"** is carried out by **Kasala Bhavana (ENG22AM0153)**, **N Pihul Waradkar (ENG22AM0159), Kapu Prathik (ENG22AM0177), Nishat N Shahu (ENG22AM0184),** bonafide students third semester of Bachelor of Technology in Computer Science and Engineering (Artificial Intelligence & Machine Learning) at the School of Engineering, Dayananda Sagar University.

Prof.Mitha Guru	**Dr.Mary Jasmine**	**Prof.Pradeep Kumar K**	**Dr.Jayavrinda Vrindavanam**
Assistant Professor	Assistant Professor	Assistant Professor	Chairperson
Dept. of CSE(AI&ML),	Dept. of CSE(AI&ML),	Dept. of CSE(AI&ML),	Dept. of CSE(AI&ML),
School of Engineering	School of Engineering	School of Engineering	School of Engineering
Dayananda Sagar University	Dayananda Sagar University	Dayananda Sagar University	Dayananda Sagar University
Date:	Date:	Date:	Date:

Name of the Examiner **Signature of Examiner**

1.

2.

DECLARATION

We **Kasala Bhavana (ENG22AM0153), Pihul Waradkar (ENG22AM0159), Kapu Prathik (ENG22AM0177), Nishat N Shahu (ENG22AM0184),** are students of third semester B. Tech in **Computer Science and Engineering (Artificial Intelligence & Machine Learning)**, at School of Engineering, **Dayananda Sagar University**, hereby declare that the Mini Project titled **"PLAGIARISM CHECKER"** has been carried out by us and submitted in partial fulfilment for the award of degree in **Bachelor of Technology in Computer Science and Engineering (Artificial Intelligence & Machine Learning)** during the academic year **2023-2024.**

Student	**Signature**
Kasala Bhavana **ENG22AM0153:**	
Pihul Waradkar **ENG22AM0159:**	
Kapu Prathik **ENG22AM0177:**	
Nishat N Shahu **ENG22AM0184:**	

Place: Bangalore
Date:

ACKNOWLEDGEMENT

It is a great pleasure for us to acknowledge the assistance and support of many individuals who have been responsible for the successful completion of this project work.

First, we take this opportunity to express our sincere gratitude to School of Engineering & Technology, Dayananda Sagar University for providing us with a great opportunity to pursue our Bachelor's degree in this institution.

We would like to thank ***Dr. Udaya Kumar Reddy K R, Dean, School of Engineering & Technology, Dayananda Sagar University*** *for his constant encouragement and expert advice.*

It is a matter of immense pleasure to express our sincere thanks to ***Dr.Jayavrinda Vrindavanam, Department Chairman****,* ***Computer Science and Engineering (Artificial Intelligence & Machine learning)****,* ***Dayananda Sagar University,*** *for providing right academic guidance that made our task possible.*

We would like to thank our ***Prof. Pradeep Kumar, Assistant Professor Computer Science and Engineering (Artificial Intelligence & Machine learning)****,* ***Prof. Mitha Guru, Assistant Professor Computer Science and Engineering (Artificial Intelligence & Machine learning)****,****Dr. Mary Jasmine, Assistant Professor Computer Science and Engineering (Artificial Intelligence & Machine learning),****for sparing their valuable time to extend help in every step of our UG Research Project work, which paved the way for smooth progress and fruitful culmination of the research.*

We are also grateful to our family and friends who provided us with every requirement throughout the course.

We would like to thank one and all who directly or indirectly helped us in the Project work.

TABLE OF CONTENTS

Page

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
ML	Machine Learning
OS	Operating System
GPU	Graphics Processing Unit

LIST OF FIGURES

ABSTRACT

In today's digital age, with never ending advances of information technology, plagiarism is becoming one of leading issues that causes wide concern in academics and professional settings.
This project focuses on developing a text similarity detection tool using the Jaccard similarity algorithm. The algorithm employs n-grams, which are subsequences of n words, to quantify the resemblance between two texts. The program reads text content from designated files, computes the Jaccard similarity between the texts, and analyzes the results. If the calculated similarity exceeds a predefined threshold, the texts are flagged as similar, indicating potential plagiarism. This tool provides a simple yet effective means of identifying textual similarities and holds relevance in the domain of plagiarism detection and content analysis.

CHAPTER 1 INTRODUCTION

Due to the digital era, the volume of digital resources has been increasing in the World Wide Web tremendously. Today, creation of such digital resources and their storage and dissemination are simple and straight forward. With the rapid growth of these digital resources, the possibility of copyright violation and plagiarism has also been increasing simultaneously. To address this issue, researchers started working on plagiarism detection in different languages since 1990. It was pioneered by a copy detection method in digital documents [1]. However, the software misuse detection was initiated even much earlier, in 1970 by detecting plagiarism among programs. Since then, a good number of methods and tools have been developed on plagiarism detection which is available online. Plagiarism is the presentation of another's words, work or idea as one's own. There are mainly two types of plagiarisms typically found to occur, such as (1) textual plagiarism and (2) source code plagiarism. Plagiarism may occur within same natural language or it may appear between two or more different languages. Many researchers and software companies are trying to provide an efficient method or tool for plagiarism detection.

