ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/379476003

Detection of Source Code Plagiarism Utilizing an Approach Based on Machine Learning

Article *in* International Journal of Computing · April 2024

DOI: 10.47839/ijc.23.1.3438

CITATIONS
0

READS
180

2 authors, including:

Raddam Sami Mehsen
Middle Technical University
12 PUBLICATIONS **17** CITATIONS

SEE PROFILE

All content following this page was uploaded by Raddam Sami Mehsen on 05 June 2024.

The user has requested enhancement of the downloaded file.

Date of publication MAR-31, 2024, date of current version JAN-31, 2024.
www.computingonline.net / computing@computingonline.net

Print ISSN 1727-6209
Online ISSN 2312-5381
DOI 10.47839/ijc.23.1.3438

Detection of Source Code Plagiarism Utilizing an Approach Based on Machine Learning

RADDAM SAMI MEHSEN[1,2], HIREN D. JOSHI[1]
[1]Department of Computer Science, Gujarat University Ahmadabad, Gujarat, India
[2]Middle Technical University, Baghdad, Iraq

Corresponding author: Raddam Sami Mehsen (e-mail: raddamsami@gujaratuniversity.ac.in).

ABSTRACT Academic institutions, which often publish papers and journals, are ideal testing grounds for the efficacy of counterfeit detection methods. Plagiarism occurs when someone uses the words of another writer without giving that writer proper credit. The proliferation of freeware text editors and the increasing availability of scientific materials online have made the detection of plagiarism a pressing concern; however, the detection of plagiarism in the source code presents a particularly difficult problem. Plagiarism detection algorithms for identification systems and software source code have been the subject of numerous academic investigations. The proposed method combines TF-IDF transformations with K-means clustering to achieve a 99.2% accuracy rate when detecting instances of plagiarism in the source code. This is because it groups similar lines of code together. On the other hand, in comparison to the outcomes produced by the random forest algorithm, the ones that it generates are significantly better. The performance of the MOSS system that was already in place was inferior to that of the system that was used for 90% and 80% of the training set. When contrasting the results, some parameters for evaluation that are considered include precision, recall, and F-measure. The proposed system is implemented in Jupyter Notebook 7 and Python. Also, graphic user interface is designed and implemented to give user friendly experience to the users.

KEYWORDS Source code; plagiarism; machine learning; C++; python; programming assignments

I. INTRODUCTION

MARTINS states that plagiarism occurs when one use someone else's work without giving it credit [36]. Because so much information is available online, plagiarism becomes the major problem for research community. Academic papers frequently consist of essays, reports, and scientific articles, making textual plagiarism at the textual level one of the most common forms of copyright infringement. According to the recent study, most among 16 out of 100 published original articles are redundant. On the other hand, plagiarism is when one use someone else's words or code in a way that is not right [1]. Plagiarism in assignments of source code is another major issue when someone tries to copy someone else's source code as their own without giving credit to the original author. Programming assignments in academia frequently feature plagiarized source code. To obtain good grades students try to copy source code assignments from their peers. New admitted students, who submit plagiarized code in their first course, will then do so in their subsequent courses. That is the reason, it is highly important to stop this unlawful conduct immediately [2]. It is possible for a teacher to obtain inaccurate information about the difficulty level and students' progress in a class. Therefore, identifying instances of plagiarism in academic work is a crucial endeavor. When there are many students in a class, it can take considerable computation time and effort to go through each solution and determine whether it is original or copied from another student [3]. When compared with automated systems, manual inspection takes too long and produces too few reliable results to be practical. To pair submissions that are similar to one another, teacher could use source code comparison tools, which can find such plagiarism, JPlag and Measure of Software Similarity (MOSS) were reported in [4]. For the most part, the syntactic aspects of the assignments are used in many algorithms to detect its plagiarism. However, code obfuscation renders both approaches ineffective at automatically detecting instances of software plagiarism. To hide the code and avoid detection, students frequently resort to dishonest methods [5]. Section III proposes k-means clustering, a machine learning based approach to examine this type of dishonest conduct.

Section IV elaborates the methodology, and Section V reports the experimental work done and the results.

II. RELATED WORK

Plagiarism in previous work was detected using program similarity metrics such as MOSS, JPlag, and others. The methods based on the assignment properties or text-based methods are used at a syntactic level to find plagiarism [6].

MOSS is based on the local fingerprinting strategy of the winnowing property of syntactic assignments. The fingerprint selection mechanism used by MOSS is not particularly precise; it simply chooses the fingerprint with the lowest value within a given time window. In addition to this fingerprint, a lookup is performed for the longest common sequence [7]. JPlag is another widely used plagiarism detection tool that uses greedy string tiling to find the longest, most common sequences in the tokenized form of the source code based on each pair of submissions. The way JPlag works is similar to the way MOSS works. JPlag, on the other hand, only looks for common tokenized structural blocks when comparing the code. Therefore, it misses important aspects such as formatting and style. There have been other attempts to find plagiarized content, such as analyzing dependency graphs in computer programs [8].

The authors [9] introduced GPlag, a novel method for detecting plagiarism by mining program dependency graphs (PDGs). A PDG is a graphical representation of the data and control flow in a process. Because PDGs do not change much during the plagiarism process, GPlag is more effective at detecting plagiarized work than PDGs.

A representation of pairs in the source code considers lexical, stylistic and structural aspects, comments and programmer text. Character sequences can convey lexical and n-gram information and comments. These traits are not meant to help one determine the programing language someone are using. Instead, they help to spot the bits of everyday language that programmers always leave behind [10].

Academics have developed a way to compare assignments that teachers can use to judge their students' work. They offered 12 features, such as comment and white-space matching and the MOSS similarity score [11]. However, both MOSS and JPlag exclude these details from their assessments. Their primary function is to serve as a signal for identifying instances of plagiarism. This system employs neural network algorithms to determine the weight to be given to each criterion in the evaluation. More specifically, they care most about finding duplicate issues within a single set. Conversely, the proposed solution is independent of the problem at hand [12]. Fig. 1. Shows a typical block diagram for source code plagiarism detection model.

