Source Code Plagiarism Detection using Random Forest Classifier

Raddam Sami Mehsen[1], Hiren D. Joshi (PhD) [2]

[1, 2] Department of Computer Science, Gujarat University Ahmadabad, Gujarat, India
[1]raddamsami@gujaratuniversity.ac.in [2]hdjoshi@gujaratuniversity.ac.in

***Abstract*— Plagiarism Detection Systems are particularly useful in identifying plagiarism in the educational sector, where scientific publications and articles are common. Plagiarism occurs when someone replicates a piece of work without permission or citation from the original creator. Because of the advancement of communication and information technologies (ICT) and the accessibility of scientific materials on the internet, plagiarism detection has become a top priority and due to the broad availability of freeware text editors, detecting source code plagiarism has become a big difficulty. There have already been several research on the many forms of plagiarism detection algorithms used in identification systems, as well as source code plagiarism detection. This work suggests a strategy that combines TF-IDF transformations with a Random Forest Classifier to achieve a 93.5 percent accuracy rate, which is high when compared to previous strategies. The suggested system is implemented using the Python programming language.**

***Index Terms*—Source code, C++ programming language, plagiarism, machine learning.**

I. Introduction

According to Martins et al. (2014) plagiarism is the use of work without crediting its authors. As a result of easy and inexpensive access to enormous amounts of web content, plagiarism has become a severe problem for academics, publishers, and educational institutions. One of the most common types of plagiarism in academia is textual plagiarism at the document level, with papers often consisting of essays, reports, and scientific articles. According to a recent assessment (M. Schein 2001), 16% of original publications published in major surgical journals are potentially redundant. Plagiarism, on the other hand, is defined as the copying of text documents as well as source code. Plagiarism in source code is defined as attempting to pass off another's source code as one's own without disclosing which sections are copied from which author (J. Hage 2010; Karl J . Ottenstein. 1976). Plagiarism in source code is widespread in academic programming assignments (Christian Arwin 2006). Students commonly attempt to replicate assignments from their friends in order to get good grades with little effort. Plagiarism is more likely to occur in following classes for freshmen who plagiarise in their first course. As a result, this illegal behaviour must be halted immediately. A course instructor may receive incorrect feedback about the course's level and students' performance. As a result, detecting plagiarism in coursework is an important task (J. A. W. Faidhi 1987). Manually reviewing and recognising comparable student provided solutions to establish whether a submission is genuine or plagiarised is difficult in a large class. Manual inspection is time-consuming and inefficient. Using automated code comparison tools like Measure of Software Similarity (MOSS) is one option (S. Schleimer 2003) and JPlag to find similar submission pairings (L. Prechelt 2002). Most well-known automatic code comparison algorithms identify plagiarism primarily through utilizing characteristics based solely on the syntactic aspects of the assignments or using a text-based approach. However, both of these methods are vulnerable to code obfuscation (J. Ming 2016), which is a significant obstacle to automatic software plagiarism detection. Students frequently utilize fraudulent tactics to obscure the code and avoid detection. Such deceptive practice is discussed in Section III of this paper.

II. Related work

Researchers used programme similarity metrics such as MOSS (S. Schleimer 2003), JPlag (L. Prechelt 2002), and others to detect plagiarism in previous work. Most well-known automatic code comparison algorithms, on the other hand, use a text-based method to detect plagiarism or features based on assignment properties at a syntactic level to detect plagiarism.

MOSS is built on syntactic assignments' winnowing property, which is a local fingerprinting strategy (S. Schleimer 2003). MOSS's fingerprint selection mechanism, which in a window, selects the fingerprint with the lowest value, isn't extremely exact. A search for the longest common sequence is performed on top of this fingerprint. JPlag is another popular tool for detecting plagiarism (L. Prechelt 2002). JPlag finds the tokenized form of source code's longest frequent sequences for each pair of submissions using greedy string tiling. JPlag is similar to MOSS in terms of how it operates. By concentrating on common tokenized structural blocks, JPlag, on the other hand, misses numerous key characteristics when comparing given code, such as formatting and style. Other methods for detecting plagiarism have been used, such as analyzing programme dependency graphs.

The authors of (C. Liu 2006) presented GPlag, a new plagiarism detection approach that mines programme dependency graphs to detect plagiarism (PDGs). A PDG graphically displays a procedure's data and control dependency. GPlag is more successful at identifying plagiarism than PDGs since PDGs are largely unchanging during the plagiarism process.