1.1 TYPES OF PLAGIARISM

As stated in [2], plagiarism can be defined as an appropriation of the ideas, words, process or results of another person without proper acknowledgment, credit or citation. Plagiarism can appear in a research article or program in following ways:

a. Claiming another person's work as your own.

b. Use of another person's work without giving credit.

c. Majority of someone's contribution as your own, whether credit is given or not.

d. Restructuring the other works and claiming as your own work.

e. Providing wrong acknowledgment of other works in your work.

Plagiarism can appear in different forms in a document, work, production or program. Two basic types of plagiarisms are (a) Textual plagiarism and (b) Source Code plagiarism.

The nearest-neighbour based outlier mining technique is able to detect a plagiarized text segment.(*Active voice*)
A plagiarized text segment is able to detect by the nearest-neighbour based outlier mining technique.(*passive voice*)

(a)

```
Data: First, Last
Result: Sum
while (Last ≠ 0) do
    Sum=First*Last;
    Last=Last-1;
end
```

```
Data: Start, Finish
Result: Total
while (Finish ≠ 0) do
    Total=Start*Finish;
    Finish=Finish-1;
end
```

(b)

Figure 1.1: Examples of (a) Textual (b) Source Code Plagiarism

TEXTUAL PLAGIARISM:

Textual plagiarism is commonly seen in education and research. Figure 1 (a) shows an example of textual plagiarism where entire word-for-word are taken from source without direct quotation.

Textual plagiarism further can be divided into seven sub categories based on its forms and application.

1. Deliberate copy-paste/clone plagiarism: This type of textual plagiarism refers to copying other works and presenting as if your own work with or without acknowledging the original source.

2. Paraphrasing plagiarism: This form of plagiarism can occur on two ways as given below.

• Simple paraphrasing: It refers to use of other idea, words or work, and presenting it in different ways by switching words, changing sentence construction and changing in grammar style.

• Mosaic/Hybrid/patchwork paraphrasing: This form of textual plagiarism generally occurs when you combine multiple research contributions of some others and present it in a different way by changing structure and pattern of sentence, replacing words with synonyms and by applying a different grammar style without citing the source(s).

3. Metaphor plagiarism: Metaphors are used to present others idea in a clear and better manner.

4. Idea plagiarism: Here, idea or solution is borrowed from other source(s) and claiming as your own in a research paper.
5. Self/recycled plagiarism: In this form, an author uses his/her own previous published work in a new research paper for publication.
6. 404 Error / Illegitimate Source plagiarism: Here, an author cites some references but the sources are invalid.
7. Retweet plagiarism: In this form an author cites the reference of proper source but his/her presentation is very similar in the scene of original content wordings, sentence structures and/or grammar usage.

SOURCE CODE PLAGIARISM:

In source code plagiarism, codes written by others are copied or reused or modified or converted a part of codes and claimed as one's own. Figure 1 (b) show the example of source code plagiarism where entire program is represented again in different way by changing syntax.

Typically it is seen, in educational institutes. It can be divided into four subtypes.
1. Manipulation from Vicinity plagiarism: Here, a developer manipulates a program by inserting, deleting, or substituting some codes in an existing program, with or without acknowledging the original source and claiming it as his/her own program.
2. Reordering structure plagiarism: In this type, the developer reorders the statements or functions of a program or changes syntax of a program without referring the original source.
3. No change plagiarism: Here, the developer adds or removes white spaces or comments or indentation of the program and claims the program as his/her own program.
4. Language switching plagiarism: In this type, the developer changes the languages, or a program written in one language is rewritten in another language and declares it as his/her own.

