International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 09 Issue: 03 | Mar 2022 www.irjet.net p-ISSN: 2395-0072

A Literature Review on Plagiarism Detection in Computer Programming Assignments

Keerthana T V [1], Pushti Dixit [2], Rhuthu Hegde [3], Sonali S K [4], Prameetha Pai[5]

[1,2,3,4]Students, Department of Computer Science and Engineering, Dayananda Sagar College of Engineering, Bengaluru, Karnataka, India
[5]Assistant Professor, Department of Computer Science and Engineering, Dayananda Sagar College of Engineering, Bengaluru, Karnataka, India

---

Abstract - *Our research aims to detect plagiarism in computer programming assignments. Plagiarism has been a problem for a long time and the problem has evolved with time. With the rise of the internet the theft of intellectual property has risen significantly and recognizing these thefts has become difficult. With this survey we have identified techniques used in detecting plagiarism. With changing times, the tools needed to detect plagiarism have to be evolved. However, to develop a tool with an ability to achieve high accuracy and greater accessibility of data has always been a demand. A comparative study on plagiarism checking tools with the technology used is presented in this paper. This study would help us determine the algorithm and methodology to proceed with the development of code to detect plagiarism.*

Key Words: plagiarism detection; machine learning; artifcial intelligence; deep learning; neural network

1.INTRODUCTION

Programming assignments play an important role to hone and evaluate programming skills of a student. But the process of finding a solution to the assignments seems frustrating to the majority of students that they borrow solutions from classmates or from external sources. 'Plagiarism' is presenting someone else's work or ideas as your own, with or without their consent, by incorporating it into your work without full acknowledgement. The increasing plagiarism cases in academics hinders fair evaluation of students, hence raises the need for plagiarism detection. Manual plagiarism detection takes a lot of time and effort, in the case of a large number of assignments. Hence, reliable automatic plagiarism detection techniques are necessary.

During this literature review, we came across several approaches that previous researchers have developed. The approaches were similarity-based, logic-based, machine learning based etc. Several comparison algorithms such as the Greedy String Tiling algorithm, winnowing algorithm etc. have been used in similarity-based detection. While logic-based algorithms try to find out one dissimilarity in terms of output and execution paths. For a system, which involved multiple submissions of a single exercise, similarity of consecutive submissions made by a student were considered which eliminates the need for finding similarity between all the source codes submitted by all the students. Machine learning techniques such as XGBoost, SVM, random forest, decision trees have also been used in some approaches. The take away and limitations of each of them have been briefed in this review. This review gives us an idea of what areas to work on while building a plagiarism detection system.

2. MOTIVATION

The practice of plagiarism is not a strange thing anymore. Students, programmers and even lecturers plagiarize or copy the source code from different sources before submitting it to the evaluator. This isn't just limited to schools and colleges but also in the industries. Detecting plagiarism practices is a solution that should be done so that the fraudulent actions can be minimized. Detection of plagiarism clusters is very important to find out how many students or groups accomplishing program homework independently. It gives instructors more opportunity to enhance or modify education. Using the software can be a deterrent for students to plagiarism. However, using this software does not provide the final answer, which is why the authors have come up with an idea of using two approaches and comparing them to find out which performs better, the first technique is to use machine learning techniques like XGBoost algorithm, SVM classifier and the second technique is to make use of Artificial Neural Network and backpropagation on the dataset to identify if there is plagiarism in code.

3. RELATED WORK

This paper by K.J.Ottenstein[1] talks about one of the earliest approaches to solving the problem of detecting similarities in student's computer programming assignments. This feature-based approach was designed for and tested on programs written in FORTRAN. It considers Halstead's metrics [2] - number of unique operators (n1), number of unique operators (n2), Total number of occurrences of operators(N1), Total number of occurrences of operands(N2). If n1 and n2 is found to occur exactly N1 and N2 times respectively in two assignments then those assignments are flagged as plagiarized.

This paper by John L Donaldson et al. [3] designed a plagiarism detection system that analyses the input programs in two phases. In the first, i.e., the Data collection phase the system keeps track of eight features. This information is stored in a two-dimensional array. The second phase is the Data Analysis phase which is further subdivided into 2 phases:

Phase 1.1 determines similarity using the information stored in the counters that keep track of the eight features. The three techniques used here are Sum of Differences, count of similarity and weighted count of similarity.

Phase 1.2 The input program is transformed to a statement order sequence and sequences of the pair of assignments are compared to detect similarity.

[1][3] are feature based methods that use software metrics to convert the input program into a feature vector that can be mapped to a point in an n-dimensional cartesian space.

The distance between the points determines the similarity of the two programs. It is observed by Lutz Prechelt et al. [8] that feature based systems do not consider valuable structural information of the programs and also observed that adding further metrics for comparison does not improve the accuracy.

Alan Parker et al [4] has given us a glimpse to the algorithms that can be used to detect plagiarism. The paper focuses on an algorithm that is based on string comparisons. It removes the comments, blank spaces, compares string and maintains count of the percentage where the characters are the same. The authors described six levels of plagiarism and their examples are shown in the paper. These algorithms have been developed on the theories of Halstead's metrics which brings out strong relation with software metrics.

Since this was an old paper, the research in this brought out automating the textual plagiarism thereby reducing human efforts.

Plagiarism in assignments by students has posed a lot of difficulties for the evaluators and to avoid that the author Michael J. Wise et .al [5] has proposed a system known as YAP3 which is the third version of YAP which works in two phases primarily. It removes the comments and string constants, converts from uppercase to lowercase, maps the synonyms to a common form, reorders the function in their calling order and also removes the token which is not a reserved word from the program. Also, the paper focuses on Running-Karp-Rabin Greedy-String-Tiling (RKR-GST) which was made after the observation of YAP and other systems for detection. The method can be used to detect transposed subsequence too. Also, the paper talks about usage on YAP on English texts which was a success.

