International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-8 Issue-4, April 2019

Plagiarism Detection in Source Code Using Machine Learning

S. Priya, Anukul Dixit, Krishanu Das, Ronak Harish Patil

Abstract *- The Source Code Plagiarism has become a major problem in today's Educational World. The boost in the technology has led to the development of major IT sector and software industry. Thus, intellectual property of a being is not less important than any other valuable property. If the plagiarism reaches above the phase 6 it becomes almost impossible to detect it using the tools which are designed for the structural analysis. Therefore, we designed the new model for source code plagiarism detection, which uses the concepts of Machine Learning in order to fight with the higher phases of plagiarism. Conventional methods like structural methods, attribute counting method and graph-based analysis don't produce the results with accuracy. Machine learning algorithms produce the most accurate result with continuous learning from the training modules. The three algorithms used are Naïve Bayes Algorithm, k-Nearest Neighbor and ADA Boost Meta Learning Algorithm. Since, no single algorithm can produce the result with accuracy, thus combining the algorithms help to produce results more accurately.*

Index Terms**:** *machine learning, k-nearest neighbor, naïve_bayes_classifier, source code., plagiarism detection.*

I. INTRODUCTION

There is a huge demand in the academia as well as industries nowadays of an accurate plagiarism checker for the source code. Plagiarism check by humans is impractical and the conventional methods of plagiarism detection in source code are not much accurate [1]. The main aim of this research is to make it easier to determine which cases of measured similarity might be plagiarism. Cases of plagiarism, either coincidental or actual are common in programming. In this research paper, the primary focus is to ensure these issues are addressed. The objectives that make up the aim are too:

1. Find better ways of investigating plagiarism in programming assignments using a machine learning based approach.
2. Combine the use of structure similarity detection, graph-based techniques and machine learning and design a combined approach, that could be used to detect plagiarism in source codes.
3. Derive an efficient method for detecting plagiarism in programs.
4. Resolve the anomalies of cases in programs. Resolution is done by the manual evaluation of the plagiarism report.

II. EXISTING SYSTEM

Source code plagiarism [2] is defined as a program which has been reproduced with a small number of routine transformations. It is a significant issue in academia as well as in industry. It is literally the reusing of programs rather than the copying of essays. Source code can be reused in different forms from copying and pasting small chunks of a program source code to copying large chunks of a source code and masking these copies with creative techniques to conceal the original copied program.

Phase 1 - Changes in indentation and comments of a program
Phase 2 - Changes in Phase 1 and changes in program identifiers
Phase 3 - Changes in Phase 2 and changes in variable position
Phase 4 - Changes in Phase 3 and changes in procedure combinations
Phase 5 - Changes in Phase 4 and changes in the statements of programs
Phase 6 - Changes in Phase 5 and changes in the control logic

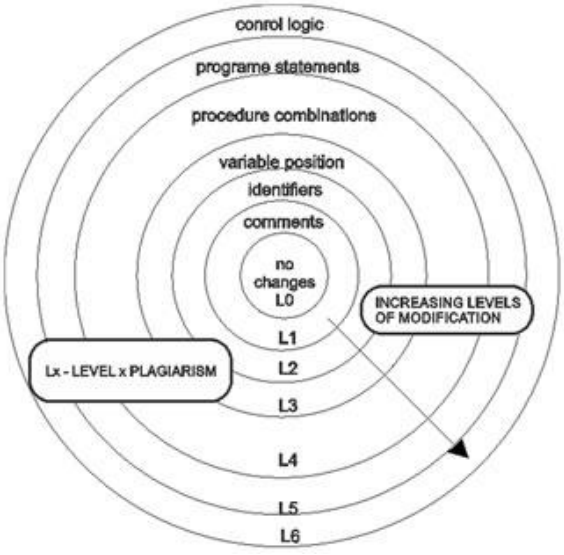

Fig 1: Conventional methods used in plagiarism detection

Manuscript published on 30 April 2019.
***** Correspondence Author (s)
S. Priya*, Computer Science & Engineering, SRM Institute of Science & Technology, Chennai, India.
Anukul Dixit, Computer Science & Engineering, SRM Institute of Science & Technology, Chennai, India.
Krishanu Das, Computer Science & Engineering, SRM Institute of Science & Technology, Chennai, India.
Ronak Harish Patil, Computer Science & Engineering, SRM Institute of Science & Technology, Chennai, India.