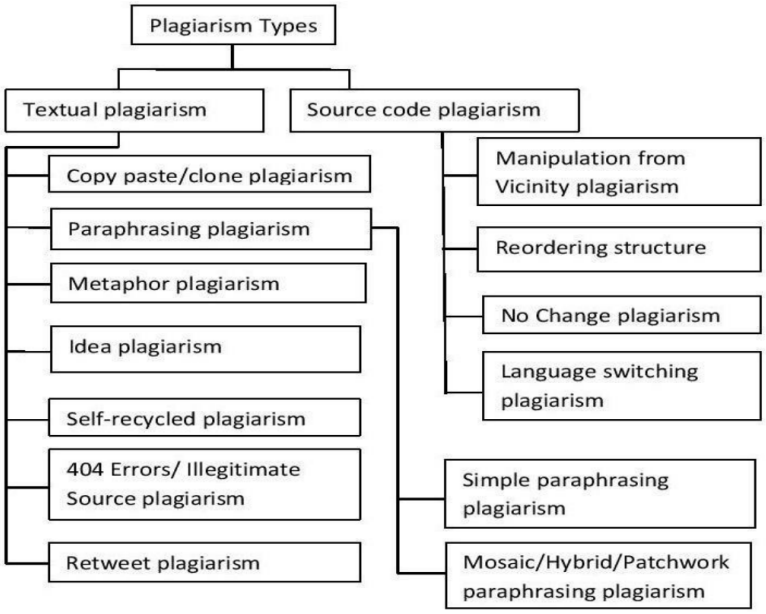

Figure 1.2: Taxonomy of Plagiarism

1.2 SOCIAL IMPACT

The development of this project for identifying plagiarism holds immense scope in academic and professional sectors. It promotes academic integrity and honesty. It helps maintain the credibility of original work, encourages proper citation and attribution, and discourages plagiarism. By detecting and preventing plagiarism, it ensures fairness in academic and professional settings, fosters a culture of originality, and upholds ethical standards. Saves time for educators and researchers by automating the process of checking for plagiarism, allowing them to focus on other aspects of their work.

Chapter 2 PROBLEM DEFINITION

In today's world of information and modern technologies, copying text and manipulating them is not a difficult task. In this modern world where any information can be obtained with the help of digital devices, plagiarism can occur easily. This project aims to address a plagiarism detection system that analyzes textual documents and determines the likelihood of plagiarism based on linguistic patterns. The system employs ML techniques to compare documents and provide insights into potential instances of plagiarism.

This AI-based plagiarism detection system uses ngram and jaccard algorithm. The goal is to create a reliable and efficient system that can accurately detect similarities and provide users with a report highlighting the percentage of the similarity. By doing so, the project aims to identification and categorization of textual content as either original or potentially plagiarized. Successful implementation of this system holds the potential to revolutionize the plagiarism, offering academic integrity and honesty.

Chapter 3 LITERATURE REVIEW

This research [3] says that for plagiarism detection to be successful, one must use machine learning algorithms. The authors were specific with the software tools and algorithms they used for developing the plagiarism detection tool. They have used Anaconda Navigator cloud, Django and node.js for creating a local environment for hosting online assessments and contests. The dataset is obtained from the user and is used for testing and for training they use from the inbuilt libraries. They worked on normalization using Natural Language Processing (NLP) and tokenization for the pre-processing. They focused on the cosine similarity and N-gram algorithms for detecting plagiarism. However, they were not able to scrap the data from the web content and are working on it.

Michael Duracik et al [4] have published the paper to detect the fraudulent activity done by the students in submitting computer programs. The author researched tools like JPlag, MOSS, Plaggie and designed an anti-plagiarism system based on that. The input source code is processed with abstract syntax tree (AST) and vectorization is done underlying the concepts of Deckard Algorithm. They have designed an algorithm for vectorization to add multiple vectors. Post this, the vectors are sent for clustering, where apart from incremental algorithm they have also optimised the K-means algorithm. The plagiarism can be detected once the merging between the matching vectors takes place. For the best efficiency this method is compared with MOSS and JPlag systems and the results are noted. They faced few problems in bringing out the normalization of inputs and bringing out an efficient algorithm for annotation of non-significant code.

Mining repository is one of the techniques which is employed in the project and the following paper by Thai-Bao Do et.al [5] highlights more on the same. The paper proposes a method to detect forks and duplicates in the repository and also checks any correlation between the forking patterns, software health, risks and success indicators. There are more and more data which are being pushed into version control systems like GitHub, GitLab, Bitbucket, PyPI for Python and Debian for open-source software and the paper talks about the method which can be used to extract the metadata from the repositories hosted on platforms like GitHub, GitLab and Bitbucket. The study is done on Software Heritage

dataset which consists of more than three million software repositories from many version control systems. The data extracted from the dataset is stored which can be used for future investigations the approach used by the authors work well and shows possible correlation between the metrics. But there is no concrete conclusion on the relationship which is also a part of their future works.

In this paper [6] the accuracy of detecting plagiarism is high by using the xgboost model with Support Vector Machine (SVM). The author says that they focused on the machine learning technique by working on feature-based extraction done by recurrent neural networks. The xgboost algorithm is trained with the features extracted and the results can be used with Support Vector Machine (SVM) to detect plagiarism. With the advantage of high accuracy, they are still working on incorporating compiler-based features.

