Detecting Contract Cheaters in Online Programming Classes with Keystroke Dynamics

Jeongmin Byun
School of Computing, KAIST
Daejeon, South Korea
jmbyun@kaist.ac.kr

Jungkook Park
School of Computing, KAIST
Daejeon, South Korea
pjknkda@kaist.ac.kr

Alice Oh
School of Computing, KAIST
Daejeon, South Korea
alice.oh@kaist.edu

ABSTRACT

In online programming classes, it is tricky to uphold academic honesty in the assessment process. A common approach, plagiarism detection, is not accurate for novice programmers and ineffective for detecting contract cheaters. We present a new approach, cheating detection with keystroke dynamics in programming classes, and evaluated the approach.

Author Keywords

online education; academic honesty; keystroke dynamics; contract cheating

INTRODUCTION

Assessment is a critical element in education, and a typical approach in online programming classes is to give take-home assignments and exams for student assessments. However, although these are effective learning tools, it is tricky to uphold academic honesty. Previous research shows that 7% of undergraduate students admitted to turning in papers written by someone else, and 3% admitted to obtaining essays from essay mills [6] for class assignments. We believe that online programming assignments are an easier target for cheaters. Since there are no in-class activities and proctored exams, students can easily hire other people to work on their tasks. Previous work shows that students are more likely to cheat when the student-instructor relationship is weaker; this is a known problem in online classes.

One of the conventional approaches to prevent this problem is to use the plagiarism detection software, and plagiarism detection for program code started in the 1970s before the plagiarism detection for natural language started. Academic misconduct in programming classes can be detected with code clone detection algorithms like Deckard [4] based on measuring code similarity, and code stylometry algorithms [1] based on n-gram frequencies. However, these algorithms may not be appropriate for novice programmers; program codes submitted in introductory programming classes are often too short and simple to distinguish. Figure 1 shows the example of a similar code submitted as answers to a programming exercise.

A

```
def merge(input_file, output_name):
    fo = open(output_name, "w")
    for i in range(len(input_file)):
        fi = open(input_file[i], "r")
        for j in fi:
            fo.write(j)
        fi.close()
fo.close()
```

B

```
def merge(inp,outp):
    x = open(outp, "w")
    for i in range(len(inp)):
        y = open(inp[i], "r")
        for j in y:
            x.write(j)
        y.close()
x.close()
```

Figure 1. A "merge" function from two different students on an exercise task. Two students separately authored program code (A) and (B), but they are hard to distinguish from each other based on the code content. Since the ASTs (abstract syntax trees) from both of the codes are the same, code clone detection algorithms detect the codes as clones, and codes are too short and similar to each other to use code stylometry algorithms.

However, when a document is written using a keyboard, there is another way to identify the author of the document. Keystroke dynamics based algorithms allow author identification based on authors' keyboard typing patterns. These algorithms are known to be very accurate and have no error in the right conditions if authors are typing fixed text [5], and have about 95% of precision and 99.9% recall on detecting fraud for long free-form natural language text [3]. In a free-form text written in natural language, the algorithm utilizes the typing pattern for frequent n-grams (e.g., a frequent sequence of letters in English stop words) to detect an anomaly. There are no stop words in program codes, but there are reserved keywords for programming languages that frequently appear in the text. Therefore we expected that keystroke dynamics based algorithms would show high accuracy with program codes, too.

Based on this idea, we built a prototype system that analyzes keyboard-related events observed in the web-based programming exercises in online programming classes. The system observes all pressing and releasing actions from the author's activity at the millisecond level, informing the instructor and the staff if anomalies are found.

For evaluation, we ran an experiment with 27 students in an *Introduction to Programming* course in KAIST. The result shows that our system achieves an accuracy of 94.8% on classifying the author of program code, and 85.9% precision and 84.4% recall for detecting contract cheaters.

L@S '20, August 12–14, 2020, Virtual Event, USA.

ACM ISBN 978-1-4503-7951-9/20/08 ...$15.00.
http://dx.doi.org/10.1145/3386527.3406726

The main contributions from this work are:

- A system that records and analyzes keystroke dynamics to classify the code authors and detect contract cheaters.
- An evaluation of the system with the data collected from the real university class.

RELATED WORK

Plagiarism Detection

Plagiarism detection software has been used to handle the academic misconducts in various classes, and programming classes are no exception. Previous research shows that code stylometry algorithms and code clone detection algorithms can be used to find cases of academic misconduct in programming assignments.