A. Attribute Counting System

The attribute counting system relied on Halstead software features in detecting plagiarism in student-generated programs by counting the number of operators and operands.

1. Number of unique operators in a program
2. Number of unique operands in a program
3. Total operators in a program
4. Total operands in a program
5. The code lines (excluding comment lines) in a program
6. The used and declared variables
7. Total control statements

B. Structure Counting Method

The structure, counting system compares the representations of programs' structures. The representation can be a program's string, data flows, work flows and parse trees. The structure, counting system does not compare exact matches as in the case of the attribute counting system, but checks for the similarity of a program's token string for suspicious plagiarism. In a structure, counting system, comments, white spaces, and variable names are discarded because they are easily customizable. It only changes the structure of programs and compares them for plagiarism.

C. XPlag

XPlag detects interlingual plagiarism, where source code is copied from one programming language to another by using a compiler suite to convert the programs. The reason for this, is because, programs have the same structure, hence, it is easy to plagiarize a source code written and converted from one programming language to another. More so, a code written in C, may be converted into Java. Hence, the plagiarism detection system explores detecting programs converted from one programming language to another to mask plagiarism

III. PROPOSED SYSTEM

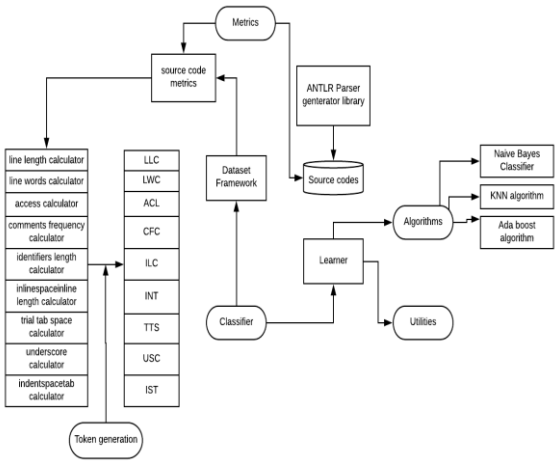

Fig 2: System architecture

Fig 2 gives us a clear idea on how the system is working. It shows a work flow architecture of the system. In the given diagram the source codes are generated by the ANTLR parser generator library. ANTLR is also known as Another Tool for Language Recognition. ANTLR takes a grammar that specifies a language as input and generates source code as output for a recognizer of that language.

Every subsystem in the figure other than the source code sub system is charted into a package of java. The source code sub system contains Java source codes, which are used for training as well as testing the system. Source code features or features are generated from the features sub system using java source code files. To extract source code features the source codes are parsed in terms of the syntactic laws of the programming language. Each source code feature is converted to a set of tokens. Along with the tokens, token frequencies are also noted. These tokens and token frequencies are taken as input and final result is displayed using the classification algorithms.

The classifier subsystem is given a high-phase interface and requests source code features via the dataset framework. The classification algorithms that includes Naïve_bayes_classifier, KNN algorithm and ada boost algorithm are implemented in algorithm subsystem. The KNN algorithm and naïve_bayes_classifier are implemented based upon the pseudo code. The ada boost algorithm is used to boost the weak learning problems. Extra functionalities of algorithms are implemented by the utility's subsystem. Learner subsystem acts as a high-phase abstraction layer. Functionalities of algorithms and utilities are requested by the classifier sub system through the interface provided by the learner subsystem.

A.Algorithms used for identification of source code

The Algorithms used for pattern recognition can be used for source code pattern recognition. In this section we will discuss about the algorithms used for the research:

1. Naïve_bayes_classifier

Naive Bayes classifier is a group of classification algorithms that is based on the Bayes' Theorem**.** It is not a one algorithm but a bunch of algorithms where all of them share a similar principle, i.e. each pair of features being classified is self-governing of each other.

$$p(C|F_1, F_2, \ldots F_n) = \frac{1}{Z} p(C) \prod_{i=1}^{n} p(F_i|C) \quad (1)$$

The Naïve_bayes_classifier [4] contains two types:

1. Multinomial Naïve_bayes_classifier

$$p(F_i|C) = \frac{N_{F_i} + 1}{N_c + 2} \quad (2)$$

2. Bernoulli Naïve_bayes_classifier

$$p(F_i|C) = \frac{T_{F_i} + 1}{\sum_{t' \epsilon v} T_{F_{i'}} + B} \quad (3)$$

2. k-Nearest Neighbor (kNN)

k nearest neighbor is a humble algorithm that keeps all available instances and classifies new instances based on a similarity measure. KNN has been used in statistical approximation and pattern recognition already at the start of 1970's as a non-parafeature technique [5].

