Journal of Techniques, ISSN: 2708-8383, Vol. 6, No. 2, 2024, Pages 18-28
DOI: https://doi.org/10.51173/jt.v6i2.1851

JOURNAL OF TECHNIQUES

Journal homepage: http://journal.mtu.edu.iq

RESEARCH ARTICLE – COMPUTER ENGINEERING

Detecting Source Code Plagiarism in Student Assignment Submissions Using Clustering Techniques

Raddam Sami Mehsen[1], Majharoddin M. Kazi[2*], Hiren Joshi[1]

[1]Department of Computer Science, Gujarat University, Ahmedabad, Gujarat, India

[2]Bill Gates College of Computer Science & Management, Osmanabad, Maharashtra, India

[*] Corresponding author E-mail: majharoddinkazi@billgatescollege.in

Article Info.

Article history:

Received
02 August 2023

Accepted
12 January 2024

Publishing
30 June 2024

Abstract

In pragmatic courses, graduate students are required to submit programming assignments, which have been susceptible to various forms of plagiarism. Detecting counterfeited code in an academic setting is of paramount importance, given the prevalence of publications and papers. Plagiarism, defined as the unauthorized replication of written work without proper acknowledgment, has become a critical concern with the advent of information and communication technology (ICT) and the widespread availability of scholarly publications online. However, the extensive use of freeware text editors has posed challenges in detecting source code plagiarism. Numerous studies have investigated algorithms for revealing different types of plagiarism and detecting source code plagiarism. In this research, we propose an innovative strategy that combines TF-IDF (Term Frequency-Inverse Document Frequency) modifications with K-means clustering, achieving a remarkable precision rate of 99.2%. Additionally, we explore the hierarchical clustering method, which estimates an even higher precision rate of 99.5% compared to previous techniques. To implement our approach, we utilize the Python programming language along with relevant libraries, providing a robust and efficient system for source code plagiarism detection in student assignment submissions.

Publisher: Middle Technical University

Keywords: Source Code; C++ Programming Language; Python; Plagiarism; Machine Learning.

1. Introduction

Plagiarism is a major issue for academics, authors, and educational organizations because is so simple and inexpensive to obtain so much internet content. Textual plagiarism represents one of the most prevalent forms of copyright infringement in academics because papers typically incorporate essays, reports, and scientific articles. According to [1, 2], 16% of the originally published papers in major surgical journals may be duplicated. Plagiarism, on the other hand, is defined as the copying of written materials and computer code. Source code plagiarism is defined as attempting to pass off another person's source code as one's own while omitting to recognize which precise sections were copied from which author [3]. Plagiarism in source code is common in academic programming coursework [4]. To swiftly get good grades, students commonly try to replicate other people's work. Plagiarism is more likely to occur in following classes for first-year students who copied in their first course. As a result, this illegal behavior must be terminated immediately. A course lecturer may receive erroneous feedback about the course's complexity and the performance of students. As a result, detecting plagiarism in coursework is critical [5]. In a large class, personally evaluating and identifying similar student-provided solutions to establish if a submission is genuine or plagiarized may be difficult. Inspection by hand is time-consuming and inefficient. Use automated code comparison tools like Measure of Software Similarity (MOSS) [6] and JPlag to uncover submission combinations that are identical to one another. The most widely recognized automatic code comparing techniques discover plagiarism by concentrating just on the submissions' textual or grammatical components. The sensitivity of both of the aforementioned methods to code obfuscation [7] creates a considerable hurdle to software-assisted plagiarism detection. Students employ deceptive tactics to obscure the code and avoid discovery. There are circumstances where students can fall into the plagiarism occurrences [16], such as:

- The student's readiness to distribute the code to other classmates as the due date for the project draws near.
- When completing a team project, individuals may submit source codes that are identical in every way but vary in vocabulary and style.
- Because only the requirements change from one academic year to the next, prior semester's programs can be utilized.
- Assignments are susceptible to theft via sharing workstations and printers.
- Why in small assignments, numerous learners independently come up with the same layout; this is not plagiarism.

This research contributes the source code plagiarism detection using different AI and machine learning approaches. In this Section, I introduce the plagiarism and it's some of the circumstances where plagiarism can happen in students assignments. Section II highlights some of the existing systems and approaches used by different researchers. Section III gives brief details about the proposed methodologies such as k-means

Nomenclature & Symbols			
ICT	Information and Communication Technology	ANTLR	ANother Tool for Language Recognition
TF-IDF	Term Frequency-Inverse Document Frequency	TP	True Positive
MOSS	Measure of Software Similarity	TN	True Negative
AI	Artificial Intelligence	FP	False Positive
PDG	Programme Dependence Graphs	FN	False Negative

clustering and hierarchical clustering. Section IV gives results and discusses accuracy measures of different methodologies, However, Section V concludes this research by stating the accuracy rate achieved from the proposed methodologies.

Fig. 1 shows a typical model for the classification of data based on features extracted. In this research, we have proposed a novel model for classification that classifies the input source code as plagiarized or non-plagiarized.