This paper by Alex Aiken et al. [6] describes the idea behind MOSS (Measure of Software Similarity) a tool that automates detecting plagiarised programming assignments. MOSS accepts the programs as input and returns HTML pages illustrating parts of the accepted programs that it detects to be similar. The paper describes the winnowing algorithm.

The input is converted into k-grams (a continuous substring of length k) where the value of k is chosen by the user. Each k gram is hashed. A subset of the hashes is chosen to be the document's fingerprint and this paper describes the winnowing to select the hashes. A window of size w is created and, in each window, minimum hash value is chosen. If there is more than one minimum then the rightmost hash is selected. This algorithm was found to be efficient by the authors.

Then, Karp Rabin algorithm for string matching [7] is used to compare all pairs of k- grams in the two documents.

This approach was found to be language specific [18]

This paper by Richard M Karp and Michael O Rabin [7] gives us insightful information on the development of the Karp Rabin Algorithm, transitioning from the older techniques and identifying the underlying problem which led to the development of the algorithm. This is a string-matching algorithm in which fingerprint functions are used in the algorithm to identify the patterns. This algorithm is also suitable for multi-dimensional rectangular arrays. This algorithm short expected computed time with negligible probability of error and has a wide range of applications suitable for checking textual plagiarism.

Prechelt et al. [8] have described JPlag's architecture, it's evaluation results, among others. JPlag is a web service which detects plagiarism, given a set of programs as input. Firstly, it takes a set of programs as input, and then compares the programs in pairs, calculating total similarity value and a set of similarity regions for each pair. They have modified Wise's Greedy String Tiling algorithm by applying the basic idea of Karp-Rabin pattern matching algorithm, to compare programs. The output is a set of HTML pages which allows us to understand the similarity regions in detail.

They have evaluated JPlag against both original and artificial programs. It was found that JPlag was able to perfectly identify more than 90% of the 77 plagiarisms and the rest were at least termed suspicious. Runtime is also just a few seconds for around 100 programs of several 100 lines each. JPlag is limited to languages like Java and supports languages such as C, C++ and Scheme; support for other languages is still an area of concern.

Sven Meyer zu Eissen and Benno Stein [9] have focused their research on the intrinsic plagiarism method. Their research on previously used methods have led to the usage and analysis of intrinsic plagiarism. They have divided the

document into sentences, paragraphs or sections, and analyzing various features like stylometric features and averaged word frequency class. Their experimental analysis was on the computer science articles in ACM digital library, represented in the XML form and plagiarism checked with XML documents. They have represented their analysis with the help of the graph and tabular form. This paper has clarified us with the intrinsic method and its usage, which can be applied only for the textual documents.

This paper proposed by Liang Zhang et. al [10] uses a metric Information distance to measure the similarity between two programs and also detects the plagiarism clusters which helps in finding out how many of them have written the code independently and helps the course setters and instructors to enhance and modify the way education is communicated to the students. The detection system in the paper works in 3 phases particularly, parsing programming codes and translation into token sequences, calculating pairwise distance or similarity and clustering analysis work on similarity matrices in phase 2.

The system is pretty robust according to the author and is effective in clustering. They plan to implement fuzzy clustering in the detection system and support more programming languages such as Java, Basic and Delphi.

In the following paper by Cynthia Kustanto et.al [11], they have developed 'Deimos' a web application interface that receives input and then triggers background process to display the result on the application. Deimos detects plagiarism for source code written in Pascal, Lisp and C programming languages. Deimos performs following functions a) Detects plagiarism b) Displays the result in readable form c) Deletes the result. The application parses the source code and transforms it into tokens and then compares each pair of tokens using Running Karp-Rabin Greedy String Tiling algorithm. The advantages of this is that it detects plagiarism efficiently, can be accessed from any computer system, and can be used on other programming languages too. We can also set the detection sensitivity and it can process more than 100 source code. The method proposed might work on long programs and not so accurately on small codes. Also processing multiple programming assignments at a time might take longer than expected. A lot of mechanisms can be added to this to make it better and more efficient.

Dejan Sraka et al [12] focuses on the plagiarism done at the education levels and identifies the reasons behind plagiarism. The authors have also conducted various surveys and brought out the results which help us understand the reasons and analyse the type of plagiarism that are generally conducted. They have drawn conclusions from the survey such that there must be formal rules and regulations for the procedures and students and teachers must be educated to understand the importance of authorship, intellectual rights and rules of proper references. Also, the teachers must frame questions such that it has multiple solutions. This paper however does not focus on the plagiarism tools, it rather is just identifying the reasons and sources.

Martin Potthast et al [13] have presented an evaluation framework for plagiarism detection. The performance of the plagiarism detection is measured with the help of the plagiarism detection algorithm and measure to quantify the precision and granularity. The authors have come up with the corpus where there are three layers for plagiarism authenticity that is real plagiarism, artificial plagiarism and simulated plagiarism. The integration of these PAN plagiarism corpus is done by the authors in the PAN-PC10 corpus. The corpus features various kinds of plagiarism cases which help in validation which is done by 10 different retrieval models. They have aimed for a realistic test bed so that better performance can be achieved.