Code stylometry algorithms identify unique programming styles from the program codes to identify the authors. There are various ways to implement codes that do the same task; we may expect to find unique programming styles from the program codes authored by one author. Previous work on this topic shows that authors of program codes can be identified by observing byte-level n-grams or word-level n-grams [1]. Also, there are attempts to identify the program code author based on the code structure represented in the form of AST (abstract syntax tree), and this method can be applied to the executable binaries after a decompilation process. Meanwhile, there are also studies on code clone detection algorithms [4], where similar techniques are used to measure the similarity between two program codes instead of identifying the authors of the codes.

However, code stylometry and clone detection algorithms are often not applicable to detect plagiarism in the introductory programming classes. Programming exercises designed to teach basic programming concepts like conditionals, loops, functions, or classes and instances usually ask students to write short and simple code due to their lack of programming skills, and this makes the students write similar-looking codes that are not distinguishable from each other especially when they are converted to ASTs. Figure 1 shows a pair of codes in our dataset where two students separately wrote and ASTs for them are identical to each other.

Compared to these algorithms, our approach is relatively accurate to use for plagiarism detection in introductory programming classes. Students write short program codes only, but they still include multiple reserved keywords and built-in function names. Therefore observing keystroke dynamics from the students' programming sessions allow us to detect the anomaly when someone tries to hire a third person and attempt contract cheating.

Keystroke Dynamics

Keystroke dynamics have often been used to improve the security in the ID-password user authentication process. Since the pattern of latency between keyboard events is unique to the individuals, previous studies have proposed methods and systems that identify or verify the person who typed the keyboard to enter a password. In the early stage, researchers verified the people based on the keyboard events observed while everyone is typing the same static text (e.g., password) [5]; and studies that try to identify the person typing free-form texts appeared later [3]. Researchers tend to collect datasets for their own research since it is hard to find the datasets for keystroke dynamics analysis. Hence, it is hard to compare the performance of the existing algorithms.

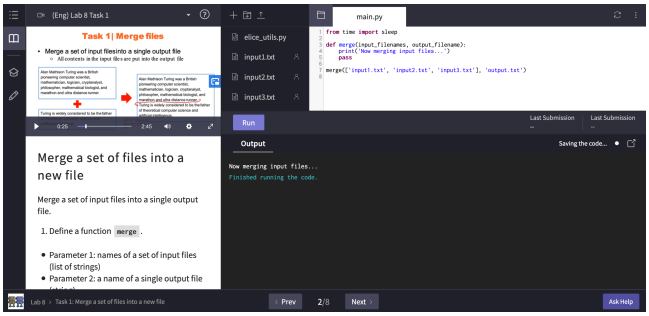

Figure 2. An overview of the user interface for web-based programming exercises. This web page from our system supports students with a programming environment, including lecture videos, task descriptions, a code editor, and a code runner. Students can learn, read the task description, write and run the code, and grade the code in a single web page, while server observes the keyboard events triggered by the students in the page.

There are also keystroke dynamics research that uses keystroke dynamics analysis in other tasks. For example, some tried to measure the stress level of the users based on the keystroke dynamics, while some tried to find if a person is lying or not based on the keystroke dynamics [7].

Although several studies already developed high-performance anomaly detection methods based on keystroke dynamics, they are all either designed to work on a static text or free-form text contents written in natural languages. In this work, we propose to use a keystroke dynamics based method to detect contract cheating cases in program codes, develop a web-based system that observes anomaly in programming code writing sessions, and evaluate the system with 27 students in a university course, *Introduction to Programming*.

DESIGN

Web Application for Programming Exercises

We built a prototype system that hosts web-based programming hands-on exercises that allows the server to observe the keyboard events triggered by students while they write and run program codes on a web browser. Figure 2 shows an overview of the web page for programming exercises. Since the system provides the lecture video, exercise task description, code editor, and code runner with grader in the single web page, students can conveniently work on the programming exercises within our system's web page.

Also, programming libraries or input file examples required for solving the programming exercises are only available in our system, so that students naturally have to work on the exercises directly on our web page. This is important for preventing the contract cheating because when students can work on the exercise out of our system, they can hire a third person to work on their exercise and copy the finished code by typing the code. When students work on the programming

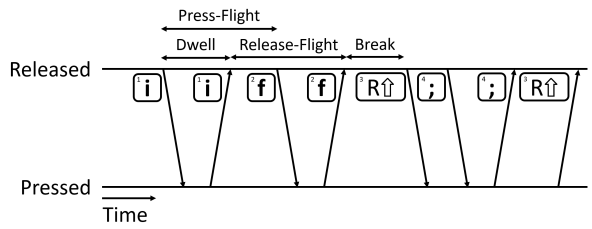

Figure 3. Keyboard key-press and key-release events while typing "if:" represented as downward and upward arrows on the time axis. Common types of time intervals used as elements of a feature vector that represents a keyboard typing session are indicated on the figure.

exercises on our web page, our system observes all keyboard press events and release events with millisecond precision to analyze and detect the contract cheaters.