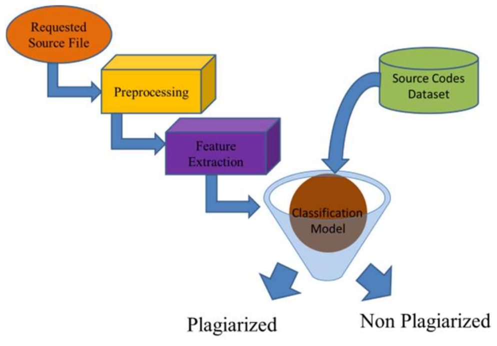


Figure 1. Typical source code plagiarism detection model

Such systems can be used for many purposes, such as finding plagiarism in student's work [13].

- Criminal Prosecution: Tracking the Malware Creator
- Corporate Litigation – If an employee breaches a non-compete clause in a contract, determine who wrote the code.
- Plagiarism detection: Tracing the original author in instances of copied work.

Plagiarized snippets are those that have taken large parts of other snippets and changed only a few small things. There are different types of situations in which plagiarism can occur:

- If the due date for the project is near, the students' openness to sharing the code with one another will be increased.
- When students work together on a project, the resulting programs may be identical in every way except for the names and structures the authors choose to give them.
- There is a risk of assignment theft when using shared resources such as printers and computers.
- The previous semester's software can be used without modification because only the requirements change from one semester to the next.

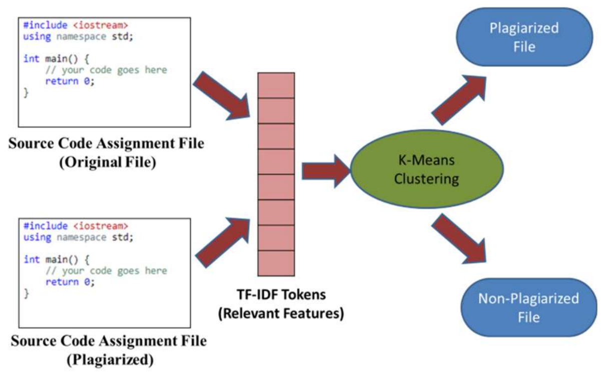
Many students independently offer the same design for short assignments, so it may look like there was plagiarism [14].

III. PROPOSED APPROACH

Here are the measures that make up the proposed system [15]:

- Browse the C++ project source code files.
- This means that we need to collect and preprocess the source codes.
- Turn each file into a list of tokens, save these as token files, join these token files together, and then create a list of terms and files. This method is the same as that used to determine the TF-IDF files.
- Examine the C++ file in which the query was written.
- It is time to return to the third stage.
- Remember to return to Step 4.
- Apply the random forest classifier to all files in the query to determine which files are similar.
- If the level of similarity is sufficiently high, the file is considered malicious; otherwise, it is considered safe.
- A report must be made if any instances of plagiarism are found. If that does not work, go back to the beginning.

Fig. 2 depicts the proposed model.



```
#include <iostream>
using namespace std;

int main() {
    // your code goes here
    return 0;
}
```

```
#include <iostream>
using namespace std;

int main() {
    // your code goes here
    return 0;
}
```

Source Code Assignment File (Original File)
Source Code Assignment File (Plagiarized)
TF-IDF Tokens (Relevant Features)
K-Means Clustering
Plagiarized File
Non-Plagiarized File

Figure 2. Proposed model for finding copied source code, where features were taken from files that had been requested using TFIDF tokens and then trained using random forest classifier.

A. CODE METRICS

When the source codes are sent to the proposed system, it pulls the characteristics from the files sent in. As parts of the proposed model, the TFIDF code metrics are chosen [16]. Code metrics are a collection of tokens taken from the source code files, such as students' assignments. Tokens like these help evaluate how a programmer's code works, how it looks, how much it costs, how reliable it is, how flexible it is, and how it is structured. Several tools and methods were proposed for obtaining these code metrics [17]:

- N-grams are a group of n objects from a text corpus, such as words and letters. Using this method, documents are broken up into a list of substrings of length n and the number of times they appear in the document. This concept was introduced using natural processing language. For instance, one United Kingdom is equal to two grams [18].
- Term Frequency-Inverse Document (TF-IDF) – It stands for "frequency." It gives words in document scores that show how important they are based on where they are in the corpus. Information extraction and text mining are two of its primary applications.
- ANTLR is a parser generator that helps read and process programing languages. Vocabulary and syntax are also taken out, which helps us understand how the code is put together [19].

Many students who plagiarize make substantial revisions to their work to conceal it. Several examples of the many ways in which modifications to source code can manifest themselves are provided below:

- Modifying the format entails only making editorial changes, such as adding or removing comments and blanks [20, 21].
- Another common thing is to change the names of identifiers to steal someone else's work that does not harm the code's integrity [22].
- Statements that do not rely on one another in a sequential fashion, such as declarations, can be easily moved by rearranging them [23].
- Control Substitution: Programming languages offer various alternatives to common coding structures, such as loops and if/else expressions.
- Code Insertion: Codes that do not change the program's original logic can be added to hide plagiarism [24].

The proposed model cannot handle these transformations. The retrieved features were trained using a supervised classifier from the machine learning toolkit, that is the random forest classifier [24, 26].

B. MACHINE LEARNING PHENOMENA

Algorithms that can learn from data and make predictions are the focus of machine learning research. These algorithms create a model that uses sample inputs rather than static program instructions to produce data-based predictions or judgments. It is used for various computing tasks when the cost of designing and implementing explicit algorithms is prohibitive [25]. Machine learning is used in a wide variety of everyday activities, including searching the web, filtering content on social media, and making product recommendations on online retailers' websites. We use machine learning every day without realizing it. Machine learning provides accurate speech recognition, speedy web searches, driverless vehicles, and a greater understanding of the human genome [25]. There are typically three types of machine learning tasks that are distinguished by the nature of the feedback given to the learning system. Machine learning techniques can be used to investigate claims of code duplication [27].