Duric and Gasevic [14] addresses the problem of making structural modifications to source code which can make detecting plagiarism very difficult and have presented a source code similarity detection system (SCSDS) which uses a combination of two similarity measurement algorithms such as RKR-GST algorithm and Winnowing algorithm. The approach consists of five phases, which are: 1) Pre-processing: In this phase, all sorts of comments from the source code file are removed. 2) Tokenization: Converting the source code into tokens, and these tokens are chosen in a way difficult for the plagiarists to modify, but still maintains the essence of the program. 3) Exclusion: In this phase, template code is excluded which can avoid many false positives. 4) Similarity measurement: RKR-GST and Winnowing algorithm are used to measure similarity. This phase is repeated twice due to the implementation of two algorithms. 5) Final similarity calculation: This calculation is performed on the results obtained in the previous phase.

The performance of SCSDS similarity measurement had shown promising results in comparison with JPlag. The tokenization phase and the usage of several similarity measurement algorithms contributes to the promising results obtained, but it's slower due to the usage of several similarity measurement algorithms which needs to be improved.

This paper by Bandara et al. [15] describes source code plagiarism detection using an attribute counting technique and uses a meta-learning algorithm to improve the accuracy of the machine learning model. Naive Bayes, K nearest neighbour algorithms were used for research and Adaboost algorithm was used for meta learning. Nine metrics were chosen to identify each source code and trained on a dataset of 904 java source code files and tested on a validation set of 741 files. The accuracy achieved was 86.64%. The authors plan to use other machine learning and meta learning algorithms in the future to improve the accuracy.

This survey by Prasanth S et al [16] tells us about the various techniques and tools used for plagiarism detection and various types of plagiarism detection. This survey gives us a complete understanding of different plagiarism methods and brings out the comparison between these methods which helps us choose the technique as per our need. This survey paper has just focused on the basic techniques and with the evolution of the Internet the challenge to plagiarism from these sources is still a big concern.

This paper by Weijun Chen et al. [17] proposes a source code plagiarism detection system that aims to combine feature-based and structure-based plagiarism detection methods into a single system. A system consisting of four components was designed by the authors. The components are: - Pre-Processor: This component removes the noise elements such as header files, comments, whitespaces, any input- output statement and any string literals as these elements could be used to fool the plagiarism detector.

Feature Based Component: This component considers two different categories of features i.e., Physical features and Halstead's Software metrics and builds a feature vector of the program given as input. Number of words and the number of source code lines are the two physical features considered in this component. Six Halstead's software metrics are considered namely arithmetic operator metrics, relational operator metrics, logical operator metrics, execution flow metrics, operand metrics and number of different operands.

The feature vectors of the two programs are compared to calculate the similarity. A sensitivity coefficient is used to define the strictness of the comparison. This component considers both Physical similarity(S1) and Halstead similarity(S2) along with weights w1 and w2. The authors have considered the weights w1=0.2 and w2 = 0.8.

Structure Based Component: This component uses a set of rules pre-defined by the authors to substitute the identifiers in the source program to a set of standard tokens to ensure that changes to names of variables or functions doesn't fool the plagiarism detector. To increase the efficiency further the token is further mapped to a single character using a mapping table defined by the authors resulting in a token string. The token string (separated into different blocks by the curly braces) is compared to derive a similarity score using the Longest Common Subsequence (LCS) Algorithm.

Integration Component: Combines the result of both feature and structure-based components using the integration algorithm. The authors tested the system by changing the variable names, adding useless statements and header files and the similarity score reported was 1.0. The similarity score reported after changing the structure of the program was 0.9. The system was also tested using the programs submitted by the students of an Introductory programming class of Tsinghua University, China and it was observed that if two programming assignments received a similarity score of 0.7 or above then there were enough code blocks that looked similar and is most likely plagiarised. The results are presented in an excel sheet that is difficult to understand.

Zhang et al. [18] proposed a program logic-based approach to software plagiarism detection (LoPD), which is both effective and efficient. In place of detecting similarity between two programs, LoPD detects dissimilarity. As long as it can find one dissimilarity i.e., in the form of either different output states or semantically different execution paths, it is not a case of plagiarism; it is, otherwise. An input is given to two suspected programs. If, a) Output States are different, it's not a plagiarism case. If they are the same, then check for path deviation. b) Execution paths are the same, try for another input. If, after many iterations, we cannot find an input for which either the output or the execution path is different, the programs are likely to be a plagiarism case. If they are different, check path equivalence to ensure true semantic deviation and not merely caused by code obfuscation.

While path characterization is done using techniques such as symbolic execution, weakest precondition and constraint solving to detect path deviation and measure semantics equivalence; constraint solver can lead to false positives and hence is bound by limitations.

This paper by Giovanni Acampora and Gerogina Cosma [19] describes a fuzzy-based approach to detect similarities in source code. First the source code is preprocessed to remove unnecessary information, after which a vector space model is created which is a matrix A that holds the frequency of the terms present in the source code after preprocessing. This frequency is normalised by a global weighing function. As the next step the dataset size is further reduced using Singular Value decomposition and the result is a matrix V.Neuro Fuzzy learning algorithm is applied on this matrix V which is a two-step process. Step 1: Fuzzy C-means clustering which groups the source code having similar identifiers together and generates a Fuzzy Inference system (FIS) that describes the rule for a particular cluster. Step 2: ANFIS algorithm [19] is used to tune the FIS to optimise the model. The System was tested on Java files and outperformed Self-Organising maps (SOM) approach and Fuzzy-C Means approach (FCM) and RKR-GST used in J-Plag.

In the following work by A. Chitra et. al [21] a support vector-based paraphrase recognizer is used which extracts lexical, syntactic and semantic features on text passages. The sentence-level paraphrase recognition system has been modified by the author to handle the text passages. There are 2 different approaches that have been used:

In the first approach, the input and the suspicious passages are split into sentences and it determines the closest matching source sentence. An SVM classifier is used to label the pairs.