The authors of (A. Ram'rez-de-la Cruz 2015) suggest a representation of source code pairings based on lexical, stylistic, comments, programmer's text, and structure aspects. They convey lexical, comment, and n-grams are a collection of characters used by programmers in source code. Rather than detecting a specific programming language, these qualities are aimed to detect aspects of normal language that programmers leave behind.

Course instructors can utilise a technique based on assignment attributes devised by researchers in (S. Engels 2007) to compare two entries. They provided 12 features, including comment similarity, white-space, and others, including the MOSS similarity score. When performing evaluations, MOSS and JPlag, on the other hand, both filter these attributes out. Their primary purpose is to detect plagiarism using these characteristics as signals. To assess the importance of each feature in the evaluation, this system uses neural network algorithms. They are, however, primarily concerned with locating plagiarized pairs inside a single issue set. Our recommended remedy, on the other hand, is unaffected by the issue set.

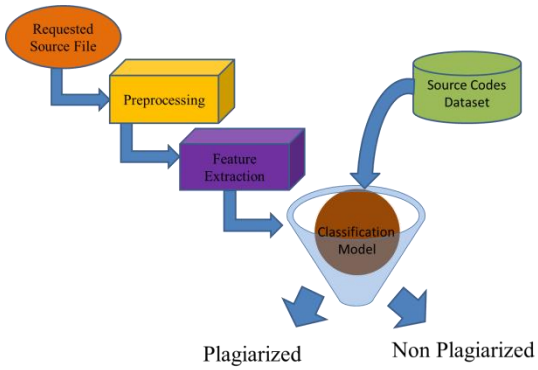

Fig. 1. Typical Working model for source code plagiarism detection

Not only may such systems be used to detect plagiarism in student assignments, but they can also be used in the following domains (Robert Lange 2007):

• Criminal Prosecution - Finding the creator of malware that belongs to the suspects.

• Corporate Litigation — In the event that an employee violates a contract's no-compete clause, the author of the code must be identified.

• Plagiarism Detection - Identifying the author in cases of academic misconduct.

Plagiarized programmes are those that have been copied from other programmes and have had minor changes made to them. Plagiarism can occur in a variety of contexts, including (Mark C. Johnson et al. 2004; Matija Novak 2019):

• Students' willingness to share the code with other students if the deadline for the project is approaching.

• When students collaborate on a project, they may submit programmes that are identical yet differ in nomenclature and structure.

• Shared printers and workstations can be used to steal assignments.

• Only the specs are changed from one semester to the next, thus programmes from prior semesters can be used.

• Small assignments, which are not plagiarised, appear to be identical because numerous students independently arrive at the same design.

III. Proposed Approach

The steps involved in the proposed system are as follows:

i. Take a look at the C++ source code files.
ii. Source codes should be gathered and preprocessed.
iii. Create a matrix of term vs. file by generating tokens for each file, joining them, and creating a matrix of term vs. file. This step is also can be referred as calculation of TF-IDF files.
iv. Read the source code for the query in C++.
v. Return to the third step.
vi. Go back to step 4
vii. For each file using the query source file, find similarity using the Random Forest Classifier.
viii. The file is suspicious if the similarity exceeds the threshold; else, the file is innocuous.
ix. If plagiarism is identified, a report should be generated. Otherwise, return to step 1 if necessary.

The proposed model, however, is shown in Fig. 2.

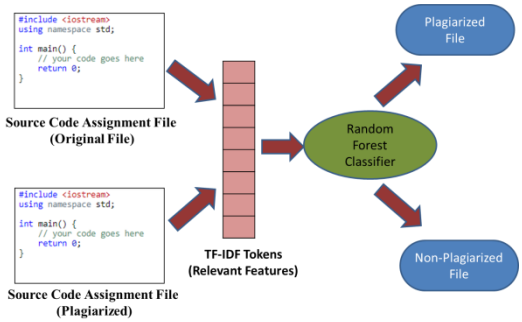

Fig. 2. Proposed model for source code plagiarism detection, where features were extracted from requested files using TFIDF tokens and then trained using random forest classifier.