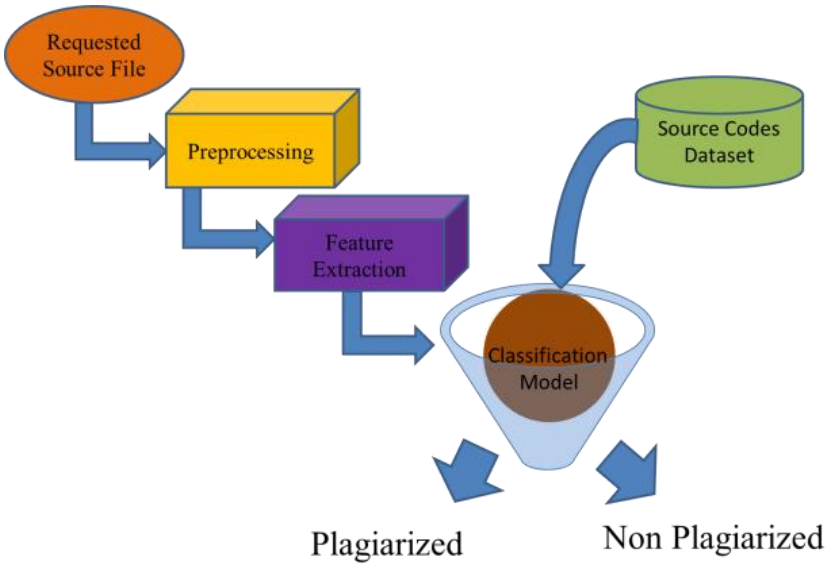

Fig. 1. A common process for identifying plagiarism in source code

2. Existing Systems and Approaches

Scholars used program similarity metrics such as MOSS [8, JPlag [9], and others to detect plagiarism. To identify plagiarism, a great deal of well-known automatic code comparison algorithms use features that consider assignment attributes at the grammatical level or a text-based method.

MOSS is built on the inspecting characteristic of syntactic assignments, which is an algorithm for local fingerprinting [10]. The MOSS fingerprint-selecting approach, which selects the lowest possible fingerprint score in a window, is not very exact. The lengthiest common sequence is looked after in addition to this fingerprint. The JPlag [11] is another popular tool for detecting plagiarism. Using greedy string tiling, JPlag determines the tokenized form of the longest frequent sequences between each pair of submissions. JPlag is similar to MOSS in terms of capability. When evaluating a given code, JPlag, on the other hand, skips many crucial elements due to its prioritization of common tokenized structural blocks, which include formatting and style.

Research reported in [12] proposed GPlag, a novel method for plagiarism detection that mines programme dependence graphs (PDGs). A PDG displays a procedure's data and control relationships. Because PDGs frequently stay constant throughout the plagiarism process, GPlag is more accurate than PDGs at identifying plagiarism.

Authors in [13] stated that lexical, stylistic, comment, programmer's text, and structure factors should be used to describe source code pairs. A set of symbols is used by programmers to represent lexical, comment, and n-gram information in source code. Instead of identifying a specific programming language, these traits are meant to spot ordinary language aspects that programmers overlook.

Using a technique devised by [14] utilizing assignment features, course instructors may contrast two submissions. The MOSS similarity score, white space, and comment similarity were among the 12 characteristics that they incorporated. On the other hand, these traits are not included in the analyses of MOSS or JPlag. Their main objective was to identify plagiarism using these traits as signals. To calculate the relative importance of every attribute in its assessment, this system utilized neural network techniques. But their foremost emphasis is on recognizing copied pairs inside a single issue set.

Research cited in [15] indicated that, in addition to detecting plagiarism in student assignments, the aforementioned methodologies can also be used in several different fields:

- Criminal prosecution - Locating the individual responsible for the virus that those in question used.
- Corporate Litigation - If a staff member violates a contract's no-compete clause, the code's author has to be named.
- Using plagiarism detection to locate the author in cases of academic malfeasance.

3. Proposed Methodologies

This research has been done to classify the source code files submitted by the students as assignments of C++ programming as plagiarized or not. The system is developed using a k-means clustering algorithm. The steps to implement k-means clustering for source code plagiarism detection are given below; however, it is depicted graphically in Fig. 2.

The following parts make up the recommended arrangement:

- Examine the documentation for the C++ source code.
- Source code preparation and collection must get underway.
- By making tokens for each file, combining them, and then determining an association between the two (TF-IDF file computation), it is possible to generate a matrix of term vs. file.
- Examine the query's C++ source code.
- Restart at step three.
- Go back to step 4
- Sort the files according to how comparable they are to the request by using k-means clustering on the source file for the request.
- The file is questionable if the similarity exceeds the threshold; otherwise, the source file is trustworthy.
- If any occurrences of plagiarism are found, a report must be filed. If not, go back to step 1 and try again.

Yet, in Fig. 2 we can see the proposed model using k-means clustering. This model classifies the source code file as plagiarized or non-plagiarized.

