Detection of Source Code Plagiarism Using Machine Learning Approach

Upul Bandara and Gamini Wijayrathna

***Abstract*—Source code plagiarism is currently a severe problem in academia. In academia's programming assignments are used to evaluate students in programming courses. Therefore, checking programming assignments for plagiarism is essential. If a course consists of a large number of students, it is impractical for a human inspector to check each assignment. Therefore, it is essential to have automated tools in order to detect plagiarism in the programming assignments.**

Majority of the current source code plagiarism detection tools are based on structured methods. Structural properties of a plagiarized program and the original program differ significantly. Therefore, it is hard to detect plagiarized programs with tools based on structural methods, when the plagiarism level is four or above.

This paper proposes a new plagiarism detection method, which is based on the attribute counting technique. Novelty of our method is that, we have utilized a meta-learning algorithm in order to improve the accuracy of our plagiarism detection system.

***Index Terms*—Plagiarism detection, machine learning, source code, naïve bayes classifier, k-nearest neighbor**

I. Introduction

Detection of source code plagiarism is valuable for both the academia and industry. Zobel [1] has pointed out that, "students may plagiarize by copying code from friends, the Web or so called 'private tutors'". Most programming courses in universities evaluate the students based on the marks of programming assignments. Therefore, it is essential to detect and prevent plagiarism at universities. Moreover Liu and et al [2] have mentioned that, "A quality plagiarism detector has a strong impact to law suit prosecution". Therefore, there is a huge demand for accurate source code plagiarism detection systems from both the academia and industry.

Woo and Cho [3] have mentioned two methods for plagiarism detection.

1) Structured Based Method: this method considers the structural characteristics of documents when developing plagiarism detection algorithms.
2) Attribute Counting Method: this method extracts various measurable features (or metrics) from documents. Extracted metrics are used as input for similarity detection algorithms.

Presently most of the source code plagiarism detection algorithms are based on the structured method [3], [4], [2]. In addition to that there are few attempts which are based on the attribute counting method [5], [6].

Faidhi and Robinson [7] have defined a spectrum of six levels in program plagiarism. Level 0 is the lowest level of plagiarism, which represents copying someone else's program without modifying it. Level 6 represents the highest level of plagiarism, which is modifying the program's control logic in order to achieve the same operation. It is to be noted that when moving from level 0 to level 6, structural characteristics of the plagiarized program varies from the original program. Moreover, Arwin and Tahaghoghi [4] have mentioned that plagiarism detection systems which use the structured techniques rely on the belief that, the similarity of two programs can be estimated from the similarity of their structures. Since structured properties of plagiarized documents vary from its original document, it is difficult to detect plagiarism when level is four or higher.

On the other hand plagiarism detection systems which are based on the attribute counting techniques do not rely on the structural properties of the source program. Therefore, they are not affected from the problem mentioned above. Presently systems which are based on the attribute counting technique are not accurate enough for practical applications [5], [6].

Therefore, we have proposed a new system which is based on the attribute counting technique and uses machine learning approach in order to detect similarities between source codes. Ethem Alpapaydin [8] has pointed out that, "There is no single learning algorithm that in any domain always induces most accurate leaner". Further, he has mentioned that by combining multiple base learners in a suitable way the prediction performance can be improved. Therefore, instead of using just one learning algorithm, we have used three learning algorithms for training our system. We tested our system with source codes belonging to ten developers. During the training period we found out that not a single algorithm was capable of identifying the source code files belong to all the developers with adequate accuracy. But one interesting observation was that the results generated by the three algorithms were complementing each other. Therefore, we decided to use a meta-learning algorithm in order to combine the results generated by the three learning algorithms. More details about the learning algorithms and the meta-learning algorithm are given in the Research Design section.

The rest of the paper is organized as follows. Section II we will be presenting plagiarism detection methods based on the attribute counting techniques. Section III we will be discussing machine learning algorithms for plagiarism detection. Section IV we will discuss training and testing our system. Finally, we will conclude our paper by discussing the

Manuscript received June 15, 2012; revised August 1, 2012.

U Bandara is with the *Virtusa* Corporation, Sri Lanka (e-mail: upulbandara@ gmail.com).

G. Wijayarathna is with the Faculty of Science, University of Kelaniya, Sri Lanka (e-mailgamini@kln.ac.lk).

final results and future works of our system in Section V.