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

$$d(X_i, X_j) = \sqrt{\left((x_{i1} - x_{j1})^2 + \cdots + (x_{ip} - x_{jp})^2\right)} \quad (4)$$

3. ada_Boost meta_learning algorithm

The boosting algorithm is utilized to boost the correctness of any weak learning algorithm increases its accuracy[12]. There are several dissimilar types of boosting algorithms are present in the research literature. For this study, we used the ada_Boost meta_learning algorithm.

IV. IMPLEMENTATION

Ding and Samadzadeh[3] told that every source code features does not contribute equally for identification of the source code Writer. According to them layout features perform a major function than other features. For data set we used java[6] source code, therefore we are using ANTLR[7] parser. Hence, these nine features shown in Table 1 where found, which performed great in identification of the program.

Table 1 Program features used to identify of the writer of the code

Features Name	Account
Line_Length_Calculator	The number of parts in one program line is calculated by this feature.
Line_Words Calculator	The number of words in one source code line is calculated by this feature.
Access_Calculator	Four phases of access controls are used by java: protected, default ,private and public. Programmers use these features to measure the respective frequencies of these access phases.
Comments_Frequency_Calculator	Three types of comments are used by java. To calculate the respective frequencies of those comment types the programmers use this feature.
Indentifiers_Length_Calculator	The length of each and every identifier of the programs is calculated by this feature.
InLine_SpaceInline_Tab_Calculator	the blank spaces that occurs along the inner parts non blank space lines are calculated by this feature.
Trail_Tab_Space_Calculator	the tabs and whitespaces showing at the last stage of each non blank space line is calculated by this feature.
Underscores_Calculator	The number of underscore characters used in identifiers is calculated by this feature.
Indent_Space_Tab_Calculator	The indentation of blank spaces used at the outset of each non-blank space lines are calculated by this feature.

For every code feature a particular code is indicated in Table 2. Afterwards that, we transformed every program file into a group of tokens along with their frequencies.

A. Producing a group of tokens

As we mentioned before, for each source code feature we assign a three-letter special code shown as follows:

Table 2 Program features with its token names

Code Feature	Coding System
Line_Length_Calculator	LLC
Line_Words_Calculator	LWC
Access_Calculator	ACL
Comments_Frequency_Calculator	CFC
Indentifiers_Length_Calculator	ILC
InLine_SpaceInline_Tab_Calculator	INT
Trail_Tab_Space_Calculator	TTS
Underscores_Calculator	USC
Indent_Space_Tab_Calculator	IST

For example, consider "Line_Words_Calculator" code feature produces the feature as shown in Table 3 for a particular program file.

Utilizing the coding format introduced in Table 2 , the tokens and its frequencies were generated for the 'Line_Words_Calculator" for the feature described in Table 3.

Table 3 Output of Line_Words_Calculator feature

Line_Distance	Number_of_ Occurrences
10	24
16	40
30	26
40	18
64	22
76	6
80	4

Table 4 Frequencies of tokens.

Token_names	Frequency
LWC_10	24
LWC_16	40
LWC_30	26
LWC_40	18
LWC_64	22
LWC_76	6
LWC_80	4

As shown in the above table, every program file are shown as a group of tokens and its frequentness. These tokens and their frequencies are used as inputs for our learning algorithms. This process is similar to the use of word and word frequencies in document classification problems. To understand how these tokens and its frequencies are used by the classification algorithms, consider the following simple training and testing data set shown in Table 4.5.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

Table 5 Data set

	Id	(token, frequencies)	Category
Training_ set	1	(LWC_10,4), (LWC_16,24), (LWC_14,4)	Writer 1
	2	(LWC_10,6), (LWC_18,4), (LWC_24,2)	Writer 2
	3	(LWC_10,2), (LWC_12,2), (LWC_16,2)	Writer 3
Analysis_ set	4	(LWC_10,4), (LWC_16,2), (LWC_24,4)	?