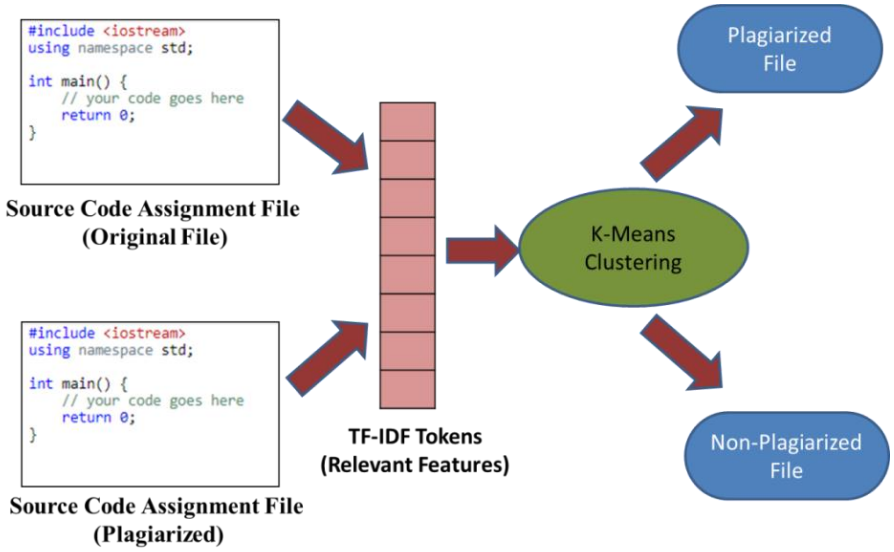

Fig. 2. A Model for detecting plagiarism in source code is proposed; it is trained with a k-means clustering algorithm [17] after features are retrieved from the required files using TFIDF tokens

3.1. Metrics of the code

The proposed method examines the input files and ascertains their characteristics after receiving the source codes. TFIDF code metrics are one of the proposed model's attributes. Code metrics are just a count of the tokens that are read from the files containing the code that, in this example, make up the student assignments that are turned in. These traits are employed to assess the effectiveness, style, system costs, reliability, adaptability, and architecture of a programmer's programs [18]. For obtaining these code metrics, several techniques and strategies, including those that follow, have been proposed:

- N-grams are collections of n textual elements (words, letters, etc.) from a database. The technique retrieves from articles [19] the average number of appearances of substrings of length n. It is a notion from processing natural languages. The United Kingdom, for example, weighs 2 grams.
- Term Frequency-Inverse Document Frequency (TF-IDF) is its abbreviation [20]. According to their prominence within the body of work, words in a document are assigned significance scores. Its primary uses are for data mining and information retrieving.
- A parser generator called ANTLR makes it easier to comprehend and interpret programming languages. To help one understand the framework of the code, syntax as well as vocabulary are also removed [21].
- Format changes only involve the addition or elimination of notes and white space.
- Altering the names of variables is an additional typical duplicating technique because it does not jeopardize the program's integrity.
- Modifying Statement - Modifying Statements that don't have consecutive dependency issues, like announcements, is simple.
- Control Replacement - In programming languages, there are numerous replacements for basic codes, such as if/else conditions, etc. [22].
- Code Insertion - Codes that don't alter the program's basic functionality can be introduced to conceal plagiarism.

These modifications are not presently supported by the proposed model. K-means clustering, a supervised classifier used in machine learning, was employed to hone the recovered attributes.

3.2. Machine learning phenomena

Machine learning refers to the research and development of algorithms that can analyze data for patterns and make predictions. These algorithms, rather than strictly conforming to explicit programmer instructions, instead develop a model using sample inputs to provide data-driven predictions or judgments. It is put to use for various computations when it would be too costly to create and implement explicit algorithms [23]. Web searches, social media content moderation, and e-commerce site recommendations are just a few of the numerous modern applications of machine learning. We presumably use machine learning millions of times every day without even realizing it at this point [24]. As a result of machine learning, we may now benefit from accurate speech recognition systems, lightning-fast web searches, autonomous vehicles, and a deeper comprehension of the human genome. The type of feedback delivered to the learning system typically categorizes machine learning jobs into one of three classes. The application of machine learning algorithms allows for the detection of duplicate source code [25]. Ahmed Aldelemy and others [34] have used a k-folding (k=5) cross-validation classifier which resulted in an accuracy of over 92%.

Here, you'll find the specifics:

3.2.1. Supervised machine learning

For a computer to learn, a teacher must first provide it with example inputs and outputs. A universal rule or function that converts inputs into outputs is what is envisaged. This includes classification as well as regression assessment [26].

3.2.2. Un-supervised machine learning

Since it is not provided with labels, the learning system must discover how the information is organized on its own. Clustering algorithms are also a part of this group of algorithms [3].

3.2.3. Reinforcement machine learning

Software in a changing setting, like a video game, must complete its objective without knowledge of its outcome. The programme receives reinforcement and correction as it works through its issue space [27].

3.3. K-Means clustering algorithm

In data science and machine learning, K-Means Clustering is a methodology for autonomous learning that is employed to tackle clustering problems [28]. Unsupervised Learning Algorithm K-Means The unstructured dataset is categorized using clustering. The total number of pre-defined clusters that have to be constructed throughout the method is determined by the parameter K; for example, if K=2, two clusters are created; if K=3, three clusters are created; and so on [29]. It provides a simple way for swiftly identifying the groups of individuals in a dataset that is unlabeled without any previous experience, allowing us to categorize data. Each cluster has a different centroid since the methodology is centroid-based.