A. Code Metrics

It extracts the characteristics from the input files when the source codes are submitted to the proposed system. TFIDF code metrics are used as selected features in the proposed model. Code metrics are nothing more than a collection of tokens taken from source code files, in this example, student assignments. These tokens describe the features of a programmer's programmes and are used to assess their effectiveness, style, system costs, reliability, adaptability, and structure (Jeong-Hoon Ji 2007; Ellis, et. al. 2005). To extract these code metrics, a variety of tools and methodologies have been proposed, including:

• N-grams - are a set of n objects taken from a text corpus, such as words, letters, and so on. Documents are separated into a list of substrings of length n with their frequencies using

http://doi.org/10.24086/cocos2022/paper.732

this method (Michal uraka, et al. 2017). It's a Natural Processing Language notion. United Kingdoms, for example, are a 2-gram.

• TF-IDF - Term Frequency-Inverse Document Frequency is what it stands for (Alan Parker 1989). It gives words in a document weights and calculates their relevance in the corpus. It is mostly used for text mining and retrieval of information.

• ANTLR - is a parser generator that aids in programming language reading and processing. It also extracts vocabulary and syntax, which aids with the comprehension of source code structure.

To avoid being discovered, students use numerous changes in their projects. These source code changes can take many different forms, some of which are listed below (C. Liu, 2006; Alan Parker 1989):

• Format Alteration –

This is merely the insertion and removal of blank spaces and remarks.

• Identifier Renaming –

Changing the names of identifiers is another typical plagiarism technique because it does not violate the program's validity.

• Reordering Statements —

Statements that do not have any sequential dependencies, such as declarations, can be simply reordered.

• Control Replacement –

In programming languages, there are numerous substitutes for simple codes, such as while loops for loops, if else conditions for the same logic, and so on (Balakrishnama, S. 1998).

• Code Insertion - Codes which do not modify the original logic of the programme can be injected to mask plagiarism.

At present the proposed model doesn't operate on such transformations. The random forest classifier, which is one of the supervised classifiers used in machine learning approaches, was used to train the retrieved features.

B. Machine Learning Phenomena

The study and development of algorithms that can learn and predict data is known as machine learning. Such algorithms build a model using sample inputs rather than following rigid static programme instructions in order to produce data-driven predictions or judgments. When designing and implementing explicit algorithms is too expensive, it's used for a range of computing jobs (Acampora, 2015). Machine learning technology is used in many parts of contemporary culture, from web searches to content filtering on social media platforms to e-commerce site recommendations. Machine learning has become so common that we probably use it millions of times each day without realising it (Upul Bandara 2011). We now have useful speech recognition systems, fast web searches, self-driving cars, and a much greater grasp of the human genome thanks to machine learning. Machine learning tasks are usually classified into three groups based on the sort of feedback delivered to the learning system. To detect source code copying, machine learning approaches can be utilised (Daniel Heres 2017).

The following are the details:

1) Supervised learning:

An instructor provides sample inputs and outputs to the computer. The goal is to come up with a universal rule or function that connects inputs and outputs. This category includes classification (Alexander Iversen, 2005) and regression.

2) Unsupervised learning:

Because the learning algorithm isn't provided labels, it must deduce the data's structure on its own. Clustering algorithms are included in this class.

3) Reinforcement learning:

In a dynamic environment, computer software must complete a task, like as playing a game against an opponent, without being told of its success. The software receives feedback in the form of incentives and punishments as it navigates its issue area.

C. Random Forest Classifier

The Random Forest algorithm is a popular supervised learning machine learning technique. In machine learning, it can be used to solve difficulties in classification and regression (Gerard Biau 2012; Vrushali Y Kulkarni 2014). It is based on the notion of ensemble learning, which is the process of integrating numerous classifiers to solve a complex problem and increase the performance of the model (Gerard Biau 2012; Vrushali Y Kulkarni 2014).

As depicted in Figure 1, the Random Forest classifier "contains a number of decision trees on various subsets of a given dataset and takes the average to increase the dataset's predictive accuracy." The random forest, rather than employing a single decision tree, uses the forecasts from each tree to predict the final output based on the majority vote of predictions. The more trees in the forest, the higher the accuracy and the smaller the chance of overfitting (Gerard Biau 2012; Vrushali Y Kulkarni 2014).

