DETECTING OUTSOURCED STUDENT PROGRAMMING ASSIGNMENTS*

Bruce S. Elenbogen and Naeem Seliya
University of Michigan – Dearborn
4901 Evergreen Rd., Dearborn, MI 48128
{boss, nseliya}@umich.edu

ABSTRACT

The task of writing computer programs outside of class is the most realistic experience students have in a programming class and hence can be the most accurate evaluation of their ability. However some students hire outside parties to produce these programs. We present a data mining and machine learning approach that can provide objective evidence for detecting such instances. Based on programs submitted by students across two lower-level CS (Computer Science) courses, we extract some basic programming style metrics. A decision tree model built on the collected measurements yields relatively good detection accuracy. In addition, an investigation into relative importance of the basic style metrics was performed which indicated Lines of Code, Number of Variables, and Number of Comments as important attributes. The methods are being implemented in a software analysis tool that instructors could possibly use for detecting outsourced program submissions.

1. INTRODUCTION

Students have attempted to cheat in classes for as long as there have been class grades. In the past however, the scarcity of programming skills has forced computer sciences students who desire to cheat to rely mainly on aid from their classmates. Several tools [1, 2, 6, 8] have been written to identify if students in the same class submit essentially the same program with superficial or minor changes. These tools measure the similarity of functionality of programs, usually through common sequences of tokens.

In the last couple of years there has been a growth of unscrupulous third parties [7] who sell programming solutions to students. This is largely due to the growth of the

---

internet and the growth of computer science talent worldwide, coupled with the wide disparity in the worlds' costs of living. A programmer in many third world nations can potentially earn a good income just by completing undergraduate assignments, even when a cut of the profits is taken by the sponsoring web sites. Programmers from more prosperous regions can also find the profit well worth the time spent to complete simple assignments. Thus the suppliers of these programs have a powerful economic incentive to continue this practice despite the obvious moral and ethical concerns.

The aforementioned plagiarism detection schemes will not work on these outsourced programs since there are a large number of programmers and websites which makes it unlikely that someone else in the same class will hire the same program author. Even if a CS instructor was willing to spend the time scanning websites for evidence of homework solicitation, many sites allow users to hide details of these transactions from everyone but the parties involved, making plagiarism detection very difficult.

We present a data mining-based approach for identifying such fraudulent programs. The method is based on developing a *programming style profile* for a student from her/his past programming assignments. We note the preliminary nature of this study; however, our results provide sufficient interest for further empirical analysis. We believe a sufficient proof of student program outsourcing can be achieved with the proposed analysis. Our result lends itself toward representing significant objective evidence that program outsourcing has occurred. In addition, we feel a data mining perspective to student plagiarism is quite unique.

The rest of the paper is structured as follows. Section 2 further elaborates the problem and presents concepts of our methodology; Section 3 includes the data analysis and results; and Section 4 summarizes our work and includes practical considerations on implementation of the detection scheme as well as suggestions of future work.

2. PROPOSED APPROACH

2.1 Programming Profiles

Researchers have been developing programming profiles since the early 1990s [3, 4, 10]. These have been developed mainly to help identify authors of malicious software. These programs attempt to identify elements of a person's programming that are independent of the purpose of the program. We will refer to these elements as programming style. These profiling schemes are based on the premise that the authors' style is relatively stable and will not vary significantly between different programs regardless of the purpose of the program. Additionally these profiles are built from an examination of dozens of previous programs created by the author.

Both of the aforementioned underlying assumptions (stable programming style and large body of previous work) concerning profiles are generally not valid in a classroom setting. Student programming styles change dramatically during their college tenure. In fact, the goal of many elementary classes in CS is to help students change their style toward conforming to the style requirements of professionals. Style requirements and preferences made by individual computer science instructors tend to obscure the programming and development style used by students.