In the second approach, paraphrase recognition features are extracted directly from the source and suspicious passage and the features extracted are used to determine if the suspicious passage is plagiarised from the source passage. The evaluation measures considered here are Accuracy, Precision, Recall, and F-measure. The system gave a good performance in both passage level and sentence level approach. It falls short in recognition of inputs having greater lexical variation also in handling very similar input phrases or passages.

The following paper by Matija Novak et. al [22] gives a review of the source code detection in academia. According to the paper, no similarity detection engine is powerful enough to be able to detect all the plagiarised code. Using text similarity detection can be useful at times to detect the plagiarism in the source code though not entirely. The author says, there should be more study on different similarity detection engines on the same dataset. The tools currently which are used for detection and stand out are jPlag, MOSS, SIM, Sherlock, and Piggie. To improve the accuracy of the detection one can, do pre-processing of the data or clean the code from unnecessary parts. There are many other tools like GATE and Gplag which haven't been compared more by the authors and should be the future scope for research.

Plagiarism detection is a difficult task as different programming languages could have different syntax and they are not only found in academic works but also in industry software codes. In the following paper by Mayank Agrawal et. al [23], the author describes two types of plagiarism techniques, textual and source code plagiarism. There are different tools which are described in the paper for both. The author has also explained about many other papers which have used techniques like Natural Language Processing (NLP), Machine Learning Technique, Running-Karp-Rabin Greedy String-Tiling algorithm, Character N-Grams, Data Mining, Latent Semantic Analysis (LSA) and Greedy String-Tiling. A comparative analysis is performed on the techniques, methods and the objectives and outcomes have been listed. The literature tells that copying is a dynamic process and is a danger to educational organisations.

In the following paper proposed by author Ahmed Hamza Osman et. al [24], it uses Semantic Role Labelling (SRL) and Support Vector Machine (SVM) for predicting and detecting plagiarised text passages. The algorithm analyses the text based on semantic position for the text. Text pre-processing is done on the input original text and the suspected text which includes text segmentation, stemming process and stop word removal.SRL procedure is then followed to find and name terms in text documents and passages. Precision and recall are the execution measures for the algorithms. The dataset used by the authors is PAN-PC-10 dataset. The proposed method is better than the other methods and has better execution. The authors later plan on including an integration of SRL-SVM with a translator method to remove the limitation of the previous proposed method.

Mirza et al. [25] analysed a Black Box dataset, which contains genuine student programming assignments (BlackBox is a project that collects data from the users of BlueJ online educational software), to check if the dataset was rich enough to apply coding style metrics to detect plagiarism. They considered random samples of 250 Java files each downloaded from the dataset. The files were pre-processed to remove white space and file headers.

They designed a small Java program which at first fetches random samples from the dataset, as the dataset contains duplicate files, only one file with the same ID is fetched out of the many with same IDs, to prevent duplication. Then, the number of lines and size of each source code file is measured(counted). Followed by measurement of complexity by counting the number of loops based on common loop words such as for, while etc. Ultimately, they are grouped into five subgroups based on the above features.

It was found that two out of the five subgroups could be ignored by any detection techniques (since they represent incomplete and template files), while the rest were rich enough for coding style metrics to be applied to detect plagiarism.

Jitendra Yasaswi et al [26] uses deep learning and natural language processing (NLP) to detect plagiarism. The authors have used char-RNN for feature extraction with an attempt to attain high accuracy. They have focused on statistical language modelling by training the char-RNN model on Linux Kernel codes. The LSTM (Long short-term memory) units have been added in char-RNN so that the learn features from C programs can be captured and used for comparison. Overall, the char-RNN model is trained first, then it is fine-tuned with some C programs which leads to obtaining embeddings for programming assignments that are submitted and use these embeddings (learn units) to detect plagiarism. However, they have used sequence-prediction which in fact can be directly applied on the different dataset without the need to fine tune each problem-set.

This paper by Jitendra Yasaswi et.al [27] describes a method which takes student's submissions to programming assignments as input and extracts the static features from the intermediate representation of the program using the MILEPOST GCC feature extraction plugin [28]. This plugin is capable of extracting 65 features. This feature is mapped to an n-dimensional space where n=55. The similarity between two different submissions is calculated using the Euclidean Distance formula and based on the similarity these submissions are clustered together. The results are consistent with the results obtained from MOSS on the same dataset. Mostly, [27] performs better and five cases illustrating this are described in the paper. The paper does

not identify any dynamic features and also does not consider the case of partially plagiarised submissions.

Tahaei and Noelle [29] have presented an approach which doesn't require the potential plagiarism sources in order to compare similarity, instead assumes an online system which allows for submission of multiple solutions for a single exercise, providing formative feedback with each submission. It compares pairs of submissions by an individual student, for similarity. They have used the 'diff' algorithm, which gives the minimum number of additions or deletions needed to transform one file into another. This was taken as a submission difference between two consecutive submissions, a vector of n – 1 dimensionality is created from the submission differences of each pair of consecutive submissions, starting from the submission difference between first and second submissions. Several features such as number of submissions; average, maximum, minimum and last differences were considered. For each examined subset of features, logistic regression was performed, identifying logistic sigmoid parameters which maximised plagiarism classification accuracy over the training set. The parameters were then used to calculate a plagiarism probability score. The results were evaluated against data collected from actual students enrolled in an undergraduate computer programming class at a research university. Though it couldn't perfectly classify plagiarism cases, there was strong correlation between these scores and actual cases of plagiarism. However, this method would fail if a student makes a single submission and also if the students (who intend to plagiarise) are aware of the features based on which plagiarism will be detected, as they will try to bypass detection.