Here are the specifics:

- In supervised learning, a teacher shows a student how to use the computer by showing them some inputs and results. The objective is to develop a generalized rule or function that maps inputs to desired outcomes. Classification and regression fall under this umbrella [21].
- When the learning algorithm is not given labels, it is up to it to determine how the data is organized (known as unsupervised learning). The algorithms in this category can be used to perform clustering [28].
- The goal of reinforcement learning is to teach a computer to perform a task in an uncertain environment, such as when playing a game against a human, without providing any feedback about the outcome of the task. As the program moves through its problem space, feedback is provided in the form of rewards and penalties [23].

IV. ALGORITHM OF K-MEANS CLUSTERING

In the fields of data science and machine learning, k-means clustering is used to address issues related to grouping similar data points together [21-24].

A. ALGORITHM FOR LEARNING WITHOUT SUPERVISION K-MEANS

This unlabeled dataset can be organized using clustering. The parameter K dictates the minimum number of clusters that must be created before proceeding with the procedure; for example, if K=2, only two clusters will be created [30].

This allows us to classify information and provides a straightforward method for identifying the classes of individuals in a dataset that has not been labeled [31].

Because it is a centroid-based method, each cluster has its own centroid. Reducing the total distances between data points and the clusters, which they belong to, is the primary focus of this method [32].

The process begins with an unlabeled dataset, divides it into k clusters, and iteratively refines the clusters until no improvements can be made. The parameter k in this algorithm must be set in advance [33].

The k-means clustering algorithm accomplishes two goals:

1) it iteratively determines the best value for the K center points or centroids;

2) it assigns the data point to the k-center that is geographically closest to that data point. Clusters of data points are formed when they share a common k-center.

As a result, the data points that make up each cluster share certain characteristics while remaining distinguishable from one another [34].

The k-means clustering algorithm is shown in Fig. 3 below [34]:

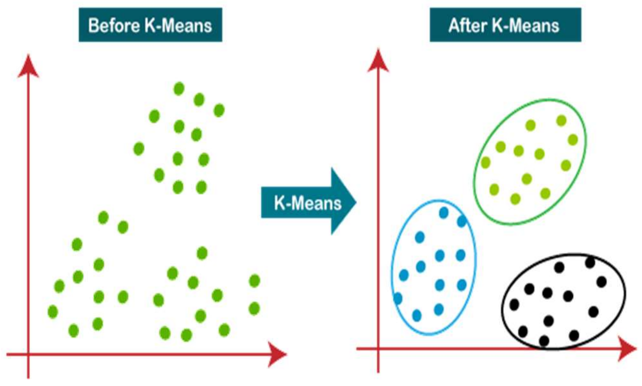


Figure 3. Working of K-means clustering algorithm

V. EXPERIMENTAL RESULTS AND PERFORMANCE EVALUATION

The proposed model is evaluated using multiple metrics. The following sections describe the metrics that should be tracked for each system and iteration: recall, accurate response rate, cross-value score, F1 score, precision and other confusion matrices [2, 4, 24, 31].

A. CORRECTLY CLASSIFYING DATA

In the context of assessing classification tasks, it is by far the most popular metric to use. Accuracy is the ratio of correct to total predictions [32].

$$Precision = True\ Predictions / Total\ Guesses. \quad (1)$$

With binary classification, the accuracy formula is [31]:

$$Accuracy = (TN+TP) / (FP + FN + TP + TN). \quad (2)$$

In this definition, TP refers to a true positive result, TN to a true negative result, FP to a false positive result, and FN to a false negative result [4].

In a scenario with unequally distributed classes, it is not a fair measure to optimize for because it may be relatively high, thus helping the majority while neglecting the minority. Therefore, it is not a fair measure to optimize for [6].

B. CONFUSION MATRIX

The performance of a classification model can be measured by calculating the confusion matrix, which is an N-by-N matrix where N is the number of target classes. The matrix evaluates the machine learning model's predictions against the actual goal values [24].

C. F-MEASURE

F-Measure uses a combined measure of precision and recall that determines the accuracy of the test data. As shown in Fig. 3, the system operates on the basis of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). Precision is defined as the proportion of correct predictions to all correct predictions [22-25].

$$p = TP / (TP + FP). \quad (3)$$

However, recall is calculated by dividing the number of accurate predictions by the total number of true positives [7].

A simple formula:

$$r = TP / TP + FN. \quad (4)$$

For the F-Score, a weighted average of the recall and accuracy values is used.

It can be written as:

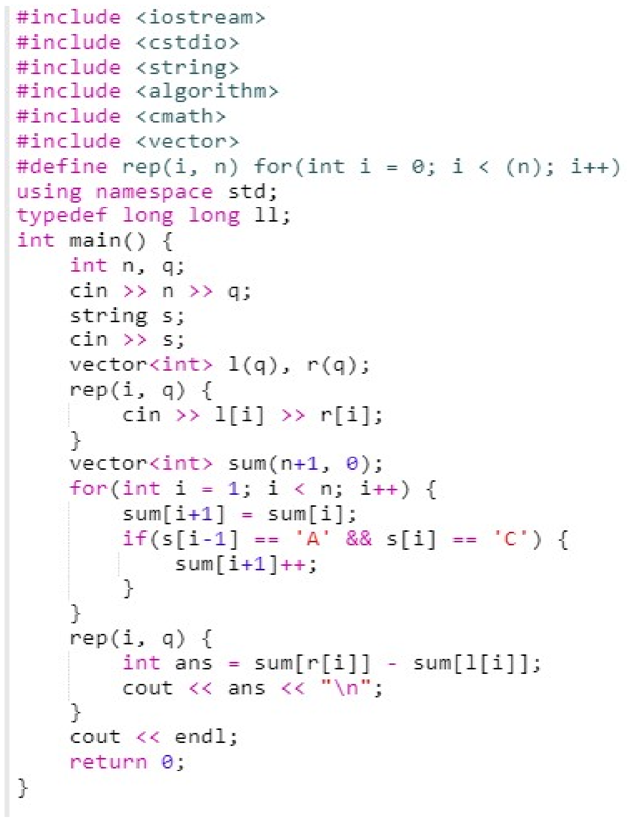
$$F = 2*((precision * recall)/(precision + recall)). \quad (5)$$

