An Approach to Detect Abnormal Submissions for CodeWorkout Dataset*

Alex Hicks*, Yang Shi, Arun-Balajiee Lekshmi-Narayanan, Wei Yan and Samiha Marwan

Dept of Computer Science, Virginia Tech, Blacksburg, VA
Dept of Computer Science, Utah State University, Logan, UT
Intelligent Systems Program, University of Pittsburgh, Pittsburgh,PA
School of Informatics, Computing and Cyber Systems, North Arizona University, Flagstaff, AZ
Dept. of Computer Science, University of Virginia, wCharlottesville, VA

Abstract

Students' interactions while solving problems in learning environments (i.e. log data) are often used to support students' learning. For example, researchers use log data to develop systems that can provide students with personalized problem recommendations based on their knowledge level. However, anomalies in the students' log data, such as cheating to solve programming problems, could introduce a hidden bias in the log data. As a result, these systems may provide inaccurate problem recommendations, and therefore, defeat their purpose. Classical cheating detection methods, such as MOSS, can be used to detect code plagiarism. However, these methods cannot detect other abnormal events such as a student gaming a system with multiple attempts of similar solutions to a particular programming problem. This paper presents a preliminary study to analyze log data with anomalies. The goal of our work is to overcome the abnormal instances when modeling personalizable recommendations in programming learning environments.

Keywords

CS1, Introductory Programming, Dataset Cleaning, Dataset Standards, Educational Data Mining

1. Introduction

Students cheating to submit programming solutions is a common occurrence. Cheating can be of any kind – copying solutions to the problem available online, by other students learning programming with the course or by other means of plagiarism. Generally, researchers have explored methods to curb cheating in the context of academic integrity [1]. Some techniques that could work [2] include the detection of collusion and continual feedback to students to encourage them towards better academic integrity. There is a tendency for students to cheat when solving programming puzzles or practice assignments. When online log data is collected using the interaction logs of the interfaces for programming assignments, there is a risk for some of these anomalies to be recorded among regular student interaction logs. This could potentially affect student modeling approaches that use the interaction logs to make recommendations for students [3].

Student modeling in the context of solving programming assignments like the Normalized Student Modeling for Programming [4] use Error Quotient and Watwin score that measure changes help estimate student knowledge or understanding [4, 5]. In other cases, student modeling facilitates the identification and prediction of students' learning profiles in tutoring systems, which, in turn, enables such systems to be adaptive and personalized to students' needs [6]. This makes them sensitive to the quality of the data and anomalies created by students gaming the system or cheating / plagiarizing solutions may cause the model to *overestimate* or *underestimate* student knowledge or understanding of the introductory programming concepts.

For example, a study conducted by Hellas et al. found instances where students copied content to complete their assignments [7]. This behavior can significantly compromise the quality of student modeling approaches applied to these data. Moreover, these cheating instances may lead to erroneous predictions, revealing a threat to the field of student modeling technology.

Another example discussed by Sosnovsky and colleagues [8] discusses student modeling anamolies observable as sudden changes in the learning rate of a student when learning with an adaptive educational system. This could be attributed to any form of assistance offered to the student by a more experienced or knowledgeable peer indicated Low-High-Low or High-Low-High patterns in the student's learning rate.

To address this challenge, researchers have developed tools for detecting plagiarism in students' code (e.g., [9]). One of the most popular approaches is "The Measure Of Software Similarity (MOSS)", an open-source tool designed to identify similarities between students' programming assignments [9]. However, to our knowledge, there is no evidence that researchers apply cheating detection methods on online shared data before applying log data analysis and student modeling.

We present a work in progress, where we look into this aspect closely in order to mitigate anamolies in student submissions : 1) using classical methods like Measure of Software Similarity (MOSS), 2) alternative approaches of analyzing log data). We use the CodeWorkout (CWO) programming dataset (as introduced in [10])[1]. While the use of generative AI has been very popular now, this dataset was collected before 2021 when Generative AI was not generally used to cheat when submitting programming solutions.