Norman Meuschke et al [30] have published the prototype Hyplag which is able to detect strong textual plagiarism. Their target reviewers were reviewers of such work such as journal editors or PhD advisors. The Hyplag has the multi-stage detection process where there is candidate retrieval, detailed comparison and human inspection. They have taken into consideration mathematical similarity, image similarity, citation similarity and text similarity. The Hyplag prototype consists of the backend server which is realised in Java using the Spring Boot framework and a web-based frontend coupled via REST web service interface. They demonstrated their work by showing results overview as well as a detailed comparison view. The authors were able to demonstrate the hybrid analysis of the retracted source having the content features and bring out interactive visualisations that would help the reviewers in assessing the legitimacy of documents.

Budiman and Karnalim [31] have proposed an approach to draw hints from seating position and source code creation process to detect plagiarism as well as the plagiarists with an accuracy of at least 80.87% and 76.88% respectively. The approach consists of two modules, they are: Student module: This module is installed as a plug-in in the student's programming workspace. It frequently captures the source code snapshots which are compressed using Huffman coding algorithm for space efficiency. The capturing period and seating position ID are set before module installation. After assessment completion, all snapshots are merged as an archive and sent to the examiner module. Examiner module: It filters code pairs based on seating position which reduces pairs to be investigated by 80.87% and suggests suspected code pairs and plagiarists based on snapshots. In the future, they plan to use versioning systems such as GitHub to record snapshots and implement advanced similarity algorithms.

Siddharth Tata et al [32] have worked on the extrinsic plagiarism detection where the assignments and projects are copied from external sources like the Internet or fellow students. The authors have focused on generating n-grams and using the Karp-Rabin algorithm which generates the hash values. The winnowing algorithm is used to select the specific hash values and uses Jaccard similarity on the fingerprints generated from the passages to detect the plagiarism. The authors are yet to work on the content available on the Internet. They have currently worked on text files and would extend working on the programs.

This paper by Huang Quibo et al [33]. describes a method to extract features from submitted programs and uses a combination of Random Forest algorithm and Gradient Boosting Decision Tree. The authors propose two algorithms. Algorithm 1: A Similarity Degree Threshold (SRT) is set along with a Top limit. If the similarity (sim) of the programs is less than SRT then the program is not plagiarised. If sim is greater than Top limit then the program is suspected to be plagiarised and the student is asked to confirm if they want to submit the assignment. If they confirm then the program is sent for further review to the course instructor. If they decline, they system assumes that the program was plagiarised. If the sim value is between SRT and Top limit then some more features need to be extracted and the program is evaluated using the Random Forest and Gradient Boost Decision tree models to calculate plagiarism suspect level. Algorithm 2: The authors constructed a Random Forest containing multiple decision trees using entropy as criteria for classification and a Gradient boosting decision tree where each tree is a regression tree. Experiments show that algorithm 2 achieved a greater accuracy compared to algorithm 1 and the accuracy rate can reach up to 95.9%.

The following paper by K.K. Chaturvedi et. al[34] analyses and scrutinises the dataset, techniques and tools used to mine the software engineering data. The authors have basically categorised different tools used in Mining Software Repositories. The categorization of the tools is done based on the ones which are newly developed, traditional, prototype implemented and scripts. In most of the papers, studies by the author data retrieval, data pre-processing and post processing are most widely used and are important. The

tools which are used most of the time depend on the availability and most of them use a combination of many tools for the task. The authors further plan to study functions of all the tools used in Mining software Repositories and their applications which can help researchers in getting tools for their applications and for more research.

Mining repository is one of the techniques which is employed in the project and the following paper by Thai-Bao Do et. al [35] highlights more on the same. The paper proposes a method to detect forks and duplicates in the repository and also checks any correlation between the forking patterns, software health, risks and success indicators. There are more and more data which are being pushed into version control systems like GitHub, GitLab, Bitbucket, PyPI for Python and Debian for open-source software and the paper talks about the method which can be used to extract the metadata from the repositories hosted on platforms like GitHub, GitLab and Bitbucket. The study is done on Software Heritage dataset which consists of more than three million software repositories from many version control systems. The data extracted from the dataset is stored which can be used for future investigations the approach used by the authors work well and shows possible correlation between the metrics. But there is no concrete conclusion on the relationship which is also a part of their future works.

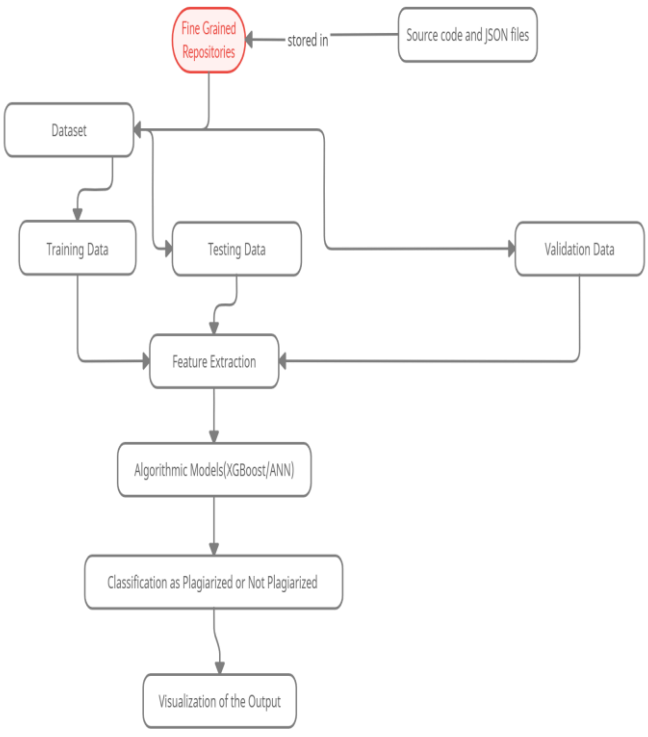
Nishesh Awale et al [36] says that the accuracy of detecting plagiarism is high by using the xgboost model with Support Vector Machine (SVM). The author says that they focused on the machine learning technique by working on feature-based extraction done by recurrent neural networks. The xgboost algorithm is trained with the features extracted and the results can be used with Support Vector Machine (SVM) to detect plagiarism. With the advantage of high accuracy, they are still working on incorporating compiler-based features.