The formula for determining the F1 score, also known as the F-value, is:

$$F1\text{-}score=Harmonic\ mean\ (precision,\ recall). \quad (6)$$

The proposed system is tested on a dataset consisting of Python solutions to homework problems from a beginner C++ programing competition. There are 44 separate entries, each of which covers 10 unique problems. Each student is permitted to submit 20 total pieces of work, which may include drawings and final drawings. Approximately 880 entries have led us to this point. Some representative examples of the dataset code are as follows [6]:

The code for a C++ application is displayed in its entirety in Fig. 4, which is taken from the original file.



```
#include <iostream>
#include <cstdio>
#include <string>
#include <algorithm>
#include <cmath>
#include <vector>
#define rep(i, n) for(int i = 0; i < (n); i++)
using namespace std;
typedef long long ll;
int main() {
    int n, q;
    cin >> n >> q;
    string s;
    cin >> s;
    vector<int> l(q), r(q);
    rep(i, q) {
        cin >> l[i] >> r[i];
    }
    vector<int> sum(n+1, 0);
    for(int i = 1; i < n; i++) {
        sum[i+1] = sum[i];
        if(s[i-1] == 'A' && s[i] == 'C') {
            sum[i+1]++;
        }
    }
    rep(i, q) {
        int ans = sum[r[i]] - sum[l[i]];
        cout << ans << "\n";
    }
    cout << endl;
    return 0;
}
```

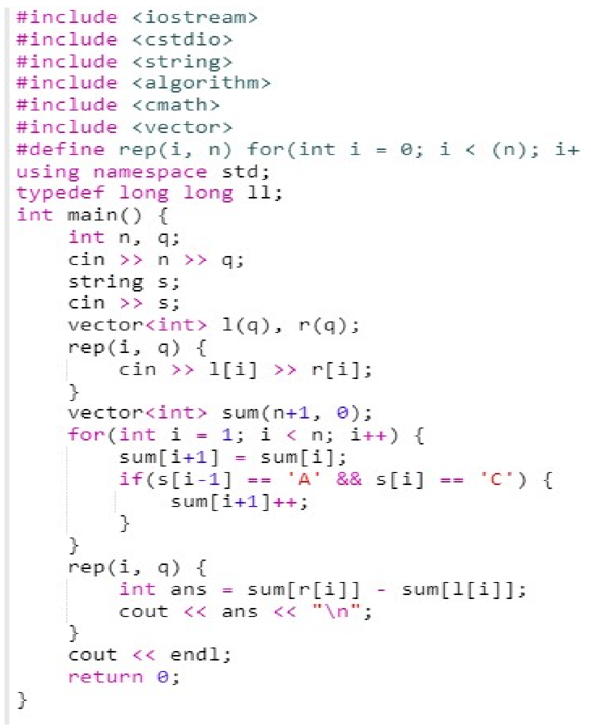
Figure 4. Example of the program file's original source code, which has not been plagiarized

The generated characteristics for the sample program are shown in Fig. 5. It represents TF-IDF tokens for the selected file.

VAR FUNC PAREN VAR PUNC VAR PAREN SENTENCE PAREN TYPE VAR OP NUM PUNC VAR OP PAREN PAREN VAR OP PAREN NAMESPACE NAMESPACE VAR PUNC TYPE TYPE TYPE TYPE PUNC TYPE FUNC PAREN PAREN PAREN TYPE VAR PUNC VAR PUNC VAR OP VAR OP VAR PUNC VAR VAR PUNC VAR OP VAR PUNC VAR OP TYPE OP FUNC PAREN VAR PAREN PUNC FUNC PAREN VAR PAREN PUNC FUNC PAREN VAR PUNC VAR PAREN PAREN VAR OP VAR PAREN VAR PAREN OP VAR PAREN VAR PAREN PUNC PAREN VAR OP TYPE OP FUNC PAREN VAR OP NUM PUNC NUM PAREN PUNC SENTENCE PAREN TYPE VAR OP NUM PUNC VAR OP VAR PUNC VAR OP PAREN PAREN VAR PAREN VAR OP NUM PAREN OP VAR PAREN VAR PAREN PUNC SENTENCE PAREN VAR PAREN VAR OP NUM PAREN OP VAR OP VAR PAREN VAR PAREN OP VAR PAREN PAREN VAR PAREN VAR OP NUM PAREN OP PAREN PAREN FUNC PAREN VAR PUNC VAR PAREN PAREN TYPE VAR OP VAR PAREN VAR PAREN VAR PAREN PAREN OP VAR PAREN VAR PAREN VAR PAREN PAREN VAR OP VAR OP STR PUNC PAREN VAR OP VAR PUNC SENTENCE NUM PUNC PAREN

Figure 5. A tokenized TF-IDF file for a file that has not been copied.