CSEDM'24: 8th Educational Data Mining in Computer Science Education (CSEDM) Workshop, June 14, 2024, Atlanta, GA
*You can use this document as the template for preparing your publication. We recommend using the latest version of the ceurart style.
*Corresponding author.
awh4kc@vt.edu (A. Hicks); yang.shi@usu.edu (Y. Shi); arl122@pitt.edu (A. Lekshmi-Narayanan); wei.yan@nau.edu (W. Yan); samihamarwan21@gmail.com (S. Marwan)
https://awhicks.github.io/ (A. Hicks); https://a2un.github.io (A. Lekshmi-Narayanan); https://weiyanedtech.com/ (W. Yan); https://www.samihamarwan.com/ (S. Marwan)
0000-0002-2143-2633 (A. Hicks); 0000-0001-6486-4340 (Y. Shi); 0000-0002-7735-5008 (A. Lekshmi-Narayanan)

[1]https://pslcdatashop.web.cmu.edu/DatasetInfo?datasetId=3458

	avg_score	median_score	first_score	last_score	n_attempts	one_shot	condition
X-Grade (before)	-0.209966	-0.187436	-0.287831	0.062299	0.220599	-0.359111	Unclean
X-Grade (after)	0.326563	0.440488	0.457784	0.333629	0.376384	0.234074	Clean

Table 1
Comparison between correlation data across potential indicators with final course grade before and after cleaning suspicious submissions.

Figure 1: MOSS results for comparing two CWO submissions

2. Methods & Analysis

In this work, we compare two ways to analyze abnormal submissions:

Proposed method: Log Data Analysis. We used two main identifiers to explore anomalies such as suspected cheating behaviors from submission log data: the number of submission attempts before completing the exercise, and the elapsed time between correct submissions. The choice of these variables correlates with the possibility that students who attempt and submit a correct solution on their first attempt could be cheating. We discuss more details on this below.

Baseline method: MOSS. MOSS is a tool used to detect cheating in programming submissions. The tool works by taking into all the students' submissions and comparing them pairwise for similarities. We compared students' code submissions using MOSS to identify similarities in submissions for a selected set of problems from a collection of easy, medium, and hard assignments made available on CWO.

3. Results & Discussions

3.1. MOSS Detection Results

We further evaluated whether accessible and common cheating detection tools such as MOSS can be applied to detect students' cheating in this dataset. However, we found that running MOSS across CWO exercises led to high rates of similarity on a majority of students' submissions. In addition, we found no clear difference between students whom we previously identified and those whom we believe that have engaged authentically with the CWO exercises. We hypothesize that this failure could be due to the size of the solutions to several CWO exercises. Some solutions to these exercises could be just 10 lines of source code as these problems are well-constrained and target specific learning goals. Hence, these problems may not have possible alternative solutions (refer Figure 1). Students like those in the example may end with 93% of their solutions matching despite no indications of anomalous behaviour. This indicates that identifying an acceptable threshold for MOSS detection on CWO exercises is unreasonable and highlights the need for other options.

3.2. Log Data Analysis Detection

We calculate students' "one shot" percent, or the percent of CWO exercises where a student correctly answers an exercise on their first attempt. In Table 1, this is represented as the one_shot column and is calculated as a correlation with the student's final course grade. Once this value was calculated, we were able to compare the differences between the correlations on a student's first score on a given problem to how often they were getting their first attempt fully correct and found a suspicious difference. Figures 2 shows the relationship between the first scores of the students' submission to the exercises and their final exam scores, and the figure on the right shows the distribution of the first scores of the students' submission. While many students perform well on

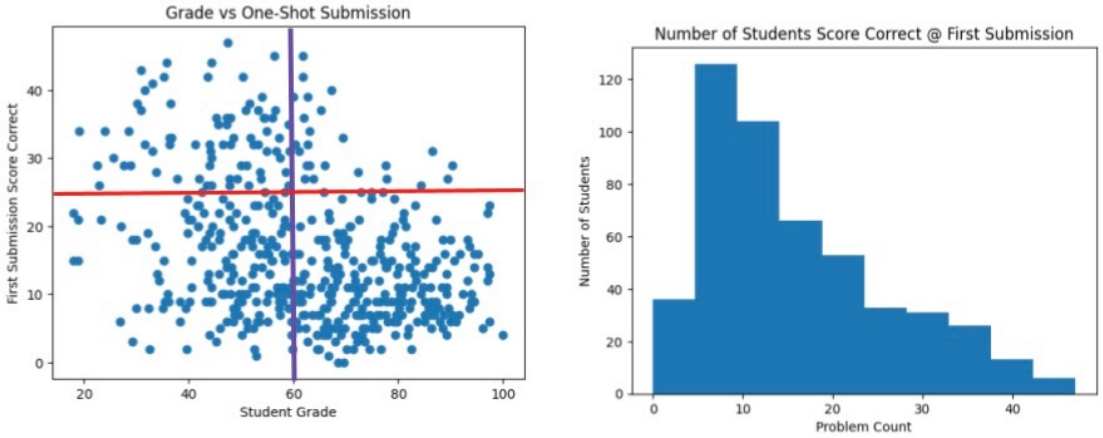

Figure 2: Cluster plot of student first submission scores on exercises (y-axis) and their final grades(x-axis) (left); Histogram of the number of students (y-axis) with the number of exercises they achieved correctly (x-axis) on first submissions.