Michael Duracik et al [37] has published the paper to detect the fraudulent activity done by the students in submitting computer programs. The author researched tools like JPlag, MOSS, Plaggie and designed an anti-plagiarism system based on that. The input source code is processed with abstract syntax tree (AST) and vectorization is done underlying the concepts of Deckard Algorithm. They have designed an algorithm for vectorization to add multiple vectors. Post this the vectors are sent for clustering, where apart from incremental algorithm they have also optimised the K-means algorithm. The plagiarism can be detected once the merging between the matching vectors takes place. For the best efficiency this method is compared with MOSS and JPlag systems and the results are noted. They faced few problems in bringing out the normalisation of inputs and bringing out an efficient algorithm for annotation of non-significant code.

P Ashwin et al [38] says that for plagiarism detection to be successful, one must use machine learning algorithms. The authors were specific with the software tools and algorithms they used for developing the plagiarism detection tool. They have used Anaconda Navigator cloud, Django and node.js for creating a local environment for hosting online assessments and contests. The dataset is obtained from the user and is used for testing and for training they use from the inbuilt libraries. They worked on normalisation using Natural Language Processing (NLP) and tokenization for the pre-processing. They focused on the cosine similarity and N-gram algorithms for detecting plagiarism. However, they were not able to scrap the data from the web content and are working on it.

Hussain K Chowdhury and Dhrubha K Bhattacharyya [39] have presented a taxonomy of various plagiarism forms, tools and various machine learning methods employed to detect the same. The authors have identified various types of plagiarism, bringing out clarity to extreme extents in which the former can be achieved. They also give us a detailed explanation on various types of plagiarism detection methods and a wide range of tools used. With clear diagrams, their survey brings out the clarity on its methods and tools to identify plagiarism. They have also highlighted the issues. They have identified all the major requirements of plagiarism and can help a developer to develop a code aptly by referring and identifying all the previously used methods and aim to improvise the accuracy.

4. FLOW DIAGRAM



5. TABLE

Table -1: Surveyed Papers

Author	Model description		
	Algorithm/features	***Advantages***	***Disadvantages***
Duric and Gasevic	RKR-GST algorithm and Winnowing algorithm	Promising results in comparison with JPlag	Slow, due to the usage of several similarity measurement algorithm
Bandara et al.	Naive Bayes, K nearest neighbour algorithms	Accuracy achieved was 86.64%.	Authors plan to improve the accuracy
Weijun Chen et al.	Six Halstead's software metrics, Longest Common Subsequence (LCS) Algorithm	Plagiarized assignments received high similarity scores (above 0.7) using this method	The results are presented in an excel sheet that is difficult to understand
Zhang et al.	LoPD, detecting dissimilarity	Resilience against most types of obfuscation techniques	Constraint solver may lead to false positives
Giovanni Acampora and Georgina Cosma	Fuzzy C-means clustering, ANFIS algorithm	Overcame problems of language dependency, misdetection due to code re-shuffling	Has not been tested on languages other than Java
A. Chitra et. al	SVM classifier	Good performance in both passage level and sentence level approach	Falls short in recognition of inputs having greater lexical variation and in handling very similar input phrases or passages.
Ahmed Hamza Osman et. al	Semantic Role Labelling (SRL) and Support Vector Machine (SVM)	Consequences of T-Tests found the advantages of proposed strategy examined in the paper were measurably huge	The time efficiency is O(n^2) cannot detect the cross-language semantic plagiarism
Jitendra Yasaswi et al	Char-RNN, LSTM (Long short-term memory) units	Can be directly applied on different datasets, no need to fine tune for each dataset.	
Jitendra Yasaswi et.al	Static features obtained using MILEPOST GCC feature extraction plugin, Euclidean Distance	Performs better compared to MOSS	Does not identify any dynamic features, does not consider the case of partially plagiarised submissions
Tahaei and Noelle	'diff' algorithm logistic regression, logistic sigmoid parameters	Online system, allows for submission of multiple solutions for a single exercise	Method fails if a student makes only a single submission
Norman Meuschke et al	Relative distance measure, Discrete Cosine Transform (DCT),Bibliograp	Online system, Considers mathematical similarity, image similarity, citation similarity and	

Author	Model description		
	Algorithm/features	***Advantages***	***Disadvantages***
	hic Coupling (BC),Longest Common Citation Sequence (LCCS),Greedy Citation Tiling (GCT),Citation Chunking (CC),full string matching, the Encoplot algorithm.	text similarity	
Budiman and Karnalim	Huffman coding algorithm	Accuracy of at least 80.87% and 76.88%, Considers seating position and source code creation process	Does not make use of versioning systems
Siddharth Tata et al	Winnowing algorithm, Karp-Rabin algorithm, Jaccard similarity	Works on text files	Yet to work on the content available on the Internet, Yet to extend the system to work on programs
Huang Quibo et al	Combination of Random Forest algorithm and Gradient Boosting Decision Tree,	Accuracy rate can reach up to 95.9%	
	Similarity Degree Threshold(SRT)		
Nishesh Awale et al.	XGBoost model with Support Vector Machine (SVM)	High accuracy	Yet to work on compiler-based features
Michael Duracik et al.	Deckard algorithm, Abstract Syntax Tree, K-means algorithm	Better results than MOSS and JPlag	Issues regarding normalization of inputs and with annotation of non-significant code
P Ashwin et al.	Cosine similarity and N-gram algorithms	Promising results	Yet to work on web scraping