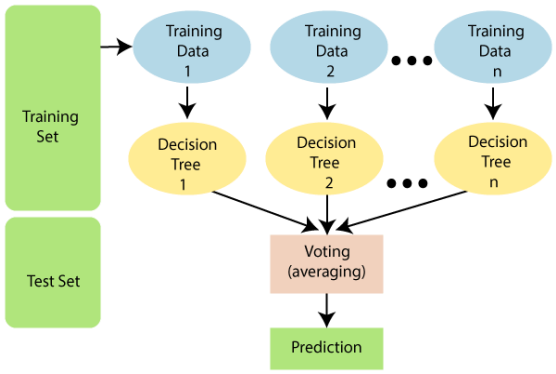

Fig. 3. Working of Random Forest Classifier

D. Mathematical Formulation of Random Forest Classifier

Random Forest is a machine learning ensemble classification algorithm. It creates a number of decision trees and uses majority voting to integrate the results of these trees (Gerard Biau 2012; Vrushali Y Kulkarni 2014). In this approach, randomization is accomplished in two ways: data sampling is done at random for bootstrap samples, and input variables are chosen at random for the creation of

http://doi.org/10.24086/cocos2022/paper.732

individualised decision trees (Gerard Biau 2012; Vrushali Y Kulkarni 2014).

The following steps can be used to create decision trees in this algorithm:

1. Bootstrap sampling is a method of sampling records at random with replacement from the original data. This is the tree-generation training set.

2. m variables are chosen at random from this data and kept constant until the tree is fully formed and the best split is used to split the node.

3. The parameters Ntree and node size can be used to determine the number of trees generated and the depth of the trees. Once the forest of decision trees has been created, test data is run through each one, yielding a classification (Divakar Kapil 2019).

The classification with majority votes (J. A. W. Faidhi 1987) is allocated to the test data after the results from each tree are pooled. Some data is left out during bootstrap sampling, which is known as out-of-bag data and is used to estimate individual tree error. This can be calculated using the following formula:

$$PE* = P_{x,y}\,(mg\,(X,Y)) < 0$$

The margin function has the following formula:

$$mg\,(X, Y) = av_k\, I(h_k\,(X) = Y) - max_{j \neq Y}\, av_k\, I(h_k\,(X) = j)$$

Random Forest strength is calculated using the following formula:

$$S = E_{X,\,Y}\,(mg\,(X, Y))$$

This technique has several advantages, including the ability to handle vast amounts of data, the ability to give unbiased generalisation error estimates, the ability to estimate missing data effectively, and the ability to retain accuracy (Gerard Biau 2012; Vrushali Y Kulkarni 2014).

IV. Experimental Results and Performance Evaluation

Different assessment measures are used to assess the suggested model. For each iteration and overall system, these metrics include cross-value score, accurate response rate, recall, F1 score, precision, and other confusion matrices, which are detailed below:

A. Classification Accuracy

It is the most widely used statistic for evaluating classification tasks. The ratio of correct forecasts to all predictions made can be described as accuracy (Chris L. 2019; Peter Flach, 2011).

$$\text{Accuracy} = \text{Number of Correct Predictions/Total Number of Predictions}$$

The accuracy formula for binary classification is as follows:

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN)$$

Where, True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) are the acronyms for True Positives, True Negatives, False Positives, and False Negatives, respectively.

However, in the situation of unequally distributed classes, it is not a fair measure to optimize because it might be relatively high, favoring the majority while ignoring the minority.

B. Confusion Matrix:

A confusion matrix (Manliguez 2016) is a N x N matrix used to assess the effectiveness of a classification model, with N denoting the number of target classes. The matrix compares the actual goal values to the machine learning model's predictions (Sokol Koco 2013).

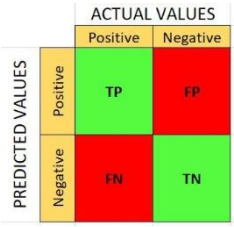

Fig. 2. Binary classification confusion matrix

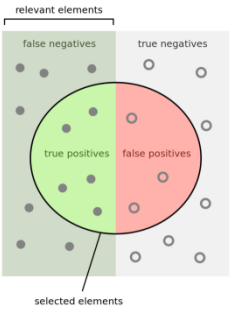

Fig. 3. Cluster representation of confusion matrix

C. F-Measure

F-Measure calculates the score by taking the precision and recall into account when measuring the correctness of the test data. It basically works according to TP - True Positives, TN - True Negatives, FP - False Positives and FN - False Negatives as shown in Figure 3. Precision is defined as the number of real positive findings divided by the total number of positive predictions (Chris L. 2019). (Goutte C., 2005).

$$p = TP/TP + FP$$

Recall, on the other hand, is defined as the number of correct predictions divided by the total number of correct positives (Chris L. 2019).