The respective original source code file also has plagiarized file. It is shown in Fig. 6.



```
#include <iostream>
#include <cstdio>
#include <string>
#include <algorithm>
#include <cmath>
#include <vector>
#define rep(i, n) for(int i = 0; i < (n); i+
using namespace std;
typedef long long ll;
int main() {
    int n, q;
    cin >> n >> q;
    string s;
    cin >> s;
    vector<int> l(q), r(q);
    rep(i, q) {
        cin >> l[i] >> r[i];
    }
    vector<int> sum(n+1, 0);
    for(int i = 1; i < n; i++) {
        sum[i+1] = sum[i];
        if(s[i-1] == 'A' && s[i] == 'C') {
            sum[i+1]++;
        }
    }
    rep(i, q) {
        int ans = sum[r[i]] - sum[l[i]];
        cout << ans << "\n";
    }
    cout << endl;
    return 0;
}
```

Figure 6. Plagiarized program 1 source code

The TF-IDF output of the plagiarized sample code shown in Fig. 6, is displayed in Fig. 7 shown below:

VAR FUNC PAREN VAR PUNC VAR PAREN SENTENCE PAREN TYPE VAR OP NUM PUNC VAR OP PAREN PAREN VAR OP PAREN NAMESPACE NAMESPACE VAR PUNC TYPE TYPE TYPE TYPE PUNC TYPE FUNC PAREN PAREN PAREN TYPE VAR PUNC VAR PUNC VAR OP VAR OP VAR PUNC VAR VAR PUNC VAR OP VAR PUNC VAR OP TYPE OP FUNC PAREN VAR PAREN PUNC FUNC PAREN VAR PAREN PUNC FUNC PAREN VAR PUNC VAR PAREN PAREN VAR OP VAR PAREN VAR PAREN OP VAR PAREN VAR PAREN PUNC PAREN VAR OP TYPE OP FUNC PAREN VAR OP NUM PUNC NUM PAREN PUNC SENTENCE PAREN TYPE VAR OP NUM PUNC VAR OP VAR PUNC VAR OP PAREN PAREN VAR PAREN VAR OP NUM PAREN OP VAR PAREN VAR PAREN PUNC SENTENCE PAREN VAR PAREN VAR OP NUM PAREN OP VAR OP VAR PAREN VAR PAREN OP VAR PAREN PAREN VAR PAREN VAR OP NUM PAREN OP PAREN PAREN FUNC PAREN VAR PUNC VAR PAREN PAREN TYPE VAR OP VAR PAREN VAR PAREN VAR PAREN PAREN OP VAR PAREN VAR PAREN VAR PAREN PAREN VAR OP VAR OP STR PUNC PAREN VAR OP VAR PUNC SENTENCE NUM PUNC PAREN

Figure 7. Tokenized TF-IDF file of the plagiarized file example program 1

Table 1 summarizes the information regarding the source files used in the experiments.

Table 1. Details of Database

Dataset	Problem Sets	Submission	Language	Average LOC
Programming Contest	10	880	C++	1207

The evaluation metrics were calculated and compared following the generation of the confusion matrix. In 90% and 80% of the training sets, the proposed system was found to be more accurate than the current MOSS system by a margin of 99.2%. We re-compared the proposed model using various assessment criteria (given in Table 2) and found that it is superior to the existing techniques, which have not been reported to have a satisfactory level of accuracy.

Table 2. Evaluation of the proposed system

	Random Forest Algorithm	K-means Clustering	Moss (90%)	Moss (80%)
Accuracy	0.935	0.992	-	-
Precision	0.960	0.009	1.000	0.920
Recall	0.906	0.054	0.631	0.819
F1 Score	0.931	0.016	0.773	0.866

Fig. 8 displays learning curve where the relationship between training samples and score is been depicted. The output window is shown Fig. 9.

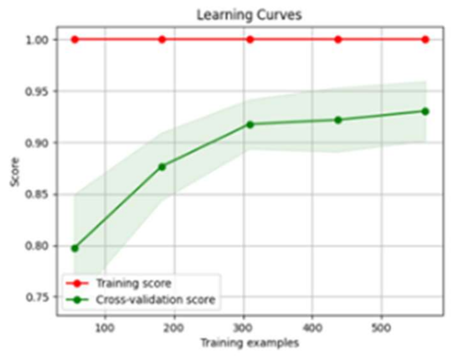


Figure 8. The relationship between the score and the number of training instances (Score Vs Training Examples)

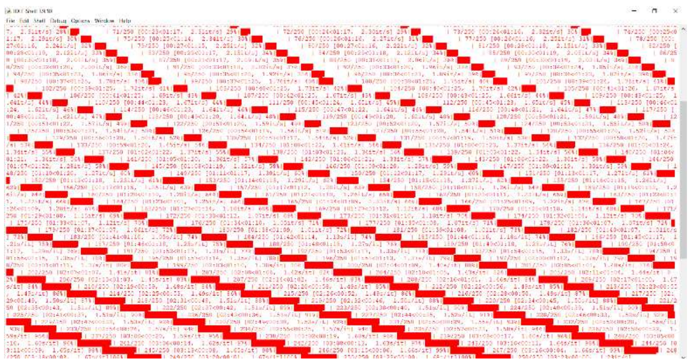


Figure 9. Resultant snippet of the python code

This research also implemented GUI based application in python, where the source code files can be tested for the performance check. The following figures show the interface designed and implemented. The first screen shows a Text Input where user can select a source code file from his/her computer and it is displayed in the Text Input. When the user / trainer clicks on the "check plagiarism" button, the written python code evaluates whether the input source code is plagiarized and which source is from the trained model. If it matches with the source plagiarized in the next window it shows the details of the input source file such as total number of characters, words, lines, spaces, etc. Also, it shows in the graph format, how much this source is plagiarized. It also shows the list of files whose source code has been plagiarized. The GUI also has the "Download Report" button. By clicking it, the user can generate the report in PDF format, which has been downloaded onto his local computer. Figure (s) 10-14 shows the output windows of the GUI designed.