6. CONCLUSIONS

Plagiarism is a ubiquitous problem faced by practitioners of different fields like academia, journalism, literature, art and so on for decades. The field has been researched intensively since the 1970's. With the advances in technology and the pervasiveness of the world wide web, everyone has all the information they need at their fingertips. Especially in academia, this poses a problem to fair evaluation of the students and also inhibits the student's learning process.

While many efforts were concentrated towards detecting textual plagiarism, significant strides have been made in the field of source code plagiarism detection. We can observe the leap from manual plagiarism checking to algorithm-based, automatic plagiarism checking made possible by advancements in technology. We can also observe the shift from using local client applications to web-based applications and now to cloud-based applications, making plagiarism detection systems easy to use and available everywhere. The detection methods started from simple feature-based approach, structure-based approach, then moved on to hybrid approaches that used similarity measurement and string-matching algorithms as students started to evade these systems. Recent approaches using

Machine Learning and Deep Learning algorithms and techniques have shown some promising results to improve the accuracy and automating the process of plagiarism detection.

REFERENCES

[1] Karl J Ottenstein, "An algorithmic approach to the detection and prevention of plagiarism", ACM SIGCSE Bulletin, Volume 8, Issue 4, Dec. 1976, pp 30–41.

[2] Joseph L.F. De Kerf, "APL and Halstead's theory of software metrics", APL '81: Proceedings of the international conference on APL, October 1981, Pages 89–93.

[3] John L Donaldson, Ann Marie Lancaster and Paula H Sposato, "A plagiarism detection system", SIGCSE '81: Proceedings of the twelfth SIGCSE technical symposium on Computer science education, February 1981, Pages 21–25.

[4] Alan Parker and James O. Hamblen, "Computer Algorithms for Plagiarism Detection", IEEE Transactions On Education, Vol. 32, No. 2. May 1989.

[5] Michael J Wise, "YAP3: improved detection of similarities in computer program and other texts", SIGCSE '96: Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education, March 1996, Pages 130–134.

[6] Saul Schleimer, Daniel S. Wilkerson and Alex Aiken, "Winnowing: Local Algorithms for Document Fingerprinting", SIGMOD 2003, June 9-12, 2003, San Diego, CA. Copyright 2003 ACM 1-58113-634-X/03/06.

[7] Richard M. Karp and Michael O. Rabin, "Efficient randomized pattern-matching algorithms", Published in: IBM Journal of Research and Development (Volume: 31, Issue: 2, March 1987), Page(s): 249 - 260.

[8] Lutz Prechelt and Guido Malpohl, "Finding Plagiarisms among a Set of Programs with JPlag", March 2003, Journal Of Universal Computer Science 8(11).

[9] Sven Meyer zu Eissen and Benno Stein, "Intrinsic Plagiarism Detection", M. Lalmas et al. (Eds.): ECIR 2006, LNCS 3936, pp. 565–569, 2006.

[10] Liang Zhang, Yue-ting Zhuang and Zhen-ming Yuan, "A Program Plagiarism Detection Model Based on Information Distance and Clustering", Published in: The 2007 International Conference on Intelligent Pervasive Computing (IPC 2007), Date Added to IEEE Xplore: 22 January 2008,Print ISBN:978-0-7695-3006-2.

[11] Cynthia Kustanto and Inggriani Liem, "Automatic Source Code Plagiarism Detection", 2009 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, Date Added to IEEE Xplore: 13 October 2009, Print ISBN:978-0-7695-3642-2.

[12] Dejan Sraka and Branko Kauþiþ, "Source Code Plagiarism", Proceedings of the ITI 2009 31st Int. Conf. on Information Technology Interfaces, June 22-25, 2009, Cavtat, Croatia.

[13] Martin Potthast, Benno Stein, Alberto Barrón-Cedeño and Paolo Rosso, "An Evaluation Framework for Plagiarism Detection", Coling 2010: Poster Volume, pages 997–1005, Beijing, August 2010.

[14] Zoran Đurić and Dragan Gašević, "A Source Code Similarity System for Plagiarism Detection", The Computer Journal, Volume 56, Issue 1, January 2013, Pages 70–86, https://doi.org/10.1093/comjnl/bxs018, Published: 13 March 2012.

[15] Upul Bandara and Gamini Wijayrathna, "Detection of Source Code Plagiarism Using Machine Learning Approach", International Journal of Computer Theory and Engineering, Vol. 4, No. 5, October 2012.

[16] Prasanth. S,Rajshree. R and Saravana Balaji B, "A Survey on Plagiarism Detection", International Journal of Computer Applications (0975 – 8887) , Volume 86 – No 19, January 2014.

[17] Weijun Chen, Chenling Duan, Li Zheng and Youjian Zhao, "A Hybrid Method for Detecting Source-code Plagiarism in Computer Programming Courses", The European Conference on Education 2013, Official Conference Proceedings 2013.

[18] Fangfang Zhang,Dinghao Wu,Peng Liu and Sencun Zhu, "Program Logic Based Software Plagiarism Detection", ISSRE '14: Proceedings of the 2014 IEEE 25th International Symposium on Software Reliability Engineering, November 2014, Pages 66–77.