$$r = TP/TP + FN$$

The F-Score is calculated by taking the weighted average of precision and recall (Goutte C., 2005).

http://doi.org/10.24086/cocos2022/paper.732

F = 2*((precision * recall)/(precision +recall))

F1-Score or F-value can also be calculated as

F1-Score= Harmonic_mean(precision, recall)

A set of assignment solutions submitted by students in a C++ programming competition for novices using Python are used to evaluate the proposed system's performance (Sambit Mahapatra. 2018). It consists of 44 entries, each containing ten issues that are specific to that entry. Each student can submit up to 20 contributions, including both the original and revised versions of their work. In total, there are about 880 entries, culminating in this. The following are some sample code snippets from the dataset:

Figure 4 shows the original file's code snippet for a C++ programme.

```
#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;
}
```

Fig. 4. Source code for non-plagiarized file example program 1

Figure 5 shows the generated characteristics for the sample programme in Figure 4.

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

Fig. 5. Tokenized TF-IDF file for non-plagiarized file example program 1

Figure 6 illustrates a code fragment of plagiarised C++ source code:

```
#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;
}
```

Fig. 6. Source code for plagiarized file example program 1

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

Fig. 7. Tokenized TF-IDF file for plagiarized file example program 1

Figure 7 shows the produced features for the sample plagiarized programme in Figure 6.

The brief details about the source files used in the experiments are given in Table I.

TABLE I
DETAILS ABOUT DATABASE

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

TABLE II
MEASURES FOR ITERATION 0 TO 5 USING RANDOM FOREST ALGORITHM

Dataset	cross-val score	Correct answer rate	Recall	F-value
0	0.91	0.94	0.88	0.93
1	0.91	0.93	0.90	0.92
2	0.92	0.95	0.98	0.95
3	0.92	0.95	0.93	0.95
4	0.91	0.93	0.91	0.93
5	0.91	0.95	0.92	0.95

Table II and III given a performance evaluation metrics for a 5 different iterations generated using random forest classifier.

http://doi.org/10.24086/cocos2022/paper.732

TABLE III
CONFUSION MATRIX (A) – (F) USING RANDOM FOREST ALGORITHM FOR ITERATION 0 TO 5

	Positive	Negative
Positive	76 (TP)	1 (FP)
Negative	10 (FN)	89 (TN)

(A)

	Positive	Negative
Positive	77 (TP)	4 (FP)
Negative	9 (FN)	86 (TN)

(B)

	Positive	Negative
Positive	84 (TP)	6 (FP)
Negative	2 (FN)	84 (TN)

(C)

	Positive	Negative
Positive	80 (TP)	3 (FP)
Negative	6 (FN)	87 (TN)

(D)

	Positive	Negative
Positive	78 (TP)	4 (FP)
Negative	8 (FN)	86 (TN)

(E)

	Positive	Negative
Positive	79 (TP)	1 (FP)
Negative	7 (FN)	89 (TN)

(F)

The confusion matrix for the entire system is shown in tables IV and V below, however the Figure 8 shows resultant learning and performance curve (Guo G., 2003):

TABLE IV
CONFUSION MATRIX FOR PROPOSED SYSTEM

	Positive	Negative
Positive	72 (TP)	3 (FP)
Negative	14 (FN)	87 (TN)

TABLE V
CONFUSION MATRIX FOR PROPOSED SYSTEM IN PERCENTAGE

	Positive	Negative
Positive	0.44% (TP)	0.02% (FP)
Negative	0.05% (FN)	0.49% (TN)

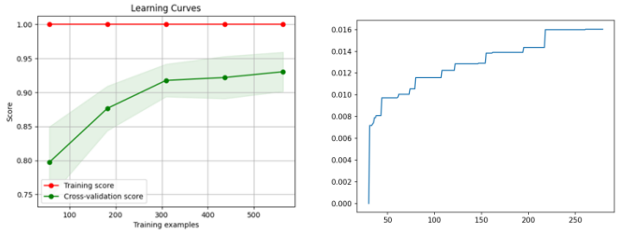

Fig. 8. the resultant learning curve for training instances as a function of score and cluster to performance curve

After the generation of the confusion matrix, the following performance evaluation metrics were computed and compared. The proposed system compared with existing MOSS system for 90% and 80% training set and found to be superior with 93.5% accuracy (Kang Soo Kim et al. 2011), as the existing approaches haven't reported the accuracy, we have again compared the proposed model using other evaluation metrics such as precision, recall and F1-score and found to be superior as these metrics are shown in Table VI:

TABLE VI
COMPARISON OF PERFORMANCE OF PROPOSED METHOD WITH MOSS AT VARIOUS THRESHOLDS

Measure	Ours	MOSS (90%)	MOSS (80%)
Accuracy	0.935	-	-
Precision	0.960	1.000	0.920
Recall	0.906	0.631	0.819
F1 Score	0.931	0.773	0.866

V. FUTURE SCOPE

The approach's scope involves creating a term-by-file matrix and a query vector from the source code corpus. We calculate the degree of similarity between the query vector and the source code files after we've found this matrix. Finally, we generate a plagiarism report that includes a percentage of files that are considered suspect. The proposed system uses Python based machine learning algorithm to implement tf-idf token files followed by the random forest classifier. The system is concerned with the code, and the input files are only be in C++. Colleges, academic institutions, and the information technology sector can all benefit from the system. It's also appropriate for usage in research centres. The system will be valuable in any industry that requires innovation, but it must be careful not to use duplicate data. The system could improve in future extensions of the work with improved deep learning algorithms.

CONCLUSION

The problem of source-code plagiarism is becoming increasingly difficult to solve as the internet and the number of websites grows. It is both crucial and necessary to work efficiently to improve the detection process's speed and accuracy in order to detect programming plagiarism. With a 93 percent accuracy rate, the suggested system employing the random forest algorithm detects source code plagiarism. The proposed system was further evaluated in comparison to other current source code plagiarism detection methods and found to be effective. Using k-means clustering and an unsupervised learning approach, this work will be expanded.

REFERENCES

A. Ram´ırez-de-la Cruz, G. Ram´ırez-de-la Rosa, C. Sanchez-S ´ anchez, H. Jim ´ enez-Salazar, ´ C. Rodr´ıguez-Lucatero, and W. A. Luna-Ram´ırez

(2015) “High level features for detecting source code plagiarism across programming languages,” In FIRE Workshops, pages 10–14, 2015.

C. Liu, C. Chen, J. Han, and P. S. Yu (2006) “Gplag: detection of software plagiarism by program dependence graph analysis,” In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 872–881. ACM, 2006.

J. Hage, P. Rademaker, and N. van Vugt (2010) “A comparison of plagiarism detection tools,” Utrecht University. Utrecht, The Netherlands, 28, 2010.

J. Ming, F. Zhang, D. Wu, P. Liu, and S. Zhu (2016) “Deviation-based obfuscation-resilient program equivalence checking with application to software plagiarism detection,” IEEE Transactions on Reliability, 65(4):1647–1664, 2016.

L. Prechelt, G. Malpohl, and M. Philippsen (2002) “Finding plagiarisms among a set of programs with jplag,” J. UCS, 8(11):1016, 2002.

M. Schein and R. Paladugu (2001), “Redundant surgical publications: tip of the iceberg?,” Surgery, 129(6):655–661, 2001.

S. Engels, V. Lakshmanan, and M. Craig (2007) “Plagiarism detection using feature-based neural networks,” In ACM SIGCSE Bulletin, volume 39, pages 34–38. ACM, 2007

S. Schleimer, D. S. Wilkerson, and A. Aiken (2003), “Winnowing: local algorithms for document fingerprinting,” In Proceedings of the 2003 ACM SIGMOD international conference on Management of data, pages 76–85. ACM, 2003.

Samuel, Arthur L (1959), “Some Studies in Machine Learning Using the Game of Checkers,” IBM Journal of Research and Development 44:1.2 (1959): 210–229.

Acampora, Giovanni & Cosma, Georgina. (2015). A Fuzzy-based Approach to Programming Language Independent Source-Code Plagiarism Detection. 10.1109/FUZZ-IEEE.2015.7337935.

Christian Arwin and S.M.M. Tahaghoghi (2006) “Plagiarism Detection across Programming Languages”. In: Twenty-Ninth Australasian Computer Science Conference (ACSC2006) 48 (2006).

Balakrishnama, S. & Ganapathiraju, Aravind. (1998). Linear Discriminant Analysis—A Brief Tutorial. 11.

Upul Bandara and Gamini Wijayarathna (2011) “A Machine Learning Based Tool for Source Code Plagiarism Detection”. In: International Journal of Machine Learning and Computing 1.4 (Oct. 2011), pp. 337–343.