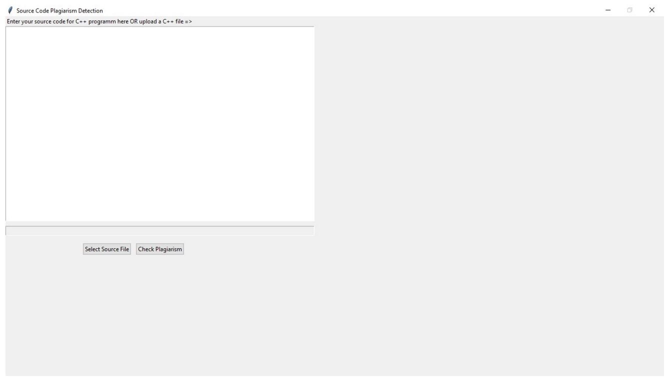
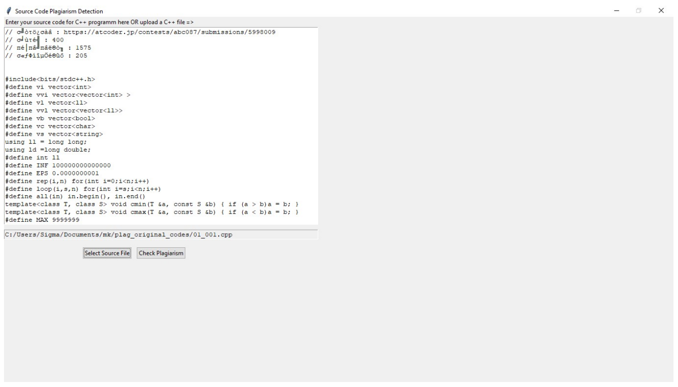

Source Code Plagiarism Detection
Enter your source code for C++ programm here OR upload a C++ file =>
Select Source File
Check Plagiarism

Figure 10. GUI Design (Screen 1)

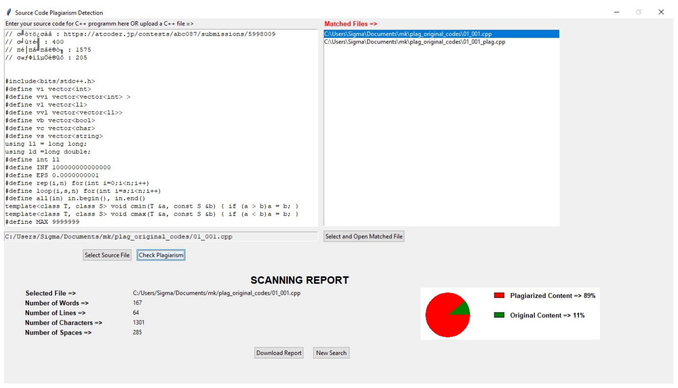

Source Code Plagiarism Detection
Enter your source code for C++ programm here OR upload a C++ file =>
Select Source File
Check Plagiarism

```
// σ╜òτö¿σàâ : https://atcoder.jp/contests/abc087/submissions/5998009
// σ╛ùτé╣ : 400
// πé│πâ╝πâëΘò╖ : 1575
// σ«ƒΦíîµÖéΘûô : 205


#include<bits/stdc++.h>
#define vi vector<int>
#define vvi vector<vector<int> >
#define vl vector<ll>
#define vvl vector<vector<ll>>
#define vb vector<bool>
#define vc vector<char>
#define vs vector<string>
using ll = long long;
using ld =long double;
#define int ll
#define INF 1000000000000000
#define EPS 0.0000000001
#define rep(i,n) for(int i=0;i<n;i++)
#define loop(i,s,n) for(int i=s;i<n;i++)
#define all(in) in.begin(), in.end()
template<class T, class S> void cmin(T &a, const S &b) { if (a > b)a = b; }
template<class T, class S> void cmax(T &a, const S &b) { if (a < b)a = b; }
#define MAX 9999999
```

C:/Users/Sigma/Documents/mk/plag_original_codes/01_001.cpp

Figure 11. Source code is uploaded in Text Input (Screen 2)


Source Code Plagiarism Detection
Enter your source code for C++ programm here OR upload a C++ file =>
Matched Files =>
Select and Open Matched File
Select Source File
Check Plagiarism
Download Report
New Search

```
// σ╜òτö¿σàâ : https://atcoder.jp/contests/abc087/submissions/5998009
// σ╛ùτé╣ : 400
// πé│πâ╝πâëΘò╖ : 1575
// σ«ƒΦíîµÖéΘûô : 205


#include<bits/stdc++.h>
#define vi vector<int>
#define vvi vector<vector<int> >
#define vl vector<ll>
#define vvl vector<vector<ll>>
#define vb vector<bool>
#define vc vector<char>
#define vs vector<string>
using ll = long long;
using ld =long double;
#define int ll
#define INF 1000000000000000
#define EPS 0.0000000001
#define rep(i,n) for(int i=0;i<n;i++)
#define loop(i,s,n) for(int i=s;i<n;i++)
#define all(in) in.begin(), in.end()
template<class T, class S> void cmin(T &a, const S &b) { if (a > b)a = b; }
template<class T, class S> void cmax(T &a, const S &b) { if (a < b)a = b; }
#define MAX 9999999
```

C:/Users/Sigma/Documents/mk/plag_original_codes/01_001.cpp

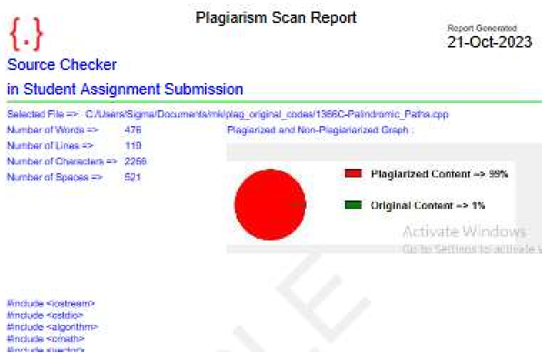
C:\Users\Sigma\Documents\mk\plag_original_codes\01_001.cpp
C:\Users\Sigma\Documents\mk\plag_original_codes\01_001_plag.cpp