The fundamental objective of this approach is to reduce the average separation between individual data points and the clusters to which they belong. This strategy begins with an unprocessed dataset and proceeds to classify the data into k categories until no further classifications are possible. A fixed value of k [30] should be used in this procedure. The k-means clustering algorithm generally achieves two objectives:

The k-means clustering algorithm generally achieves two objectives:

- The best value for K center points or centroids is the one that is closest to the center of each data point.
- Clusters of data points are formed when they share a common k-center.

As a result, there are unique data points with specific features in each cluster. The K-means Clustering Algorithm can be seen in Fig. 3.

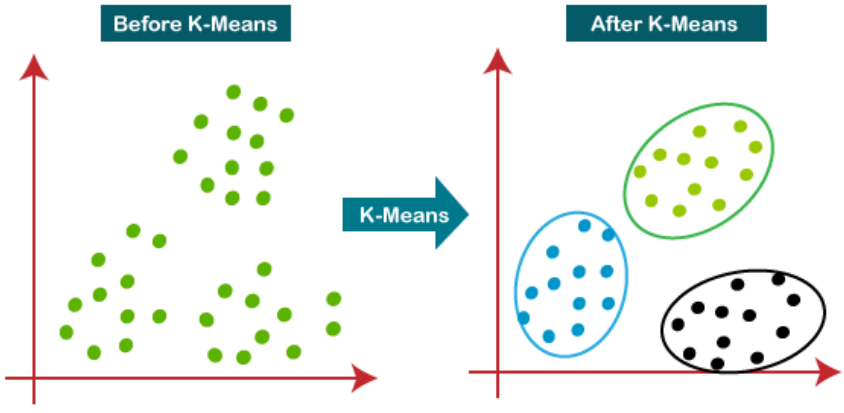

Fig. 3. Algorithm of K-means clustering

3.4. Machine learning and hierarchical clustering

Hierarchical clustering, also known as hierarchical cluster analysis (HCA), is an additional unsupervised machine learning technique used to classify data sets that have no labels. To visualize the relationship between groups, we create a dendrogram [31] that looks like a tree. Although

K-means clustering and hierarchical clustering work in different ways, they may yield identical results. It is not necessary to predict the number of clusters like we did with the K-Means algorithm. There are two approaches to using the hierarchical clustering method:

Agglomerative: Agglomerative is a bottom-up approach that initially treats each data point as its cluster, before combining them into larger ones.

Divisive: The division algorithm is the reverse of the agglomerative algorithm since it employs a top-down strategy.

Although it has been acknowledged that K-Means Clustering has some boundaries. Hierarchical clustering is an important method since it allows for a fixed number of clusters and a regular effort to create clusters of uniform size. It's interesting to note that employing the hierarchical clustering procedure does not mandate a specific number of clusters [32].

Clustering the datasets is another application of the bottom-up method. This means that the algorithm considers each dataset to be a separate cluster before combining the two most comparable ones. This procedure is repeated until a single cluster including all datasets has been created. The dendrogram is a graphical representation of this cluster structure.

The steps which make up the AHC algorithm's operation are as follows [33]:

- Generate a cluster for every data point in Step 1. Begin to suppose that there are N data points; therefore there will also be N clusters, see Fig. 4.

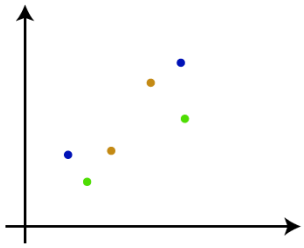

Fig. 4. Illustration of step 1

- Create a new cluster by joining together two neighboring ones. Therefore, we can expect N-1 clusters to form, see Fig. 5.

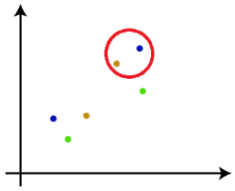

Fig. 5. Illustration of step 2

- Generate a single cluster by joining the two nearby clusters. There will be N-2 clusters, see Fig. 6.

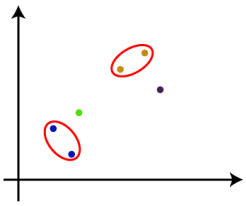

Fig. 6. Illustration of step 3

- Step 3 should be repeated until only one cluster remains. Accordingly, the following clusters will be obtained as represented in Fig. 7.

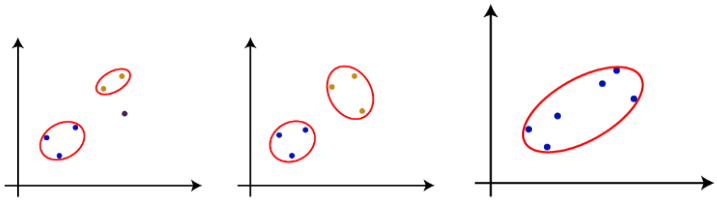

Fig. 7. Illustration of step 4

- If you find that your clusters have merged into one massive cluster, you can use a dendrogram to re-divide them into more manageable pieces.

1. Measuring the distance between two clusters

The hierarchical grouping is influenced by how close together the two groups are. The clustering criteria are defined by the procedure(s) used to measure the distance(s) between two clusters. These kinds of behaviors are denoted as connection approaches. Popular connecting methods are listed below [27] and include [28]:

Single Linkage in Hierarchical Clustering: The cluster's nearest points are divided by the least amount of space. Observe the illustration in Fig. 8.