Gerard Biau (2012) “Analysis of a Random Forests Model”. In: Journal of Machine Learning Research 13 (Apr. 2012), pp. 1063–1095.

Michal Ďuračíka, Emil Kršáka*, and Patrik Hrkúta (2017) “Current trends in source code analysis, plagiarism detection and issues of analysis big datasets”. In: International scientific conference on sustainable, modern and safe transport, pp. 136–141.

Ellis, Matt & Anderson, Claude (2005) Plagiarism Detection in Computer Code.

J. A. W. Faidhi and S. K. Robimox (1987) “An Empirical Approach For Detecting Program Similarity And Plagiarism Within A University Programming Environment”. In: Pergamon Journals Ltd 11.1 (Mar. 1987), pp. 11–19.

Peter Flach, Jose Hernandez-Orallo, and Cesar Ferri (2011) “A Coherent Interpretation of AUC as a Measure of Aggregated Classification Performance”. In: Proceedings of the 28th International Conference on Machine Learning (2011).

Manliguez, Cinmayii. (2016). Generalized Confusion Matrix for Multiple Classes. 10.13140/RG.2.2.31150.51523.

Goutte C., Gaussier E. (2005) 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. https://doi.org/10.1007/978-3-540-31865-1_25

Guo G., Wang H., Bell D., Bi Y., Greer K. (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. https://doi.org/10.1007/978-3-540-39964-3_62

Daniël Heres (2017) Source Code Plagiarism Detection using Machine Learning . Master's thesis.

Alexander Iversen, Nicholas K. Taylor, and Keith E. Brown (2005) “Classification and Verification through the Combination of the Multi-Layer Perceptron and Auto-Association Neural Networks”. In: Proceedings of International Joint Conference on Neural Networks, Montreal, Canada, (July 2005), pp. 1166–1171.

Jeong-Hoon Ji, Gyun Woo, and Hwan-Gue Cho (2007) “A Source Code Linearization Technique for Detecting Plagiarized Programs”. In: ITiCSE’07, Dundee, Scotland, United Kingdom (June 2007), pp. 73–77.

Divakar Kapil (2019) Stochastic vs Batch Gradient Descent. (2019). url: https://www.quora.com/Howcan-we-use-KNN-Machine-Learning-for-classification-of-cars.

Kang Soo Kim et al. (2011) “Comparison of k-nearest neighbor, quadratic discriminant and linear discriminant analysis in classification of electromyogram signals based on the wrist-motion directions”. In: www.elsevier.com/locate/cap (2011), pp. 740–745.

Sokol Koco and Cecile Capponi. (2013) “On multi-class classication through the minimization of the confusion matrix norm”. In: JMLR: Workshop and Conference Proceedings (2013), pp. 277– 292.

Vrushali Y Kulkarni and Pradeep K Sinha. (2014) “Effective Learning and Classification using Random Forest Algorithm”. In: International Journal of Engineering and Innovative Technology (IJEIT) 3.11 (May 2014), pp. 267–273.

Chris L. (2019) Evaluating ML Models: Precision, Recall, F1 and Accuracy. 2019. url: https : / / medium . com / analytics - vidhya / evaluating - ml - models - precision - recall - f1 - and - accuracy-f734e9fcc0d3.

Robert Lange and Spiros Mancoridis. (2007) “Using Code Metric Histograms and Genetic Algorithms to Perform Author Identification for Software Forensics”. In: GECCO’07 (July 2007).

Sambit Mahapatra. (2018) Linear Discriminant Analysis using Python. 2018. url: https://medium. com/journey- 2- artificial- intelligence/lda- linear-discriminant- analysis- usingpython-2155cf5b6398.

Matija Novak, Mike Joy, And Dragutin Kermek. (2019) “Source-code Similarity Detection and Detection Tools Used in Academia: A Systematic Review”. In: ACM Trans. Comput. Educ 19.3 (May 2019).

Karl J . Ottenstein. (1976) “An Algorithmic Approach To The Detection And Prevention Of Plagiarism”. In: CSD-TR 200 (Aug. 1976), pp. 30–41.

Alan Parker and James O. Hamblen. (1989) “Computer Algorithms for Plagiarism Detection”. In: IEEE TRANSACTIONS ON EDUCATION 32.2 (May 1989), pp. 337–343.