Initially the multinomial naïve_bayes_classifier was utilized, followed by the equations (1) and (2) respectively. P (Writer 1) = 1/3, P (Writer 2) = 1/3, P (Writer 3) = 1/3 are the probabilities. probabilities of every token are given in the table below.

Table 6 Probability measurement with equation (2)

	X=1	X=2	X=3
P ()	6/42	8/22	4/16
P ()	26/42	2/22	4/16
P ()	2/42	4/22	2/16

Using equation (5)

$P(\text{Writer 1} \mid \text{Id} = 4) \propto (1/3)\ (6/42)^2\ (26/42)\ (2/42) \approx 0.0002$

$P(\text{Writer 2} \mid \text{Id} = 4) \propto (1/3)\ (8/22)^2\ (2/22)\ (4/22) \approx 0.0007$

$P(\text{Writer 3} \mid \text{Id} = 4) \propto (1/3)\ (4/16)^2\ (4/16)\ (2/16) \approx 0.0006$

Therefore, according to the multinomial naïve_bayes_classifier, Id = 4 fit to Writer2. Similarly, In the above dataset, Bernoulli naïve_bayes_classifier can be applied followed by equations (1) and (3). For the above dataset, application of k-NN with equation (4), is shown in Table 7.

Table 7 Computation of KNN equation (4)

	Id = 4
Id = 1	$\sqrt{((2-2)^2 + (1-12)^2 + (2-0)^2)} = 11.18$
Id = 2	$\sqrt{((2-3)^2 + (1-0)^2 + (2-1)^2)} = 1.73$
Id = 3	$\sqrt{((2-1)^2 + (1-1)^2 + (2-0)^2)} = 234$

Thus, seeing the computation provided in Table 7, It can be conclude that, Id = 4 fits Writer2.

The above computations shown, provides us a clear knowledge on how the classification algorithms can be used in order to distinguish program writers. In order to classify program files written by respective writers, our tool uses the same classification algorithms.

V. CONCLUSION AND FUTURE WORK

In our research work we discussed the three machine learning methods in order to improve the sharpness of the plagiarism checker. The central idea is to use a meta-learning algorithm for plagiarism detection. The different layers act as filter system to check the possible plagiarism even up to the phase 6. It was fascinating to see the detection of source code plagiarism using the basic machine learning algorithms. adaboost is not the only meta learning algorithm we can use for combining various weak- learners. In future we will be working on the improvement of our system in order to overcome the various other issues related to the source code plagiarism.

REFERENCES

1. J. Zobel, "Uni Cheats Racket: A case study in plagiarism investigation,"Proceedings_of_the_sixth_conference on Australasian computing education-Volume 30,2004, pp. 357–365.
2. J.H. Ji, G. Woo, and H.G. Cho, "A_source_code_linearization technique_for_detecting_plagiarized_programs," ACM SIGCSE Bulletin, vol. 39,2007, p. 77.
   H. Ding and M.H. Samadzadeh, "Extraction_of_Java_program fingerprints_for_software_authorship_identification," Journal of Systems and Software, vol. 72, 2004, pp. 49–57.
3. S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 2002.
4. R.E. Schapire, "A_brief_introduction_to_boosting," International JointConference on Artificial Intelligence, 1999, pp. 1401–1406.
5. Developer Resources for Java Technology.", http://www.oracle.com/technetwork/java/index.html [Accessed: Jan 25, 2011]
6. T. Parr, The_Definitive_Antlr_Reference: Building Domain-Specific Languages, Pragmatic Bookshelf, 2007.

AUTHORS PROFILE

S. Priya, is an Assistant Professor at SRM Institute of Science & Technology. She has over 10 years of experience in teaching. She got her Master's degree from Anna University. Main area of research interest is Network Security

Anukul Dixit is an Undergraduate Scholar pursuing Computer Science & Engineering from SRM Institute of Science and Technology. He is working under the guidance of Mrs. S. Priya.

Krishanu Das is an Undergraduate Scholar pursuing Computer Science & Engineering from SRM Institute of Science and Technology. He is working under the guidance of Mrs. S. Priya.

Ronak Harish Patil is an Undergraduate Scholar pursuing Computer Science & Engineering from SRM Institute of Science and Technology. He is working under the guidance of Mrs. S. Priya