II. Related Work

Few research papers have been written on source code plagiarism detection using attribute counting techniques. Steve Engels and et al [5] describe code metrics and feature based neural network for detecting plagiarism. Initially, this method extracts metrics from software source codes. The extracted metrics were used as the input for a feature-based neural network. The output of the feature-based neural network was presented in terms of precision and recall. The precision for cheating cases was 0.6464 and the recall for cheating cases was 0.4158.

Dian and Samadzadeh [9] have published a paper on "identification of source code authors using source code metrics". This technique is based on extracting a large number of source code metrics and in this instance they have used 56 source code metrics. The extracted source code metrics are subjected to various statistical techniques in order to identify the authors of each source code. In our research we were able to gain better results than mentioned in this paper using very few metrics.

Lange and Mancoridis [6] have proposed a source code plagiarism detection method, which uses source code metric histograms and genetic algorithms. Initially, they have extracted source code metrics from software source code files. Then they generated the normalized histogram for each source code metric. The normalized histograms were used as the input for the nearest neighbor classifier. One interesting feature of this research is that, they have used genetic algorithm in order to identify the optimized set of source code metrics. According to their research paper their system is capable of identifying the true author of each source code file with 55 percent accuracy. We feel that 55 percent accuracy is not adequate for real-world applications.

III. Machine Learning Algorithms for Source Code Author Identification

Most of the machine learning algorithms which are suitable for pattern recognition can be used for source code author identification. In this section we will be discussing three such algorithms we used for our research.

A. Naïve Bayes Classifier

This classifier is based on Bayes theorem. When C with small number of classes or outcomes conditional on several features denoted by $F_1, F_2, \dots, F_n$ using Bayes theorem:

$$p(C|F_1, \dots, F_n) = \frac{p(C)p(F_1, \dots, F_n|C)}{p(F_1, \dots F_n)}$$

Using conditional probability:

$$p(C, F_1, \dots, F_n) = p(C)p(F_1, \dots, F_n|C)$$

Using chain rule:

$$p(C, F_1, \dots, F_n) = p(C)p(F_1|C)p(F_2|C, F_1) \dots p(F_n|C, F_1, F_2, F_3, \dots, F_{n-1})$$

Using naïve property:

F_i is conditionally independent of every other F_j for all $i \neq j$ this means

$$p(F_i|C, F_j) = p(F_i|C)$$

Therefore Naïve Bayes model can be written as:

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

Z is a constant which is dependent only on $F_1, \dots, F_n$

We can directly use the Naïve Bayes classifier for our research. In our research C means the number of authors in the experiment and $F_1, F_2, \dots, F_n$ means a set of code metrics extracted from the source code.

Christopher and et al [10] have mentioned two methods to construct the Naïve Bayes classifier as enumerated below.

1) Multinomial Naïve Bayes classifier
2) Bernoulli Naïve Bayes classifier

The main difference between the above two variations is the way they calculate the posterior probability or $p(F_i|C)$. Christopher and et al [10] have described in-depth the details about the Naïve Bayes classifier. For this research we will be using both variations of the Naïve Bayes classifier.

B. k-Nearest Neighbor (kNN) Algorithm

The k Nearest Neighbor Algorithm is one of the simplest machine learning algorithms that is suitable for pattern recognition. The k-Nearest Neighbor (kNN) algorithm often performs well in most pattern recognition applications [11]. 'k' is a parameter in the kNN algorithm. It is necessary to select the correct k value for the kNN algorithm by conducting several tests with various k values. The kNN algorithm is based on the "Euclidian Distance".

Let $X_i = (x_{i1}, x_{i2}, \dots, x_{ip})$ be a source code file with p features or metrics. The Euclidian distance between source code files X_i and X_j is defined as given below:

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

kNN assigns source code files in the testing dataset to the majority class of its k nearest neighbors in the training data set.

kNN simply memorizes all the documents in the training dataset and compares the test document against the training dataset. For this reason the kNN is also known as "memory-based learning" or "instance-based learning".

C. AdaBoost Meta-Learning Algorithm

Boosting is used to boost the accuracy of any given learning algorithm [12]. There are many different kind of boosting algorithms available in the research literature. For this research we used the AdaBoost meta-learning algorithm. According to Russell and Norvig [11], AdaBoost possess a very important property: if the learning algorithms are weak leaning algorithms, AdaBoost will classify the testing dataset perfectly for large number of weak learners.