Additionally, the number of programs that a student submits during any given semester is too small to constitute a representative sample from which to develop a meaningful programming profile. However programming instructors have an advantage over the profilers of suspected malicious programmers. Computer science instructors tend to have dozens of other students who have written programs that have similar or identical functional goals. It is through comparisons with other students that we hope to develop a programming profile.

The method is based on an intuitive assumption that although a student's style may change dramatically over the course of a semester, it may not vary much in relation to the rest of his or her class. For example, if an instructor requests more comments on the next programming assignment, all students will tend to provide more comments on their next program. However, the student who turned in the fewest comments in prior programs may still write fewer comments relative to the class. We believe this to be true as students in the same class are affected by similar forces of change.

The student population in our study has all been subject to identical lectures and requirements from the same instructor. In addition, the comments instructors give to individual students is usually given to a large percentage of students who make similar errors. It is because of this shared classroom experience, that we believe that relative programming style characteristics can be used to construct an effective style profile.

2.2 Programming Style Measurements

In contrast to the schemes that measure the similarity of programs from the students in the same class, we measured only programming style elements that do not affect functionality. The six metrics used in this study included: number of lines of code, number of comments, average length of variable names, number of variables, number of for-loops/number of total loops, and the number of bits in the zipped (compressed) program. In our study relative values of these metrics are used, representing the student's deviation from other students of the class.

This initial set of metrics were considered based on the instructors heuristic- and experience-based knowledge of typical student programming preferences from CS1 and CS2 classes. Note that all programming assignments were given to students by the same instructor. The metrics chosen were similar to those used in the literature for identification of programming style [2, 3, 4, 5]. They represent choices students make in choosing looping structures, verboseness, naming conventions, and efficiency of their code. A larger set of programming style metrics is planned to be used in future versions of our software analysis tool. Each program of a student was measured using the six metrics and their average of each metric was used to create a style vector. This vector approach allows a unique set of metrics to be used for individual students.

2.3 Case Study Data and Analysis

We take a data mining approach [11] to determining whether a program submission by a student is authored by the same student. The empirical goals of our preliminary study are: (1) develop a classification model which can detect any diversion from a student's underlying coding style, and (2) investigate which style metrics are important for a given

student and the general student population. Compared to some related work [3, 5], we approach the problem from a knowledge discovery perspective which finds patterns that are generally hidden.

The programming style measurement dataset is obtained from 12 students each of whom completed seven programming assignments; however, one of the students submitted only six programs. The same instructor provided these assignments from two lower-level CS courses. The six style measurements are recorded for each program of every student, giving a total of 83 instances of programs. An instance (program) is assigned a class label of either A, B, C, …, or L representing one of the students. We have a dataset consisting of 83 instances and 12 classes.

Toward the first goal of our study, we use the C4.5 decision tree model to build a classification model for the dataset. We use the implementation of C4.5 obtained from the Java-based data mining tool[i], Weka [11]. The decision tree model extracts data patterns in the form or rules based on the available instances and their measurements. These rules can then be used by instructors and pedagogy researchers toward predicting authorship deviance of student programming assignment submissions.

The second goal of our study is to investigate the relative importance of the six style metrics considered in our study. We are interested in learning which of the style metrics are more important for a given student. In addition, an insight into whether all the collected metrics are useful in characterizing the programming style of students is warranted for our future studies.

The original dataset is modified to create 12 different two-group datasets. More specifically, for student A we create a dataset consisting of instances labeled as either class A or class Not_A. Similarly, a two-group dataset is created for each of the students. The aim here is to identify relative importance of the style metrics for a given student compared to the other students combined. This can provide the analyst with insight into important characteristics to identify for a suspicious student.

A two-sample two-sided Kolomogorov-Smirnov (K-S) [9] test is used for each of the two-group datasets. Given a two-group (e.g. X and Y) dataset, the K-S distance of an attribute is the maximum difference between the cumulative distribution functions (CDFs) of its X and Y classes. The higher the K-S distance of an attribute, the higher is its significance in identifying the two classes [9].