This paper by Huang Quibo et al [7] describes a method to extract features from submitted programs and uses a combination of Random Forest algorithm and Gradient Boosting Decision Tree. The authors propose two algorithms. Algorithm 1: A Similarity Degree Threshold (SRT) is set along with a Top limit. If the similarity (sim) of the programs is less than SRT then the program is not plagiarized. If sim is greater than Top limit then the program is suspected to be plagiarised and the student is asked to confirm if they want to submit the assignment. If they confirm then the program is sent for further review to the course instructor. If they decline, they system assumes that the program was plagiarised. If the sim value is between SRT and Top limit then some more features need to be extracted and the program is evaluated using the Random Forest and Gradient Boost Decision tree models to calculate plagiarism suspect level. Algorithm 2: The authors constructed a Random Forest containing multiple decision trees using entropy as criteria for classification and a Gradient boosting decision tree where each tree is a regression tree. Experiments show that algorithm 2 achieved a greater accuracy compared to algorithm 1 and the accuracy rate can reach up to 95.9%.

Tahaei and Noelle [8] have presented an approach which doesn't require the potential plagiarism sources in order to compare similarity, instead assumes an online system which allows for submission of multiple solutions for a single exercise, providing formative

feedback with each submission. It compares pairs of submissions by an individual student, for similarity. They have used the ‘diff’ algorithm, which gives the minimum number of additions or deletions needed to transform one file into another. This was taken as a submission difference between two consecutive submissions, a vector of n – 1 dimensionality is created from the submission differences of each pair of consecutive submissions, starting from the submission difference between first and second submissions. Several features such as number of submissions; average, maximum, minimum and last differences were considered. For each examined subset of features, logistic regression was performed, identifying logistic sigmoid parameters which maximised plagiarism classification accuracy over the training set. The parameters were then used to calculate a plagiarism probability score. The results were evaluated against data collected from actual students enrolled in an undergraduate computer programming class at a research university. Though it couldn’t perfectly classify plagiarism cases, there was strong correlation between these scores and actual cases of plagiarism. However, this method would fail if a student makes a single submission and also if the students (who intend to plagiarise) are aware of the features based on which plagiarism will be detected, as they will try to bypass detection.

Chapter 4 PROJECT DESCRIPTION

4.1 PROPOSED DESIGN

This project aims to develop a tool that measures the similarity between two texts using the Jaccard similarity coefficient based on n-grams. The project addresses the need for an efficient and reliable method to identify potential plagiarism or significant similarity in textual content.

The project focuses on detecting similarities between texts using the Jaccard similarity algorithm with a specified n-gram size. The tool is designed for general textual content and is not limited to specific domains.

The algorithm for plagiarism checker involves several key steps. Initially, both the original document and the suspicious document undergo tokenization and preprocessing, where they are separated into individual words. Subsequently, unique sets of unigrams (single words) are created for each document. The algorithm then calculates the Jaccard similarity coefficient, a measure of similarity between the sets of unigrams of the original and suspicious documents. The Jaccard Similarity is determined by the size of the intersection of the two sets divided by the size of their union.

A threshold value is established to serve as a criterion for similarity, above which documents are considered potentially plagiarized. This threshold provides a customizable parameter that influences the sensitivity of the plagiarism detection. Finally, a plagiarism decision is made based on the calculated Jaccard similarity. If the similarity exceeds the predetermined threshold, the documents are flagged as potentially plagiarized; otherwise, they are considered non-plagiarized. This algorithm provides a systematic approach to identifying potential instances of plagiarism by comparing the overlap of unique words between documents and making decisions based on a defined similarity threshold.

4.2 ASSUMPTIONS AND DEPENDENCIES

In developing this plagiarism checker project, using n-gram algorithm, several key assumptions shape its trajectory. Assumptions include

- The plagiarism detection system focuses on textual content, assuming that similarities and potential instances of plagiarism can be identified by analyzing written language.
- The system assumes that the basic unit of analysis is a document or a piece of text, and it doesn't consider other media types like images, videos, or audio.
- The system assumes that it is designed for a specific language or set of languages. Detecting plagiarism across multiple languages may require additional considerations and complexities.
- The system assumes that the content to be analyzed is in a digital text format. Handwritten or non-digital content is not considered.
- The system assumes a tokenization process to break down texts into smaller units, such as words or n-grams, to facilitate comparison and analysis.
- The system assumes that the length of the documents being compared is within a reasonable range. Extremely short or extremely long documents may pose specific challenges.
- Dependencies rest on Python environment and availability of sample text files.

CHAPTER 5 REQUIREMENTS

5.1 FUNCTIONAL REQUIREMENTS

- File Reading: The code should be able to read text content from two specified files.
- N-gram Generation: The code must generate n-grams from the text content of both files. In this case, the n-grams are generated with a specified value of 'n'.
- Similarity thresholding: The code must apply a threshold to the calculated Jaccard similarity coefficient to determine whether the two texts are considered similar or not. In the provided code, a threshold of 0.4 is used.
- Jaccard Similarity Calculation: The code is expected to calculate the Jaccard similarity between the n-grams of the two texts using the following formula:

  JS = Set of intersection / set of union .
- Output Display: The code should display the calculated Jaccard similarity coefficient and a corresponding message indicating whether the texts are considered similar or not.