SCANNING REPORT

Selected File =>	C:/Users/Sigma/Documents/mk/plag_original_codes/01_001.cpp
Number of Words =>	167
Number of Lines =>	64
Number of Characters =>	1301
Number of Spaces =>	285

🟥 Plagiarized Content => 89%
🟩 Original Content => 11%

Figure 12. GUI Screen with the report (Screen 3)



Plagiarism Scan Report

Report Generated
21-Oct-2023

Source Checker
in Student Assignment Submission

Selected File =>	C:/Users/Sigma/Documents/mk/plag_original_codes/1366C-Palindromic_Paths.cpp
Number of Words =>	476
Number of Lines =>	110
Number of Characters =>	2266
Number of Spaces =>	521

Plagiarized and Non-Plagiarized Graph :

🟥 Plagiarized Content => 99%
🟩 Original Content => 1%

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <cmath>
```

Figure 13. Sample PDF Report Generated

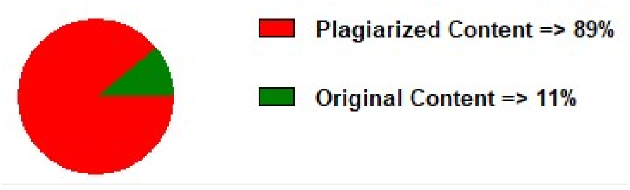


Figure 14. Graph result displayed in the report.

VI. CONCLUSION

When it comes to identifying instances of plagiarism in the source code, the k-means clustering technique has the highest accuracy rate of 99.2 %. On the other hand, the results it generates are superior to those generated by the random forest algorithm. For 90% and 80% of the training sets, respectively, the proposed system performed better than the MOSS system that was already in place. When comparing the findings, some evaluation criteria that are taken into consideration include precision, recall, and F-measure. The proposed model shows good results as compared to the reported results. This has also been experienced with the GUI implementation of the system.

References

[1] A. Ramírez-de-la-Cruz, G. Ramírez-de-la-Rosa, C. Sánchez-Sánchez, H. Jiménez-Salazar, C. Rodríguez-Lucatero, W. A. Luna-Ramírez, "High level features for detecting source code plagiarism across programming languages," *Proceedings of the FIRE Workshops*, 2015, pp. 10-14.

[2] G. Acampora and G. Cosma, "A fuzzy-based approach to programming language independent source-code plagiarism detection," *Proceedings of the 2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2015, pp. 1-8, https://doi.org/10.1109/FUZZ-IEEE.2015.7337935.

[3] J. Itsarawisut, K. Kanjanawanishkul, "Neural network-based classification of germinated hang rice using image processing," *IETE Technical Review*, vol. 36, issue 4, pp. 375-381, 2019, https://doi.org/10.1080/02564602.2018.1487806.

[4] A. Parker and J. O. Hamblen, "Computer algorithms for plagiarism detection," *IEEE Transactions on Education*, vol. 32, issue 2, pp. 337–343, 1989. https://doi.org/10.1109/13.28038.

[5] A. Iversen, N. K. Taylor, and K. E. Brown, "Classification and verification through the combination of the multi-layer perceptron and auto-association neural networks," *Proceedings of the International Joint Conference on Neural Networks*, Montreal, Canada, July 2005, pp. 1166–1171.

[6] S. Balakrishnama, & A. Ganapathiraju, "Linear discriminant analysis – A brief tutorial," *Institute for Signal and Information Processing*, vol. 11, pp. 1-8, 1998.

[7] C. Liu, C. Chen, J. Han, and P. S. Yu, "Gplag: detection of software plagiarism by program dependence graph analysis," *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006, pages 872–881. https://doi.org/10.1145/1150402.1150522.

View publication stats

[8] Chris L. Evaluating ML Models: Precision, Recall, F1 and Accuracy. 2019. [Online]. Available at: https://medium.com/analytics-vidhya/evaluating-ml-models-precision-recall-f1-and-accuracy-f734e9fcc0d3.
[9] C. Arwin and S. M. M. Tahaghoghi, "Plagiarism detection across programming languages," *Proceedings of the Twenty-Ninth Australasian Computer Science Conference (ACSC2006)*, 2006, vol. 48, pp. 277-286.
[10] D. Heres, *Source Code Plagiarism Detection using Machine Learning*, Master's thesis, Utrecht University, 2017, pp. 1-37.
[11] G. Biau, "Analysis of a random forests model," *Journal of Machine Learning Research*, vol. 13, pp. 1063–1095, 2012`.
[12] M. Ellis, et al., "Plagiarism detection in computer code," 2005, pp. 1-10. [Online]. Available at: http://www.rose-hulman.edu/class/csse/faculty-staff/csse-department/seniorTheses/Matt Ellis.pdf.
[13] V. Y. Kulkarni and P. K. Sinha, "Effective learning and classification using random forest algorithm," *International Journal of Engineering and Innovative Technology (IJEIT)*, vol. 3, issue 11, pp. 267–273, 2014.
[14] C. Goutte, E. Gaussier, "A probabilistic interpretation of precision, recall and f-score, with implication for evaluation," In: Losada, D.E., Fernández-Luna, J.M. (eds) Advances in Information Retrieval. ECIR 2005. Lecture Notes in Computer Science, vol 3408. Springer, Berlin, Heidelberg, 2005, pp. 345–359. https://doi.org/10.1007/978-3-540-31865-1_25.
[15] G. Guo, H. Wang, D. Bell, Y. Bi, K. Greer, (2003). KNN Model-Based Approach in Classification. In: Meersman, R., Tari, Z., Schmidt, D.C. (eds) On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE. OTM 2003. Lecture Notes in Computer Science, vol 2888. Springer, Berlin, Heidelberg, 2003, pp. 986–996. https://doi.org/10.1007/978-3-540-39964-3_62.
[16] J. A. W. Faidhi and S. K. Robimox, "An empirical approach for detecting program similarity and plagiarism within a university programming environment," *Pergamon Journals Ltd*, vol. 11, issue 1, pp. 11–19, 1987. ttps://doi.org/10.1016/0360-1315(87)90042-X.