their first submissions of exercises, showing their mastery of programming skills, only a small subset match this performance in the course as a whole. Specifically, students who perform well in the CWO exercises on their first attempt, often do not perform well for their final grade of the course. This preliminary data analysis did not make intuitive sense and led us to further investigate this phenomenon using more traditional methods, including MOSS.

4. Limitations and Future Work

This preliminary investigation focused only on CWO submissions, but we hope this data cleaning approach can be generalized to other datasets that use the ProgSnap2 format. We also hope to continue investigating the metadata about submissions included in this format to find more accurate indicators of cheating behavior in the programming snapshot data. While MOSS is generally used to compare final students' submissions with other final students' submissions, in future work, we will consider the case for running MOSS with sequential data where submissions made on platforms like CWO that allow multiple submissions. For example, we could compare attempt 1 of a student 1 with attempt 2 of student 2 and so on to see if a students copy each others' solutions from their first attempt onwards or after trying multiple attempts, failing and then cheat to proceed to the next programming problem on the CWO platform.

Acknowledgments

We thank the contributions by Dr. Thomas Price for his guidance on this work. We also thank the 2023 Session of LearnLab Summer School Organizers and our sponsors Dr. Peter Brusilovsky and SPLICE project PI(s) for bringing us all together to work on this.

References

[1] S. E. Allen, R. F. Kizilcec, A systemic model of academic (mis) conduct to curb cheating in higher education, Higher Education (2023) 1–21.

[2] O. Karnalim, Simon, W. Chivers, B. S. Panca, Educating students about programming plagiarism and collusion via formative feedback, ACM Transactions on Computing Education (TOCE) 22 (2022) 1–31.

[3] P. Brusilovsky, E. Millán, User models for adaptive hypermedia and adaptive educational systems, in: The adaptive web: methods and strategies of web personalization, Springer, 2007, pp. 3–53.

[4] A. S. Carter, C. D. Hundhausen, O. Adesope, The normalized programming state model: Predicting student performance in computing courses based on programming behavior, in: Proceedings of the eleventh annual international conference on international computing education research, 2015, pp. 141–150.

[5] T. W. Price, D. Hovemeyer, K. Rivers, G. Gao, A. C. Bart, A. M. Kazerouni, B. A. Becker, A. Petersen, L. Gusukuma, S. H. Edwards, et al., Progsnap2: A flexible format for programming process data, in: Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education, 2020, pp. 356–362.

[6] R. Umer, T. Susnjak, A. Mathrani, L. Suriadi, Current stance on predictive analytics in higher education: Opportunities, challenges and future directions, Interactive Learning Environments 31 (2023) 3503–3528.

[7] A. Hellas, J. Leinonen, P. Ihantola, Plagiarism in takehome exams: Help-seeking, collaboration, and systematic cheating, in: Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE '17, Association for Computing Machinery, New York, NY, USA, 2017, p. 238–243. URL: https://doi.org/10.1145/3059009.3059065. doi:10.1145/3059009.3059065.

[8] S. Sosnovsky, L. Müter, M. Valkenier, M. Brinkhuis, A. Hofman, Detection of student modelling anomalies, in: Lifelong Technology-Enhanced Learning: 13th European Conference on Technology Enhanced Learning, EC-TEL 2018, Leeds, UK, September 3-5, 2018, Proceedings 13, Springer, 2018, pp. 531–536.

[9] K. W. Bowyer, L. O. Hall, Experience using "moss" to detect cheating on programming assignments, in: FIE'99 Frontiers in Education. 29th Annual Frontiers in Education Conference. Designing the Future of Science and Engineering Education. Conference Proceedings

(IEEE Cat. No. 99CH37011, volume 3, IEEE, 1999, pp. 13B3–18.
[10] Y. Shi, R. Schmucker, M. Chi, T. Barnes, T. Price, Kc-finder: Automated knowledge component discovery for programming problems., International Educational Data Mining Society (2023).