5.2 NON-FUNCTIONAL REQUIREMENTS

- Performance: Must be highly efficient with response time for the system to respond to a user request.
- Reliability: Must perform without failure in 95 percent use cases.
- Code Readability: Specifies coding standards and practices to ensure the code is maintainable.
- Portability: Must run on Windows 10 and Windows 11

5.3 SOFTWARE/SYSTEM REQUIREMENTS

5.3.1 System Requirements

- OS: Windows 10/ Linux
- Processor: 4 cores or more processor; 7th Gen or higher (Ryzen/Intel)
- Power Supply: 600W power supply
- GPU: Minimum 4GB VRAM
- RAM: Minimum 16GB RAM
- Robust Cooling System

5.3.2 Software Requirements

- Programming Language: Python 3.10 or higher
- Jupyter Notebook / Google Colab
- Third –Party Libraries: Any external library required for the application user is using.

CHAPTER 6 METHODOLOGY

Problem Definition:

Identify the need for a plagiarism detection tool.

Define the problem of measuring similarity between two texts using Jaccard similarity and n-grams.

Methods used to collect the test data:

We used jupyter notebook to execute the code for our project.

First, we created five text files in jupyter notebook.

Assuming file 1 to be original text document and file 2, file 3 and file 4 to be suspicious documents.

While executing the code we are reading these files using the path of the file and with statement.

Strategy of this project includes :

Preprocessing

N-gram Generation

Jaccard similarity calculation

Threshold settings

File reading and comparison

Reporting and Visualization

Assumptions in this project includes include

- The plagiarism detection system focuses on textual content, assuming that similarities and potential instances of plagiarism can be identified by analyzing written language.
- The system assumes that the basic unit of analysis is a document or a piece of text, and it doesn't consider other media types like images, videos, or audio.
- The system assumes that the content to be analyzed is in a digital text format. Handwritten or non-digital content is not considered.
- The system assumes a tokenization process to break down texts into smaller units, such as words or n-grams, to facilitate comparison and analysis.

Flowchart:

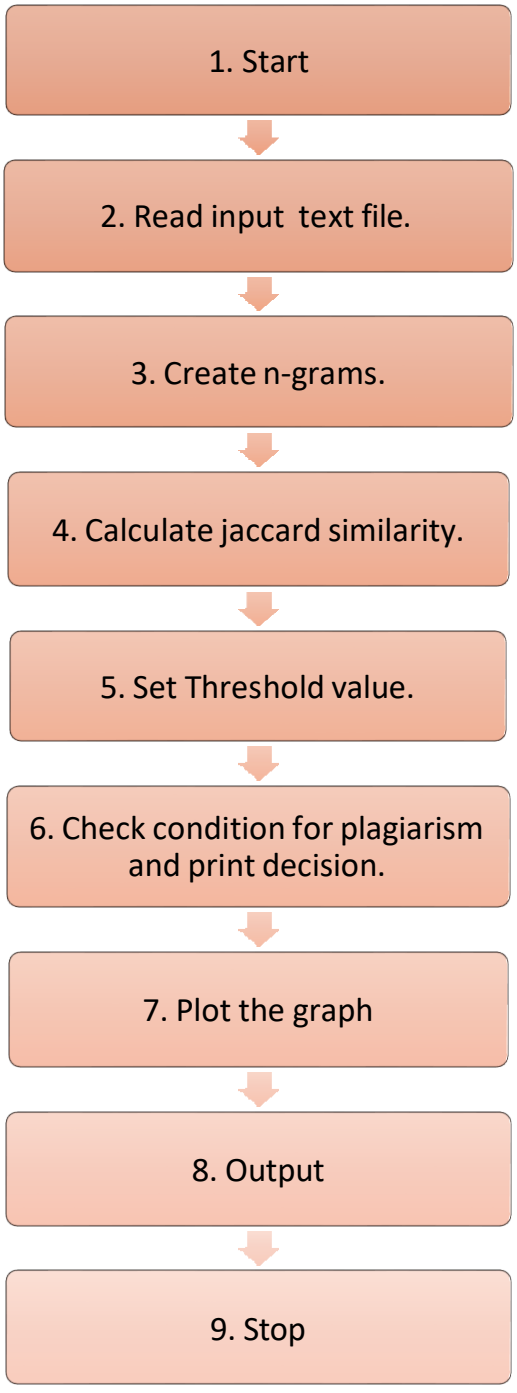

Fig 6.1: Flowchart for Methodology

CHAPTER 7 EXPERIMENTATION

7.1 ALGORITHM:

1. Input:

Original Document ("original_document")

Suspicious Document ("suspicious_document")

2. Tokenization and Preprocessing:

Tokenize both the original and suspicious documents into individual words.