[17] J. Hage, P. Rademaker, and N. van Vugt, "A comparison of plagiarism detection tools," *Utrecht University. Utrecht, The Netherlands*, no. 28, pp. 1-26, 2010.
[18] J. Ming, F. Zhang, D. Wu, P. Liu, and S. Zhu, "Deviation-based obfuscation-resilient program equivalence checking with application to software plagiarism detection," *IEEE Transactions on Reliability*, vol. 65, issue 4, pp. 1647–1664, 2016. https://doi.org/10.1109/TR.2016.2570554.
[19] J.-H. Ji, G. Woo, and H.-G. Cho, "A source code linearization technique for detecting plagiarized programs," *Proceedings of the ITiCSE'07*, Dundee, Scotland, United Kingdom, June 2007, pp. 73–77. https://doi.org/10.1145/1269900.1268807.
[20] K. S. Kim et al., "Comparison of k-nearest neighbor, quadratic discriminant and linear discriminant analysis in classification of electromyogram signals based on the wrist-motion directions," *Current Applied Physics*, vol. 11, issue 3, pp. 740–745, 2011. https://doi.org/10.1016/j.cap.2010.11.051.
[21] K. J. Ottenstein, "An algorithmic approach to the detection and prevention of plagiarism," Purdue University, Department of Computer Science Technical Reports, Report number 76-200, August 1976, 16 p.
[22] L. Prechelt, G. Malpohl, and M. Philippsen, "Finding plagiarisms among a set of programs with jplag," *Journal of Universal Computer Science*, vol. 8, no. 11, pp. 1016-1038, 2002.
[23] M. Schein and R. Paladugu, "Redundant surgical publications: tip of the iceberg?," *Surgery*, vol. 129, issue 6, pp. 655–661, 2001. https://doi.org/10.1067/msy.2001.114549.
[24] C. Manliguez, "Generalized confusion matrix for multiple classes," pp. 1-2, 2016, https://doi.org/10.13140/RG.2.2.31150.51523.
[25] M. Novak, M. Joy, and D. Kermek, "Source-code similarity detection and detection tools used in academia: A systematic review," *ACM Trans. Comput. Educ.*, vol. 19, issue 3, Article 27, pp. 1-37, 2019. https://doi.org/10.1145/3313290.
[26] M. Ďuračíka, E. Kršáka, and P. Hrkúta, "Current trends in source code analysis, plagiarism detection and issues of analysis big datasets," *Proceedings of the International Scientific Conference on Sustainable, Modern and Safe Transport*, 2017, pp. 136–141. https://doi.org/10.1016/j.proeng.2017.06.024.
[27] P. Flach, J. Hernández-Orallo, C. Ferri, "A coherent interpretation of AUC as a measure of aggregated classification performance," *Proceedings of the 28th International Conference on Machine Learning*, 2011, pp. 657-664.
[28] R. C. Lange and S. Mancoridis, "Using code metric histograms and genetic algorithms to perform author identification for software forensics," *Proceedings of the 9th ACM Annual Conference on Genetic and Evolutionary Computation (GECCO'07)*, New York, NY, USA, 2007, pp. 2082–2089. https://doi.org/10.1145/1276958.1277364.
[29] S. Engels, V. Lakshmanan, and M. Craig, "Plagiarism detection using feature-based neural networks," *ACM SIGCSE Bulletin*, vol. 39, pp. 34–38, 2007. https://doi.org/10.1145/1227504.1227324.
[30] S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: local algorithms for document fingerprinting," *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, 2003, pp. 76–85. https://doi.org/10.1145/872757.872770.
[31] U. Bandara and G. Wijayarathna, "A machine learning based tool for source code plagiarism detection," *International Journal of Machine Learning and Computing*, vol. 1, issue 4, pp. 337–343, 2011. https://doi.org/10.7763/IJMLC.2011.V1.50.
[32] A. L. Samuel, Arthur L (1959), "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 44, no. 1-2, pp. 210–229, 1959. https://doi.org/10.1147/rd.33.0210.
[33] H. Han & U. Chong, "Neural network based detection of drowsiness with eyes open using AR modelling," *IETE Technical Review*, vol. 33, issue 5, pp. 518-524, 2016. https://doi.org/10.1080/02564602.2015.1118362.
[34] K. Deergha Rao & D. C. Reddy, "Transputer implementation of the EKF-based learning algorithm for multilayered neural networks used in classification of EEG signals," *IETE Technical Review*, vol. 14, issue 3, pp. 177-182, 1997. https://doi.org/10.1080/02564602.1997.11416668.
[35] S. Koco and C. Capponi, "On multi-class classication through the minimization of the confusion matrix norm," *JMLR: Workshop and Conference Proceedings*, 2013, pp. 277–292.
[36] B. Martin, "Plagiarism: a misplaced emphasis," *Journal of Information Ethics*, vol. 3, issue 2, pp. 36-47, 1994.

RADDAM SAMI MEHSEN working for his doctoral research from Gujarat University, Gujarat. Graduated Middle Technical University, Technical Institute of Baqubah. Received bachelor from al Mustansiriyah University, and Master degree of Computer science, 2011, Dr. Babasaheb Ambedkar Marathwada University, Aurangabad, MH, India. Specialist in computer science, expert in many programming languages HTML, PHP, JavaScript, C ++, Visual Basic. Experience in project management and client relations. He has a good experience in designing programs and databases for people and institutions, correcting their mistakes, and making modification operations.

HIREN D. JOSHI Professor in Computer Science, Gujarat University, Gujarat. He is life member of CSI, Member of ACM and ISTE. His research interests are IoT, Machine Learning, NLP, Artificial Intelligence and Image Processing. He has published more than 30 research articles in various International Journals and presented various research themes in conferences. He is editorial member for various international journals and conferences.