In AdaBoost each data point in the training dataset is associated with a weight. Initially, an equal weight is assigned to all data points in the training dataset. The first iteration starts with the first weak leaner. After the first iteration, weights of misclassified data points are increased.

Then the second iteration is stared with the second weak learner. Furthermore, the second iteration is carried out with newly calculated weights. This process is repeated until all the classifiers are trained. During the training process, a score is given for each classifier, and the final classifier will be the liner combination of the weak classifier used in the iteration.

The AdaBoost algorithm can be implemented in different ways. Different algorithms calculate the weights associated with each data point in various ways. For this research we used the AdaBoost algorithm described by Russell and Norvig [11].

IV. Training and Evaluation

Ding and Samadzadeh [9] have mentioned that not all source code metrics contribute equally for source code author identification. According to Ding and Samadzadeh [9] layout metrics perform a much important role than the other metrics. In addition to that, Lange and Mancoridis [6] have identified and listed source code metrics that perform well for source code identification. Therefore, we have identified the following nine metrics shown in Table I, which perform well for source code identification

TABLE I: Source Code Metrics for Source Code Author Identification

Metric Name	Description
LineLengthCalculator	This metric measures the number of characters in one source code line.
LineWordsCalculator	This metric measures the number of words in one source code line.
AccessCalculator	Java uses the four levels of access control: public, protected, default and private. This metric calculates the relative frequency of these access levels used by the programmers.
CommentsFrequencyCalculator	Java uses three types of comments. This metric calculates the relative frequency of those comment types used by the programmers.
IndentifiersLengthCalculator	This metric calculates the length of each identifier of Java programs.
InLineSpaceInlineTabCalculator	This metric calculates the whitespaces that occurs on the interior areas of non-whitespace lines.
TrailTabSpaceCalculator	This metric measures the whitespace and tab occurrence at the end of each non-whitespace line.
UnderscoresCalculator	This metric measures the number of underscore characters used in identifiers.
IndentSpaceTabCalculator	This metric calculates the indentation whitespaces used at the beginning of each non-whitespace line.

In order to extract some of the metrics, it was essential to parse source codes files according to the syntactic rules of the programming language which was used to write that source code. Since our dataset consisted only Java [13] source code files, we used ANTLR [14] parser in order to extract some of the source code metrics.

For each code metric we gave a unique code as shown in Table II. Thereafter, we converted each source code file into a set of tokens together with token frequencies. This process is explained in the next section.

A. Generating a Set of Tokens from Source Code Files

As we discussed in the previous section, for each source code metric we assign a three letter unique code as shown in Table II

TABLE II: Coding System of Source Code Metrics

Code Metric	Coding System
LineLengthCalculator	LLC
LineWordsCalculator	LWC
AccessCalculator	ACL
CommentsFrequencyCalculator	CFC
IndentifiersLengthCalculator	ILC
InLineSpaceInlineTabCalculator	INT
TrailTabSpaceCalculator	TTS
UnderscoresCalculator	USC
IndentSpaceTabCalculator	IST

For example consider "LineLengthCalculator" code metric generates the following metric as shown in Table III for a particular source code file.

TABLE III: Output of LineLengthCalculator Metric

Length of the Line	Number of Occurrences
5	12
8	20
15	13
20	9
32	11
38	3
40	2

Using the coding scheme introduced in Table II, we can deduce the token frequencies for the metric shown in Table III. The tokens with token frequencies are shown in Table IV.

TABLE IV: Token Frequencies of LineLengthCalculator Metric

Token	Token Frequency
LLC_5	12
LLC_8	20
LLC_15	13
LLC_20	9
LLC_32	11
LLC_38	3
LLC_40	2

As described above, we can represent each source code file as a set of tokens together with token frequencies. Those tokens and token frequencies are used as inputs for our learning algorithms. This process is almost identical to the use of word and word frequencies in document classification problems.

B. Training Dataset

We used the same dataset used by Lange and Mancoridis [6], which consists of Java source code files belonging to 10 developers. We divided the dataset into two parts called the "training dataset" and "validation dataset". The training dataset consisted of 904 source code files. Each developer has written at least 40 source code files. The validation dataset was used as a "hide-out" dataset and it was used only for the final evaluation of the system. The validation dataset consisted of 741 source code files.