3. Create Sets of Unigrams:

Create sets of unique words (unigrams) for both the original and suspicious documents.

4. Calculate Jaccard Value:

Calculate the Jaccard similarity coefficient between the sets of unigram of the original and suspicious documents.

Jaccard Similarity = Size of Intersection/Size of Union

5. Set Threshold:

Choose a threshold value for similarity above which the documents are considered potentially plagiarized.

6. Plagiarism Decision:

If the calculated Jaccard similarity is above the threshold, flag the documents as potentially plagiarized. Otherwise, consider them non-plagiarized.

7.2 CODE SNIPPETS:

```
In [23]: def get_ngrams(text, n):
             """
             Generate n-grams from a given text.
             """
             words = text.split()
             ngrams = [tuple(words[i:i+n]) for i in range(len(words) - n + 1)]
             return set(ngrams)
```

Figure 7.1: Function to get n-gram of the given input file

```python
def jaccard_similarity(text1, text5, n=3):
    """
    Calculate Jaccard's similarity between two texts using n-grams.
    """
    ngrams1 = get_ngrams(text1, n)
    ngrams5 = get_ngrams(text5, n)

    intersection = len(ngrams1.intersection(ngrams5))
    union = len(ngrams1.union(ngrams5))

    similarity = intersection / union if union > 0 else 0
    return similarity
```

Figure 7.2: Function to calculate Jaccard similarity

```python
def read_text_from_file(file_path):
    """
    Read text content from a file.
    """
    with open(file_path, 'r', encoding='utf-8') as file:
        return file.read()
if __name__ == "__main__":

    file1_path = "f1.txt"
    file5_path = "f5.txt"

    text1 = read_text_from_file(file1_path)
    text5 = read_text_from_file(file5_path)
```

Figure 7.3: Function to read the text from the input file

```python
def jaccard_similarity(text1, text5, n=3):
    """
    Calculate Jaccard's similarity between two texts using n-grams.
    """
    ngrams1 = get_ngrams(text1, n)
    ngrams5 = get_ngrams(text5, n)

    intersection = len(ngrams1.intersection(ngrams5))
    union = len(ngrams1.union(ngrams5))

    similarity = intersection / union if union > 0 else 0
    return similarity
```

Figure 7.4: Calling the Jaccard similarity function

In [38]:
```
import matplotlib.pyplot as plt

def plot_similarity_graph(similarity1, similarity2, file_name):
    """
    Plot a bar graph comparing similarities for a common text file.
    """
    labels = ['Similarity1', 'Similarity2']
    percentages = [similarity1 * 100, similarity2 * 100]

    # Plotting
    plt.bar(labels, percentages, color=['Red', 'Green'])
    plt.title(f'Similarity Comparison for {file_name}')
    plt.xlabel('files')
    plt.ylabel('Percentage')
    plt.ylim(0, 100)
    plt.show()
```

Figure 7.5: Defining the graph

```
if __name__ == "__main__":
    file1_path = "f1.txt"
    file5_path = "f5.txt"

    text1 = read_text_from_file(file1_path)
    text5 = read_text_from_file(file5_path)
    similarity1 = jaccard_similarity(text1, text5, n=3)
    print(f"Jaccard Similarity 1: {similarity1*100}")

    file3_path = "f3.txt"
    text3 = read_text_from_file(file3_path)
    similarity2 = jaccard_similarity2(text1, text3, n=3)
    print(f"Jaccard Similarity 2: {similarity2*100}")

    # Plotting graph for similarities
    plot_similarity_graph(similarity1, similarity2, "f1.txt")
```

Figure 7.6: Plotting the plagiarism percentage

CHAPTER 8 RESULTS

Output for the given input parameters:

Case-1 : Text file 1-original document

Text file 2 –suspicious document

Threshold value=0.4

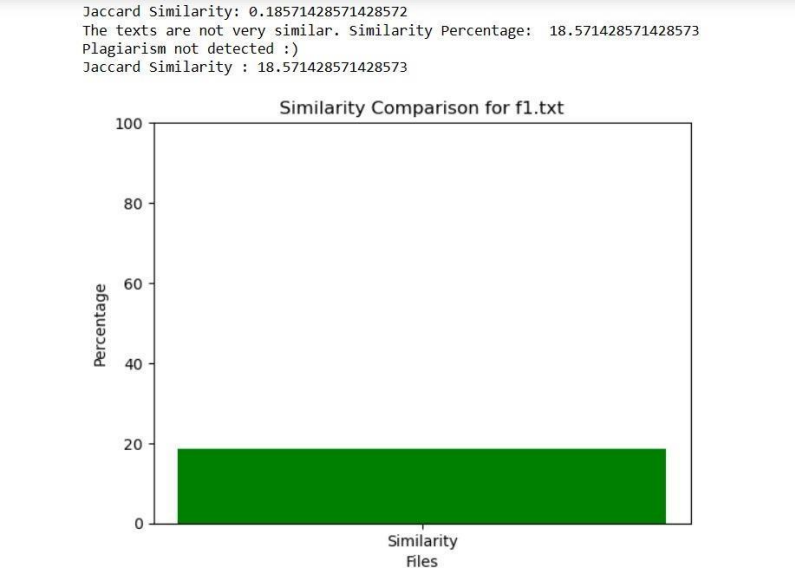

Figure 8.1: Output for test data 1

Case 2: Text file 1-original document

Text file 3–suspicious document

Threshold value=0.4

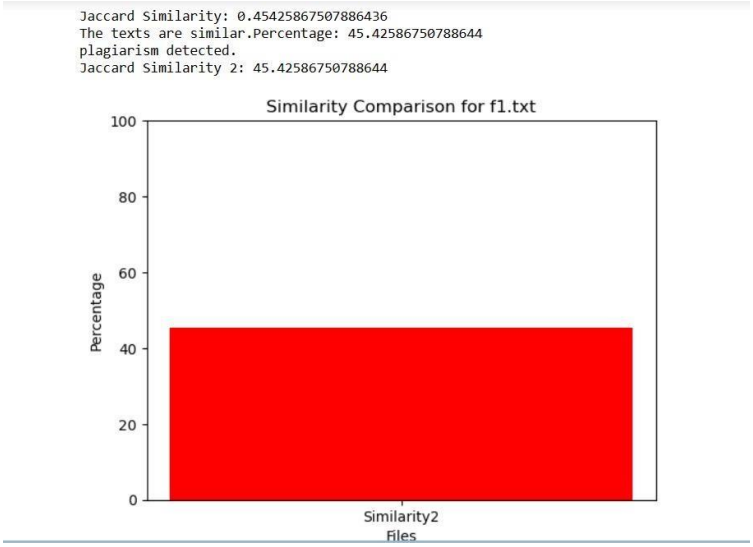

Figure 8.2: Output for test data 2

Case 3: Text file 1-original document
Text file 4–suspicious document
Threshold value=0.4

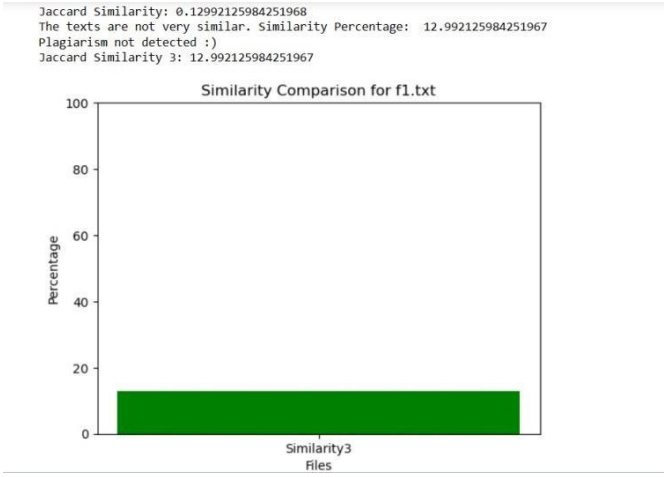

Figure 8.3: Output for test data 3

Case 4: Text file 1-original document
Text file 5–suspicious document
Threshold value=0.4

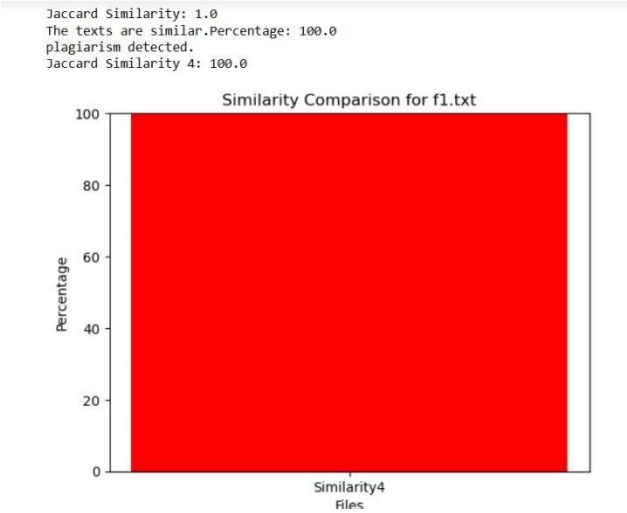

Figure 8.4: Output for test data 4

Similarity Chart : File 1 with file 2, file 3, file 4, file 5.
Here, file 1 – original document