3. RESULTS AND DISCUSSION

The K-S statistic is used to determine the relative importance of the six style metrics with respect to their significance in segregating a given student from the others. In addition, the K-S statistical test determines which of the six style metrics is significant, at a given α level, in identifying a given student from the others. We consider two significance levels, 95% and 90%. The latter was considered because analysis with α = 95% did not provide conclusive insight into significant metrics for all students.

[1] Additional details on the data mining algorithm used are not presented due to paper size limits.

For a given student, the relative importance of the six style metrics is shown in Table 1. The six metrics are denoted as 1, 2, 3, 4, 5 and 6 each representing Zipped Size, Lines of Code, Number of Variables, Number of For Loops, Number of Comments and Variable Length, respectively. For example, Lines of Code is denoted as 2 and Number of Comments is denoted as 5 respectively. A close inspection of Table 1 suggests that the Lines of Code metric tends to be of more importance for detecting a given student from the others. For example, students A, B, G, and H are associated with Lines of Code as the most significant metric. Along the same lines, Number of Comments and Variable Length are also of high importance in characterizing student programs, and they often are ranked in the top three metrics for a given student.

A notable inference is that Number of For Loops (relative to total number of loops) is generally of lesser significance compared to the other programming style metrics. The top three metrics that are generally better at detecting students in our dataset are Lines of Code, Number of Comments, and Variable Length. The Number of Variables also shows as important for a few students in our dataset. We do note that Table 1 does not provide a clear-cut preference of style metrics for each student. This is an indication that additional programming style metrics are needed to possibly provide a better characterization of student programming profiles.

The C4.5 decision tree model is built using Weka with default parameters, such as tree depth and minimum observations in a node [11]. A practical advantage of using a decision tree learner is that a white-box model is built, allowing the analyst to clearly observe how the different metrics play into building the model. The classification model yielded an accuracy of 74.70% based on a 10-fold cross validation technique. Given the small size of the training dataset (83 instances), a cross validation approach is used to estimate classification accuracy. With the trained model, 62 programs were correctly classified (74.7%) while the remaining 21 were misclassified (25.3%).

The classification performance for individual students (a student is a class) is summarized in Table 2. Given a student X, the True Positive performance represents percentage of X instances that were predicted as X, while the False Positive performance represents percentage of Not X instances that were predicted as X. A good classification model has high True Positive values and low False Positive values. Among the 12 groups, programming assignments of students J, K, and F tend to be more difficult to classify (detect) with the six style metrics. This is observed by their respective low True Positive percentages. In contrast, programming assignments of students A, C, D, E, and G can be detected with good accuracy, i.e. 85% or higher.

A likely issue concerning the poor detection of programs by J, K, and F is that the six style metrics may not be sufficient enough to delineate code written by these students from those of the other students. This is further evidenced by our study in which two-group classification models are built using a dataset of X and Not_X students, where X is A, B, C, …, or L. The subsequent decision tree models built for J, K, and F indicated that very few of these students can be correctly separated from the other students with the given style metrics. Hence, there is a logical need to investigate other coding style metrics that can properly characterize these difficult-to-detect students.

In addition to the above data mining studies, we investigated a simple analysis based on deviation of a given (suspicious) program from the author's other programs. This

method measured the difference to the mean of all metrics for a given program author. When we set the cutoff for a suspicious program to be two standard deviations away from the mean, we correctly identified 73% of the programs and found 6% of them suspicious when compared to their real author. A more detailed empirical analysis on determining optimal value for variance from the mean is out of scope for this paper.

4. CONCLUSION

Our preliminary finding makes the proposed approach a viable tool to identify outsourced programs for student coding submissions. However, the most easily identifiable programs generally belonged to the best programmers who have the least need to outsource their assignments. This can be further investigated by exploring coding style measurements unique to difficult-to-detect programmers. We note the relative small size of the dataset warrants additional empirical evidence of the proposed approach, and that such a data collection in currently in progress.