C. Training the System

First, the system was trained by using multinomial naïve

bayes classifier. We started with 100 source code files with 10 files per developer. We increased the number of files per training set by 100 files in each iteration. After checking the "confusion matrices" we found out that the multinomial naïve bayes classifier performed well when the training set consisted of 800 source code files.

Thereafter, multinomial naïve bayes classifier tested with the hideout dataset According to the generated "confusion metric" the multinomial naïve bayes classifier's prediction performance is poor for Authors 5, 6 and 8.

As the second step we trained the bernoulli naïve bayes classifier. Similar to the way we trained the multinomial naïve bayes classifier we increased the training dataset by 100 source code files at a time. Moreover we have found out that the bernoulli naïve bayes classifier performs well when the training dataset consists of 800 source code files.

According to the generated "confusion matric" for the Bernoulli naïve bayes classifier we gained some improvement for Authors 8 and 5. But Bernoulli naïve bayes was lacking of classifying source code files of Author 6.

Thereafter, we investigated the k-Nearest Neighbor (kNN) classifier with the intension of improving the accuracy when classifying Author 6. Since k is a parameter in the kNN learner we needed to train the leaner for several k values. As in the previous two learners we trained the kNN leaner for various sample sizes. Finally, we identified the optimum k value and the sample size for the source code author identification problem.

Further, we calculated the confusion matrices for various k values similar to the way we calculate for the naïve bayes classifier. Fig. 1 shows the variation of success rates for various k values.

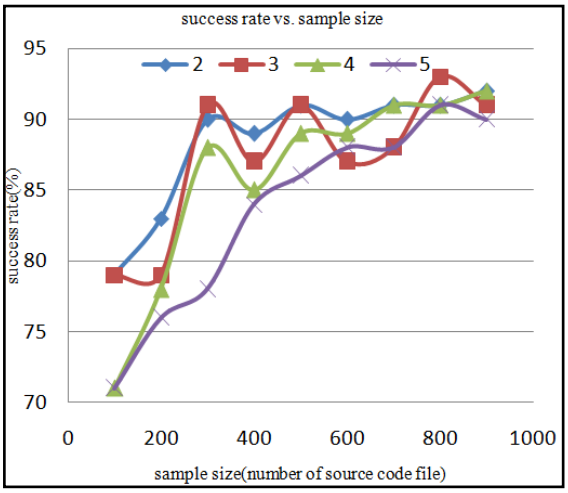

Fig. 1. Success rate vs. sample size of kNN algorithm for training dataset.

According to Fig. 1 the optimum training data set size is 800 documents and k value is 3.

Finally we tested the kNN learning algorithm with the above configuration against the hideout data set. According to the final confusion matrix, the kNN leaner performed well for all developers except for developer number 8. As such we can consider the kNN leaner as a suitable weak leaner to construct an ensemble learner in the next section.

After training the system with three different learning algorithms we have identified no single algorithm that satisfactorily classified the source code files of all the authors. But all the algorithms satisfactorily classified the source code files belonging to a subset of authors. Further the above three learning algorithms complemented each other. Since the three algorithms we trained above complement each other we can use the ensemble learning method to improve the overall accuracy of the process. As such we have used the AdaBoost algorithm for improving the accuracy of the process.

For training the individual weak-learner we used 800 source code files with no less than 40 source code files for each developer and for training the AdaBoost algorithm we used 800 source code documents with not less than 40 documents per author. Fig. 2 shows the confusion matrix of running the AdaBoost for our validation dataset.

We were able to achieve 86.64 percent accuracy by using the same dataset used by Lange and Mancoridis[6]. According to the research paper published by Lange and Mancoridis[6] their accuracy was 55 percent.

		Predicted Author											
		#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	Tot:	Suc: %
Actual Author	#1	16	0	0	0	0	0	0	0	1	0	17	16 94.12
	#2	0	238	4	0	0	1	0	1	0	4	248	238 95.97
	#3	0	0	25	0	0	0	0	1	2	0	28	25 89.29
	#4	0	1	0	19	0	0	0	0	0	0	20	19 95.00
	#5	0	0	0	0	19	0	0	0	2	0	21	19 90.48
	#6	0	5	0	2	0	45	0	2	1	5	60	45 75.00
	#7	0	0	0	0	1	0	54	1	1	0	57	54 94.74
	#8	0	1	0	2	8	1	2	141	32	3	190	141 74.21
	#9	0	3	4	2	1	0	1	1	44	1	57	44 77.19
	#10	0	0	0	0	0	0	1	1	41	43	41	95.35

TOTAL FILES CLASSIFIED : 741
TOTAL SUCCESSES : 642
TOTAL FAILURES : 99
SUCCESS RATE : 86.6397

Fig. 2. Confusion matrix of adaBoost algorithm for hideout dataset.