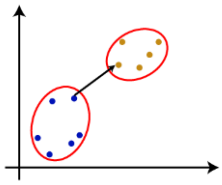

Fig. 8. Illustration of Single Linkage

Complete Linkage in Hierarchical Clustering: The distance between the centers of the two different clusters is at its maximum. It serves as one of the well-known linking approaches as it yields more compact clusters than single-linking, see Fig. 9.

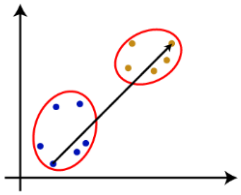

Fig. 9. Complete Linkage Illustration

Average Linkage in Hierarchical Clustering: To calculate the mean separation between clusters, the connection method totals the distance between every pair of datasets and divides the result by the total number of datasets. It's also one of the most common methods of linking people.

Centroid Linkage in Hierarchical Clustering: As shown in the graphic in Fig. 10, the linking method is employed to compute the distance between the cluster centroid:

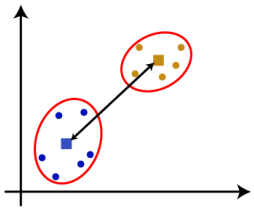

Fig. 10. Illustration of Centroid Linkage

According to the scope of the issue or the requirements of the business, it may employ any of the techniques listed above.

2. Hierarchical clustering and Working of Dendrogram [27]

HC's memory is mostly employed to keep track of intermediate steps in the form of tree-like diagrams called dendrograms. Each data point in the given dataset (on the X-axis) and all of the data points (on the Y-axis) are represented by their respective Euclidean distances on the dendrogram plot. The dendrogram's operation is shown in Fig. 11.

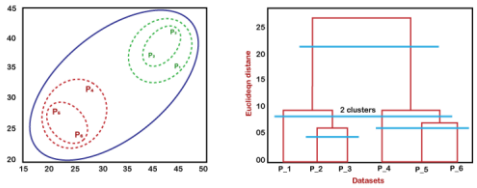

Fig. 11. Illustration of Centroid Linkage

The following picture shows how clusters get better throughout agglomerative clustering on the left, while the corresponding dendrogram is shown on the right [25].

- As was previously indicated, a cluster is created initially by the joining of data points P2 and P3. This results in a dendrogram with a rectangle connecting points (P2) and (P3). The height is determined by the Euclidean distance between the points in the data.

- In the next step, a dendrogram is generated alongside an established cluster consisting of Phases 5 and 6. The Euclidean distance between points 5 and 6 is slightly larger than that between points 2 and 3 and hence has grown.
- Two new dendrograms are created, with points P1, P2, and P3 in one and P4, P5, and P6 in the other.
- Finally, a dendrogram is made from all of the data.

Depending on our demands, the dendrogram tree structure can be sliced at any level. Fig. 12 depicts a recommended hierarchical clustering-based source identification approach.

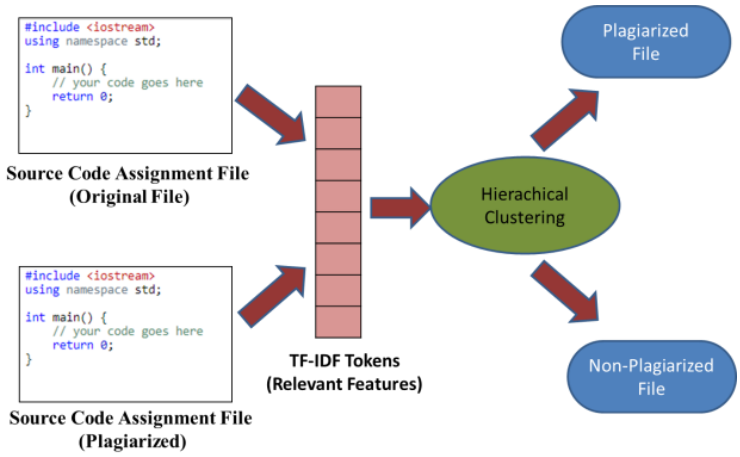

Fig. 12. Projected model using Hierarchical Clustering [25]

4. Results and Performance Evaluation

Several criteria are used to rate the effectiveness of the proposed model. The proposed system was evaluated using the following metrics [16]: cross-value score; accurate response rate; recall; F1 score; precision; and additional confusion matrices for the entire system and each iteration.

4.1. Classification accurateness

The most common statistic to evaluate classification duties is this one. The ratio of precise projections to all other forecasts is how precision is measured [8].

Accurateness of the system = # Correct Predictions / #Total Number of Predictions

This is the principle for determining how precise a binary categorization is:

Accurateness = (TN + TP) / (FP + TP + TN + FN)

The acronyms True positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) are employed in this situation. However, when classes are not evenly distributed, optimizing may not be an accurate metric because it may be relatively high and favor the common while ignoring the marginal.

4.2. Matrix of confusion

A confusion matrix [15] is an N by N matrix that gauges how successfully a model categorizes data, where N is the number of classes to be targeted. Real objective values and estimates provided by the machine learning model are compared in the matrix [21]. Fig. 13 gives an illustration of the confusion matrix.