File 2 , file 3, file 4, file 5 – Suspicious document

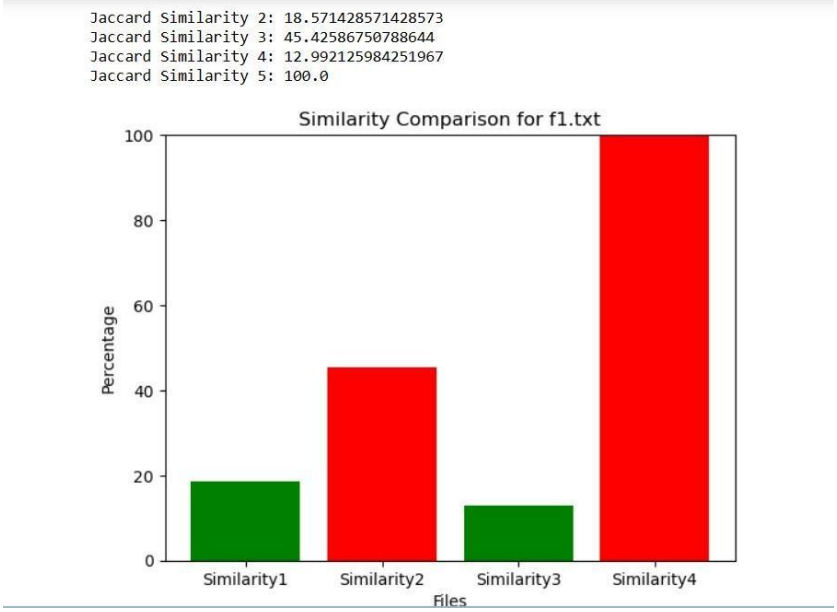

Figure 8.5: Comparison of all test data outputs

CONCLUSION

In conclusion our project aims to enhance the academic integrity and honesty. It helps maintain the credibility of original work, encourages proper citation and attribution, and discourages plagiarism. Employing n-grams and jaccard similarity for plagiarism detection proves to be a robust and effective approach in identifying textual similarities across documents. The utilization of n-grams, which involves breaking down text into contiguous sequences of n items (typically words), allows for a granular representation of textual content. When combined with the jaccard similarity measure, which computes the intersection and union of sets formed by the n-grams, we gain a powerful tool for assessing the degree of similarity between two documents.

Furthermore, the computational efficiency of n-gram and jaccard similarity calculations enables the handling of large datasets, making it suitable for real-time or batch processing in plagiarism detection systems.

Thus, the Jaccard Similarity-based Plagiarism Detection project has successfully delivered a valuable tool for comparing texts and detecting potential plagiarism. Its simplicity, flexibility, and accuracy make it a useful asset for users concerned with maintaining the integrity of textual content.

REFERENCES

[1] S. Brin, J. Davis, H. Garcia-Molina, Copy detection mechanisms for digital documents, in: ACM SIGMOD Record, Vol. 24, ACM, 1995, pp. 398-409. 24.

[2] M. S. Anderson, N. H. Steneck, The problem of plagiarism, in: Urologic Oncology: Seminars and Original Investigations, Vol. 29, Elsevier, 2011, pp. 90-94.

[3] P.Ashwin, M.B.Boominathan and G.Suresh, "Plagiarism Detection Tool for Coding Platform using Machine Learning", International Research Journal of Engineering and Technology (IRJET), Volume: 08 Issue: 05 May 2021, Tamil Nadu, India.

[4] Michal Duracik , Patrik Hrkut , Emil Krsak, (Member, IEEE) and Stefan Toth, "Abstract Syntax Tree Based Source Code AntiPlagiarism System for Large Projects Set", October 6, 2020, Volume 8, 2020,Digital Object Identifier:10.1109/ACCESS.2020.3026422.

[5] Thai-Bao Do, Huu-Nghia H. Nguyen, Bao-Linh L. Mai and Vu Nguyen, "Mining and Creating a Software Repositories Dataset", 2020 7th NAFOSTED Conference on Information and Computer Science (NICS), 02 February 2021, 978-0-7381-0553-6, Ho Chi Minh City, Vietnam .

[6] Nishesh Awale, Mitesh Pandey, Anish Dulal and Bibek Timsina, "Plagiarism Detection in Programming Assignments using Machine Learning", Journal of Artificial Intelligence and Capsule Networks 2020),21.07.2020.

[7] Huang Qiubo, Tang Jingdong and Fang Guozheng,"Research on Code Plagiarism Detection Model Based on Random Forest and Gradient Boosting Decision Tree", ICDMML 2019: Proceedings of the 2019 International Conference on Data Mining and Machine Learning, April 2019.

[8] Narjes Tahaei and David C. Noelle, "Automated Plagiarism Detection for Computer Programming Exercises Based on Patterns of Resubmission", ICER '18: Proceedings of the 2018 ACM Conference on International Computing Education Research, August 2018.