V. Conclusion and Future Work

In this paper we discussed a machine learning based method for plagiarism detection. The main feature of our method is that, we used a meta-learning algorithm in order to improve the prediction accuracy of our system.

In our research we investigated only three learning algorithms. However, it is interesting to see how other learning algorithms work in source code author identification problems. Furthermore, AdaBoost is not the only meta-learning algorithm we can use for combining several weak-learners. In the future we will be investigating other learning algorithms to combine with our weak-learners.

Our system will not work correctly, if programmers follow some coding standard and source code formatting tools specified in their projects. Since our main target was

References

[1] J. Zobel, "Uni Cheats Racket: A case study in plagiarism investigation," *Proceedings of the Sixth Conference on Australasian Computing Education*, vol. *30*, 2004, pp. 357–365.

[2] C. Liu, C. Chen, J. Han, and P. S. Yu, "GPLAG: detection of software plagiarism by program dependence graph analysis," *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006, pp. 881.

[3] J. H. Ji, G. Woo, and H. G. Cho, "A source code linearization technique for detecting plagiarized programs," *ACM SIGCSE Bulletin*, vol. 39, 2007, pp. 77.

[4] C. Arwin and S. M. M. Tahaghoghi, "Plagiarism detection across programming languages," *Proceedings of the 29th Australasian Computer Science Conference*, vol. 48, 2006, pp. 286.

[5] S. Engels, V. Lakshmanan, and M. Craig, "Plagiarism detection using feature-based neural networks," *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, 2007, pp. 38.

[6] R. C. Lange and S. Mancoridis, "Using code metric histograms and genetic algorithms to perform author identification for software

forensics," *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, 2007, pp. 2089.
[7] J. A. W. Faidhi and S. K. Robinson, "An empirical approach for detecting program similarity and plagiarism within a university programming environment," *Computers and Education*, vol. 11, 1987, pp. 11–19.
[8] E. Alpaydin, *Introduction to Machine Learning, Second Edition*, the MIT Press, 2010.
[9] 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.
[10] M. C. D, R. Prabhakar and S. Hinrich, *Introduction to Information Retrieval*, Cambridge University Press, 2008.
[11] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, 2002.
[12] R. E. Schapire, "A brief introduction to boosting," in *Proc. of International Joint Conference on Artificial Intelligence*, 1999, pp. 1401–1406.
[13] Developer Resources for Java Technology. [Online]. Available: http://www.oracle.com/technetwork/java/index.html.
[14] ANTLR Parser Generator. [Online]. Available: http://www.antlr.org/ [Accessed: Jan 25, 2011].

Upul Bandara received the B.Sc in Civil Engineering and M.Sc.in IT from University of Moratuwa, Sri Lanka in 2004 and 2011. His major research interests include machine learning, information retrieval and compiler construction. Presently, he works for Virtusa Corporation, Sri Lanka as a senior engineer.

Gamini Wijayarathna received Dr. Eng. degree (specializing in Software Engineering) and M.Eng. degree from the University of Electro-Communications, Tokyo, Japan. He graduated from the Faculty of Science, University of Kelaniya with an Honors degree in 1984. He has received special training in System Engineering by the Center of the International Cooperation for Computerization (CICC) in Tokyo, Japan and IBM System 34/36 by IBM Sri Lanka. He has over twenty years of experience in professional software development for various business and scientific applications, conducting research in software engineering related fields, and teaching Information Technology related subjects. His client base includes private and public sector organizations in Sri Lanka, and few Japanese companies. Dr. Wijayarathna worked as a Research Associate at the Software Design Laboratory, Graduate School of Information Systems, University of Electro-Communications, Tokyo, Japan before he joined the Department of Industrial Management as a senior lecturer in year 2001. During last few years, he has extended his services to other institutes in Sri Lanka.