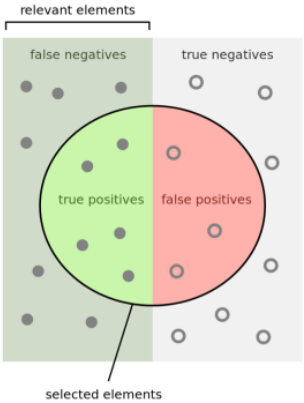

Fig. 13. Cluster illustration of a matrix of confusion

4.3. F-Measure

F-Measure takes into account both precision and recall when calculating a test's accuracy score. It's a four-way split between FP (False Positives), TN (True Negatives), FN (False Negatives), and TP (True Positives). Precision or Accuracy can be defined as the proportion of times that forecasts that turn out to be correct turn out to be correct [8].

p = (TP / (TP +FP))

Recall is determined by dividing the number of correct predictions by the total number of right positives [7].

r = (TP / (TP +FN))

The F-Score [13] is created using the recall and accuracy weighted average.

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

The significance level known as the F1-score (or F-value) is calculated as

F1Score = Harmonic mean (recall, precision)

The effectiveness of the proposed system is measured by analyzing student submissions to a Python-based C++ programming competition for beginners [30]. There are 44 separate entries, and each one has its own set of ten problems. The initial and amended copies of each student's work may be included in a maximum of twenty submissions. This is as a result of the approximately 880 entries overall. The source code file given in Fig. 14 is a sample code of the C++ program available in the dataset used. This program is a non-plagiarised C++ source 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;
}
```

Fig. 14. Source code for Example Program 1

The TF-IDF file generated for the source code program is considered the feature set for the classification. A sample TF-IDF is shown in Fig. 15. Table 1 shows the details of the dataset used for the experimental setup.

Table 1. Database Details

Dataset	Problem Sets	Submission	Language	Average LOC
Programming Contest	10.0	880.0	C++	1207

Using the k-means clustering and hierarchical clustering the proposed research has calculated recall, accuracy, F1-score, and precision metrics. These metrics were compared with the recall, accuracy, F1-score, and precision values of the existing system mentioned in the literature. These metrics are shown in Table 2. For the 90% and 80% training datasets, the proposed approach with hierarchical clustering was demonstrated to be stronger with 99.5% accuracy, whereas the system with k-means clustering had a 99.2% accuracy rate. We examined the recommended model, where accuracy, recollection, and F1-score are all more indicators that can be used to assess a model's performance since the correctness of the prior approaches hasn't been presented. Table 2 clearly shows that the proposed approach using a hierarchical algorithm outperforms the other systems (accuracy 0.995 i.e. 99.5%).

Table 2. Comparison of Existing and Proposed Systems

	K-means Clustering	Moss (90%)	Moss (80%)	Random Forest Algorithm	Proposed Approach
Recall	0.054	0.631	0.819	0.906	0.036
Accuracy	0.992	-	-	0.935	0.995
F1 Score	0.016	0.773	0.866	0.931	0.018
Precision	0.009	1.000	0.920	0.960	0.012

Fig. 16 shows different performance results and associated graphs.

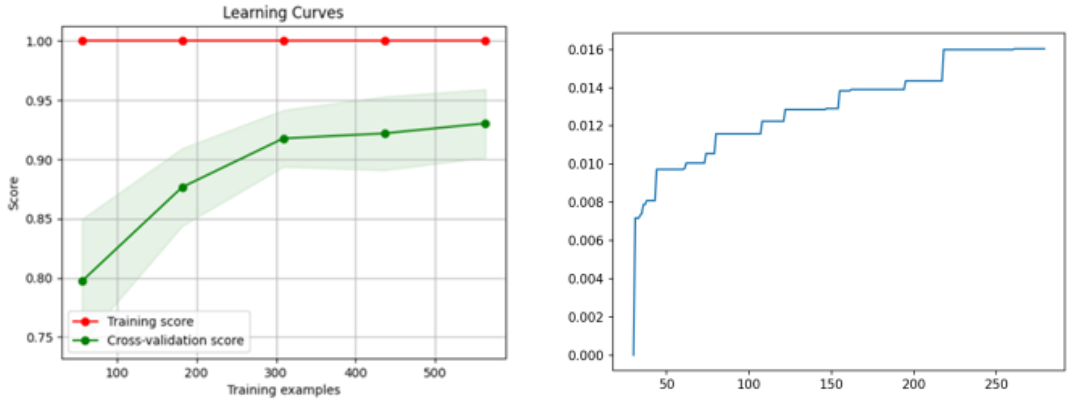

Fig. 15. a) Training samples vs Score curve, b) Accuracy curve

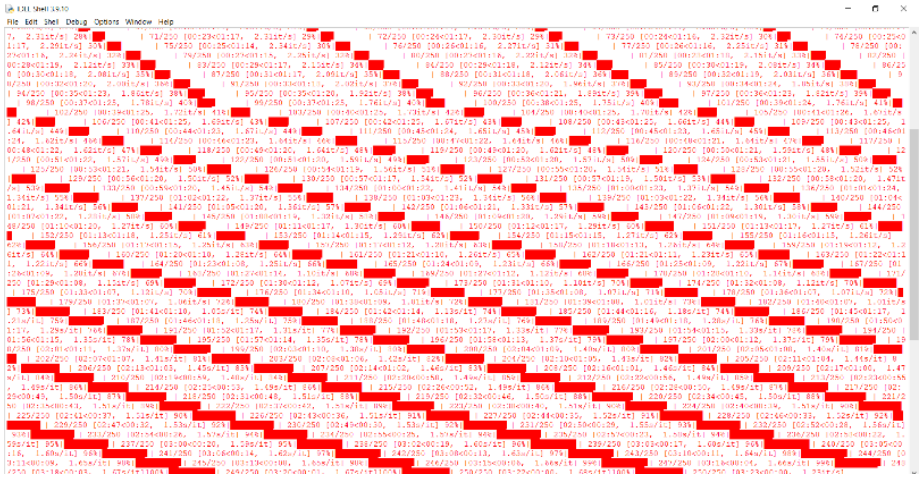

Fig. 16. Python output window

5. Conclusion

The proposed system was found to be superior with 99.2 percent accuracy when compared to the existing MOSS system for 90 percent and 80 percent training sets. 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 it to be superior at these metrics. The k-means clustering method outperforms the Random Forest Algorithm. The K-means clustering algorithm has some drawbacks, including a preset number of clusters and a tendency to always try to generate clusters of the same size. Because we don't need to be aware of the specified number of clusters while using the hierarchical clustering technique, we can choose this algorithm to address these two problems. In this study, we suggest a novel clustering strategy i.e. hierarchical clustering that achieves a higher rate of success 99.5% on training. sets the accuracy, recall, and F-measure are a few of the assessment metrics utilized to compare the results of the proposed system with the existing approaches like Random Forest Algorithm and MOSS. The work for GUI development using Python for the proposed system is ongoing. The researchers will soon publish it online for further research by the research community in the field.