[19] Giovanni Acampora and Georgina Cosma, "A Fuzzy-based Approach to Programming Language Independent Source-Code Plagiarism Detection", Published in: 2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 30 November 2015.

[20] J.-S.R. Jang, "Input selection for ANFIS learning", Proceedings of IEEE 5th International Fuzzy Systems, 06 August 2002, 0-7803-3645-3,New Orleans, LA, USA.

[21] A. Chitra and Anupriya Rajkumar, "Plagiarism Detection Using Machine Learning-Based Paraphrase Recognizer", From the journal Journal of Intelligent Systems, https://doi.org/10.1515/jisys-2014-0146.

[22] Matija Novak, "Review of source-code plagiarism detection in academia", 2016 39th International Convention

on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 28 July 2016,978-953-233-086-1,Opatija, Croatia,DOI:10.1109/MIPRO.2016.7522248.

[23] Mayank Agrawal and Dilip Kumar Sharma, "A state of art on source code plagiarism detection", 2016 2nd International Conference on Next Generation Computing Technologies (NGCT), 16 March 2017, 978-1-5090-3257-0, Dehradun, India, DOI:10.1109/NGCT.2016.7877421.

[24] Ahmed Hamza Osman and Omar M. Barukab, "SVM significant role selection method for improving semantic text plagiarism detection", International Journal of Advanced and Applied Sciences, 4(8) 2017, Pages: 112-122.

[25] Olfat M. Mirza, Mike Joy and Georgina Cosma, "Style Analysis for Source Code Plagiarism Detection — An Analysis of a Dataset of Student Coursework", 2017 IEEE 17th International Conference on Advanced Learning Technologies (ICALT), 08 August 2017, 978-1-5386-3870-5, Timişoara, Romania,DOI: 10.1109/ICALT.2017.117.

[26] Jitendra Yasaswi, Suresh Purini and C. V. Jawahar, "Plagiarism Detection in Programming Assignments Using Deep Features", 2017 4th IAPR Asian Conference on Pattern Recognition, 17 December 2018, 978-1-5386-3354-0, Nanjing, China, DOI:10.1109/ACPR.2017.146.

[27] Jitendra Yasaswi, Sri Kailash,Anil Chilupuri, Suresh Purini and C. V. Jawahar, "Unsupervised Learning Based Approach for Plagiarism Detection in Programming Assignments", ISEC '17: Proceedings of the 10th Innovations in Software Engineering Conference, February 2017, Pages 117–121,DOI:10.1145/3021460.3021473.

[28] Reference publications about cTuning.org long-term vision: GCC Summit' 09, ACM TACO'10 journal and IJPP'11 journal, https://ctuning.org/wiki/index.php/CTools:MilepostGCC.

[29] Narjes Tahaei and David C. Noelle, "Automated Plagiarism Detection for Computer Programming Exercises Based on Patterns of Resubmission", ICER '18: Proceedings of the 2018 ACM Conference on International Computing Education Research,August 2018, Pages 178–186, DOI:10.1145/3230977.3231006.

[30] Norman Meuschke, Vincent Stange, Moritz Schubotz and Bela Gipp, "HyPlag: A Hybrid Approach to Academic Plagiarism Detection", SIGIR '18: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, June 2018, Pages 1321–1324, DOI:10.1145/3209978.3210177.

[31] Ariel Elbert Budiman and Oscar Karnalim, "Automated Hints Generation for Investigating Source Code Plagiarism and Identifying The Culprits on In-Class Individual Programming Assessment", Computers 2019, 8(1), 11, DOI:10.3390/computers8010011,Published: 2 February 2019

[32] Siddharth Tata, Suguri Charan Kumar and Varampati Reddy Kumar, "Extrinsic Plagiarism Detection Using Fingerprinting",International Journal of Computer Science And Technology (IJCST) Vol. 10, Issue 4, Oct - Dec 2019.

[33] Huang Qiubo, Tang Jingdong and Fang Guozheng, "Research on Code Plagiarism Detection Model Based on Random Forest and Gradient Boosting Decision Tree", ICDMML 2019: Proceedings of the 2019 International Conference on Data Mining and Machine Learning, April 2019, Pages 97–102, DOI:10.1145/3335656.3335692.

[34] K.K. Chaturvedi, V.B. Sing and Prashast Singh, "Tools in Mining Software Repositories", 2013 13th International Conference on Computational Science and Its Applications,12 December 2013, 978-0-7695-5045-9, Ho Chi Minh City, Vietnam,DOI: 10.1109/ICCSA.2013.22.

[35] Thai-Bao Do, Huu-Nghia H. Nguyen, Bao-Linh L. Mai and Vu Nguyen, "Mining and Creating a Software Repositories Dataset", 2020 7th NAFOSTED Conference on Information and Computer Science (NICS), 02 February 2021, 978-0-7381-0553-6, Ho Chi Minh City, Vietnam, DOI: 10.1109/NICS51282.2020.9335894.

[36] Nishesh Awale, Mitesh Pandey, Anish Dulal and Bibek Timsina, "Plagiarism Detection in Programming Assignments using Machine Learning", Journal of Artificial Intelligence and Capsule Networks (2020),21.07.2020,Vol.02/ No. 03, Pages: 177-184, DOI:10.36548/jaicn.2020.3.005.

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

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

[39] Hussain A Chowdhury and Dhruba K Bhattacharyya, "Plagiarism: Taxonomy, Tools and Detection Techniques",Paper of the 19th National Convention on Knowledge, Library and Information Networking (NACLIN 2016) held at Tezpur University, Assam, India from October 26-28, 2016, ISBN: 978-93-82735-08-3.