Although the detection scheme had some trouble identifying the worst students from each other, it should have not any trouble identifying a professional program as being written by a different author, since a professional's style has the greatest difference from

Table 1: Metrics importance per student

Student	High ---------- Low					
A	2	3	1	4	6	5
B	2	6	3	5	1	4
C	6	1	2	4	3	5
D	3	4	2	5	6	1
E	5	6	2	4	1	3
F	5	6	4	3	2	1
G	2	3	5	1	6	4
H	2	4	3	6	5	1
I	1	3	2	6	4	5
J	6	5	2	3	4	1
K	6	1	3	5	2	4
L	3	5	6	2	1	4

Table 2: Classification performance

True Positive	False Positive	Student
85.70%	3.90%	A
71.40%	2.60%	B
85.70%	2.60%	C
85.70%	3.90%	D
85.70%	0.00%	E
66.70%	1.30%	F
100.00%	5.30%	G
71.40%	1.30%	H
71.40%	2.60%	I
42.90%	1.30%	J
57.10%	2.60%	K
71.40%	0.00%	L

the worst students. Given the high number of students to detect from each other, one may consider clustering similar students according to their programming profiles. Toward this goal the grades awarded to student for their programs could provide a good metric for grouping similar students. Related empirical investigation is planned.

The analysis presented here is dependent on a cache of old programs. The sample programs we used had the advantage of all being obtained from the same instructor, adding to the consistency of expectations. Based on this sample, we determined a minimum of five programs are needed to make an adequate baseline for student programming profiles. Prior student work is now routinely stored as representative examples and as part of accreditation reviews. We suggest that CS departments may want to consider electronically archiving all student programs for a four year period. This could form an excellent source for input into our analysis scheme.

Future work will include refinement and expansion of the metrics and analysis illustrated in this paper as well as an automated detection tool. The tool would take as input old student programs, the suspicious program and its author, and return the likelihood of it being written by the author on record. A long term goal is to pursue program outsourcing detection regardless of the programming language used, providing a more universal and practical application of the designed methods. Additional enhancements would include extracting boiler-plate code supplied by the instructor to all students when assigning programming tasks.

REFERENCES

[1] Aiken, A. MOSS: A System for Detecting Software Plagiarism (unpublished), http://www.cs.berkeley.edu/~aiken/moss.html.

[2] Chen, X., Francia, B., Li, M., McKinnon, B., and Seker, A. Sharing information and programming plagiarism detection, *IEEE Trans. on Information Theory*, 50(7):1545-1551, July 2004.

[3] Ding, H. and Samadzadeh, M. H. Extraction of Java program fingerprints for software authorship identification. *Jour. of Systems and Software*, 72(1):49-57, 2004.

[4] Gray, A., Sallis, P., and MacDonell, S. Software forensics: extending authorship techniques to computer programs, In *Proc. of 3rd Biannual Conf. of the Intl. Association of Forensic Linguists*, pages 1-8, 1997.

[5] Oman, P. W., and Cook C. R. Programming style authorship analysis, In *Proc. of 17th ACM Annual Computer Science Conf.*, pages 320-326, 1989.

[6] Prechelt, L., Malpohl, G. and Philippsen, M. Finding plagiarisms amongst a set of programs with JPlag, *Jour. of Universal Computer Science*, 8(11):1016-1038, 2002.

[7] Ross, K. Academic dishonesty and the Internet, *Communications of the ACM*, 48:(10): 29-31, October 2005.

[8] Schleimer, S., Wilkerson, D. and Aiken, A. Winnowing: local algorithms for document fingerprinting, In *Proc. of SIGMOD 2003*, June 9-12, 2003, San Diego, CA.

[9] Seber, G. F. Multivariate Observations, John Wiley, New York, NY, 1984.

[10 Spafford, E, H. and Weber, S. A., Software forensics: can we track code to its authors, *Purdue Technical Report CSD–TR 92–010, SERC Tech. Report 110–P*.

[11] Witten, I. H. and Frank, E. *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd Ed., Morgan Kaufmann, San Francisco, CA, 2005.