Acknowledgment

We acknowledge the support from staff and colleagues in the Department of Computer Science, Middle Technical University, Baghdad, Iraq, and the team of the journal for their continuous support and help during the publication of this paper

References

[1] A. Ahtiainen, S. Surakka, and M. Rahikainen. "Plaggie: GNU-licensed source code plagiarism detection engine for Java exercises." Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling 2006, Uppsala, Sweden, 2006. https://doi.org/10.1145/1315803.1315831.

[2] C. Daly and J. Horgan. (2005, February) "Patterns of plagiarism." SIGCSE Bull., vol. 37, no. 1, pp. 383–387, 2005, 10.1145/1047124.1047473.

[3] I. Rahal and C. Wielga. (2014, September). "Source Code Plagiarism Detection Using Biological String Similarity Algorithms," Journal of Information & Knowledge Management, vol. 13, no. 03, p. 1450028, 10.1142/s0219649214500282.

[4] D. M. Breuker, J. Derriks, and J. Brunekreef, "Measuring static quality of student code," presented at the Proceedings of the 16th annual joint conference on Innovation and Technology in computer science education, Darmstadt, Germany, 2011, https://doi.org/10.1145/1999747.1999754.

[5] R. Brixtel, M. Fontaine, B. Lesner, C. Bazin, and R. Robbes, "Language-Independent Clone Detection Applied to Plagiarism Detection," presented at the Proceedings of the 2010 10th IEEE Working Conference on Source Code Analysis and Manipulation, 2010, https://doi.org/10.1109/SCAM.2010.19.

[6] H. Cheers, Y. Lin, and S. P. Smith, "Detecting Pervasive Source Code Plagiarism through Dynamic Program Behaviours," presented at the Proceedings of the Twenty-Second Australasian Computing Education Conference, Melbourne, VIC, Australia, 2020, https://doi.org/10.1145/3373165.3373168.

[7] X. Chen, B. Francia, M. Li, B. McKinnon, and A. Seker. "Shared information and program plagiarism detection," IEEE Trans. Inf. Theor., vol. 50, no. 7, pp. 1545–1551, (2004, June), 10.1109/tit.2004.830793.

[8] D. Chuda, P. Navrat, B. Kovacova, and P. Humay. "The Issue of (Software) Plagiarism: A Student View," IEEE Trans. on Educ., vol. 55, no. 1, pp. 22–28, (2012, February), 10.1109/te.2011.2112768.

[9] J. A. W. Faidhi and S. K. Robinson. "An empirical approach for detecting program similarity and plagiarism within a university programming environment," Comput. Educ., vol. 11, no. 1, pp. 11–19, (1987, May), 10.1016/0360-1315(87)90042-x.

[10] E. Flores, A. Barrón-Cedeño, L. Moreno, and P. Rosso. (2014, August). "Uncovering source code reuse in large-scale academic environments," Comput. Appl. Eng. Educ., vol. 23, no. 3, pp. 383–390, 10.1002/cae.21608.

[11] E. Flores, A. Barrón-Cedeño, P. Rosso, and L. Moreno. "Towards the detection of cross-language source code reuse." Natural Language Processing and Information Systems: 16th International Conference on Applications of Natural Language to Information Systems, NLDB 2011, Alicante, Spain, June 28-30, 2011. Proceedings 16. Springer Berlin Heidelberg, 2011, https://doi.org/10.1007/978-3-642-22327-3_31.

[12] E. Flores, A. Barrón-Cedeño, P. Rosso, and L. Moreno. "DeSoCoRe: Detecting source code re-use across programming languages." Proceedings of the Demonstration Session at the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2012.

[13] E. Flores, P. Rosso, L. Moreno, and E. Villatoro-Tello. "On the Detection of Source Code Re-use." presented at the Proceedings of the 6th Annual Meeting of the Forum for Information Retrieval Evaluation. Bangalore, India. p. 21-30, 2014, https://doi.org/10.1145/2824864.2824878.

[14] M. Freire. "Visualizing program similarity in the Ac plagiarism detection system." presented at the Proceedings of the working conference on Advanced visual interfaces. Napoli, Italy. p. 404-407, 2008, https://doi.org/10.1145/1385569.1385644.

[15] D. Gitchell and N. Tran. "Sim: a utility for detecting similarity in computer programs." SIGCSE Bull. vol. 31, no. 1, pp. 266–270, 1999, Available: 10.1145/384266.299783.

[16] J. Hage, P. Rademaker, and N. v. Vugt. "Plagiarism detection for Java: a tool comparison." presented at the Computer Science Education Research Conference, Heerlen, Netherlands, p. 33-46, 2011.

[17] B. Heeren, D. Leijen, and A. v. IJzendoorn. "Helium, for learning Haskell." presented at the Proceedings of the 2003 ACM SIGPLAN workshop on Haskell. Uppsala, Sweden. p. 62-71, 2003, https://doi.org/10.1145/871895.871902.

[18] D. Heres and J. Hage. "A Quantitative Comparison of Program Plagiarism Detection Tools." presented at the Proceedings of the 6th Computer Science Education Research Conference. Helsinki, Finland. p. 73-82, 2017, https://doi.org/10.1145/3162087.3162101.

[19] M. Joy and M. Luck. "Plagiarism in programming assignments." IEEE Trans. on Educ., vol. 42, no. 2, pp. 129–133, (1999, May), DOI: 10.1109/13.762946.

[20] R. M. Karp and M. O. Rabin. "Efficient randomized pattern-matching algorithms." IBM J. Res. Dev., vol. 31, no. 2, pp. 249–260, (1987, March), DOI: 10.1147/rd.312.0249.

[21] M. Kaya and S. A. Özel. "Integrating an online compiler and a plagiarism detection tool into the Moodle distance education system for easy assessment of programming assignments." Comput. Appl. Eng. Educ., vol. 23, no. 3, pp. 363–373, (2014, July), DOI: 10.1002/cae.21606.

[22] H. Li, C. Reinke, and S. Thompson. "Tool support for refactoring functional programs." presented at the Proceedings of the 2003 ACM SIGPLAN workshop on Haskell. Uppsala, Sweden, p. 27-38, 2003. DOI: 10.1145/871895.871899.

[23] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic. "An Information Retrieval Approach to Concept Location in Source Code." presented at the Proceedings of the 11th Working Conference on Reverse Engineering, p. 214-223, 2004, https://doi.org/10.1109/WCRE.2004.10.

[24] P. Modiba, V. Pieterse, and B. Haskin. "Evaluating plagiarism detection software for introductory programming assignments." presented at the Proceedings of the Computer Science Education Research Conference 2016. Pretoria, South Africa, p. 37-46, 2016, DOI: 10.1145/2998551.2998558.

[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, no. 3, p. Article 27, (2019, May), DOI: 10.1145/3313290.

[26] V. Pieterse, "Automated Assessment of Programming Assignments," presented at the Proceedings of the 3rd Computer Science Education Research Conference on Computer Science Education Research, Arnhem, Netherlands, 13, 4-5, 2013.

[27] R. Rivest, RFC1321: The MD5 Message-Digest Algorithm. RFC Editor, 1992.

[28] F. Rosales, A. Garcia, S. Rodriguez, J. L. Pedraza, R. Mendez, and M. M. Nieto. "Detection of Plagiarism in Programming Assignments." IEEE Trans. on Educ., vol. 51, no. 2, pp. 174–183, (2008, May), DOI: 10.1109/te.2007.906778.

[29] N. Tahaei and D. C. Noelle, "Automated Plagiarism Detection for Computer Programming Exercises Based on Patterns of Resubmission," presented at the Proceedings of the 2018 ACM Conference on International Computing Education Research, Espoo, Finland, 2018. DOI: 10.1145/3230977.3231006.

[30] N. R. Wagner, "Plagiarism by student programmers," Ph.D. dissertation, The University of Texas at San Antonio Division Computer Science San Antonio, TX, vol. 78249, 2000.

[31] R. A. Wagner and M. J. Fischer. "The String-to-String Correction Problem." J. ACM, vol. 21, no. 1, pp. 168–173, (1974, January) DOI: 10.1145/321796.321811.

[32] G. Whale. "Software metrics and plagiarism detection." J. Syst. Softw. vol. 13, no. 2, pp. 131–138, (1990, October), DOI: 10.1016/0164-1212(90)90118-6.

[33] M. J. Wise. "YAP3: Improved detection of similarities in computer program and other texts." Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education. p. 130-134, 1996, DOI: 10.1145/236452.236525.

[34] Aldelemy, A., & Raed A. Abd-Alhameed. Binary Classification of Customer's Online Purchasing Behavior Using Machine Learning. Journal of Techniques, 5(2), 163–186, (2023, June), https://doi.org/10.51173/jt.v5i2.1226.