Analysis on Keyboard Events

Keystroke dynamics contain various timing information composed of all keyboard press and release events triggered during the document editing session. Previous research shows that time intervals between specific timestamps from two consecutive keyboard inputs are useful when building a feature vector for classification or anomaly detection from keyboard events. For example, there are a few types of time interval values that commonly appear in multiple research work like press-flight time, release-flight time, dwell time, and break time. According to these types, typing "`if:`" on the keyboard could be interpreted as a series of time intervals like in Figure 3 [8].

EVALUATION

To evaluate our system, we observed and analyzed the keystroke dynamics data from the students who take *Introduction to Programming* class in KAIST. The class covers the basic programming concepts and skills using Python 3, and we used the data from the students worked on their programming exercises in two offline lab sessions where they practice how to use mathematical operators and file input/output (27 students, 5 exercise tasks in total 6 hours). We verified all students' identity to make sure no one is trying contract cheating in our experiment. Based on the keystroke dynamics observed from the students, we extracted feature vectors from each of the typing sessions for solving a programming exercise and applied anomaly detection algorithms to find the session authored by someone else than the expected student.

Feature Extraction

We extracted feature vectors to use in anomaly detection from the keyboard dynamics data observed from the students solving the programming exercises. There are a few effective ways to extract the keyboard dynamics based feature vectors from free-form text keyboard typing sessions suggested by previous research, but they used the data collected from the natural language typing sessions. Therefore, we tried to find a compact keyboard dynamics based feature vector that includes the necessary information from programming language typing sessions.

We are extracting keystroke dynamics based features from a free-form text typing, so we decided to use various time interval values from most frequently appearing digraphs and trigraphs, and dwell time for frequently appearing characters (monographs). We ran two preliminary experiments to decide the set of time interval values and n-graphs to use in building the feature vectors and evaluated the final feature extraction method by comparing the performance against the methods suggested in previous studies. Many previous studies based on keyboard dynamics in free-form text typing are focused on the author classification task, so we have used classification accuracy from author classification for this procedure to evaluate our method against them.

Classification Method	Acc.
Random Baseline	0.037
Clone Detection (Deckard) [4]	0.052
Stylometry (Lexical & Layout) [2]	0.193
Keystroke (All Digraphs) [8]	0.830
Keystroke (Python Keywords) [8]	0.533
Keystroke (Common N-graphs)	**0.948**

Table 1. Accuracy scores from classification compared with baseline methods.

Time Interval Values

To find the best time interval values to use in our feature vectors, we ran 5-fold cross-validation of author classification on our data for all possible combinations of time interval values (dwell, press-flight, release-flight, and break), with all possible combinations of top k monographs, l digraphs, and m trigraphs that are most likely to be found in the source code where $0 \leq k, l, m \leq 10$. Results show that including *dwell* time in the feature vector is required to get the best accuracy in over 90% of the n-graph sets. Other than that, an optimal set of the time interval values depends on the set of n-graphs used.

N-graphs

To find the optimal set of n-graphs for building feature vectors, we ran 5-fold cross-validation of author classification using k monograph, digraph, and trigraph that are most likely to be found in our data to build feature vectors ($1 \leq k < 100$). The result shows that the classification accuracy is higher when more n-graphs are used in the range of $1 \leq k \leq 30$, but the accuracy converges around $k = 30$. We had the highest accuracy at $k = 35$ in $1 \leq k < 100$, when *dwell* time and *release-flight* time are used to build the feature vector.

We tried to find and omit insignificant features in this feature vector, so we applied the feature elimination algorithm on the elements. However, omitting any of the elements resulted in a lower classification accuracy score; thus, we omitted no element from the feature vector.

Meanwhile, the accuracy scores after omitting features extracted with each of the n-graph give us a brief insight on which n-graphs are most effective in feature vector building. The top five n-graphs that cause the biggest loss of the accuracy score when omitted from the feature vector are: monograph *"T"*, trigraph *"Spacebar = Spacebar"*, monograph *"B"*, digraph *"T E"*, and trigraph *"O R Spacebar"*, respectively.

Comparison with Baseline Methods

We compared and assessed our feature extraction method with baseline methods from previous studies. We used the random forest classifier with our feature vectors since it resulted in the highest mean accuracy score among six commonly used classifiers in *scikit-learn* library. We applied various methods to the author classification task on our data (with 5-fold cross-validation).

Table 1 shows the accuracy scores achieved with baseline methods and our method. Baseline methods show significantly lower accuracy scores compared to the scores from their original experiments; this may be caused by relatively short and simple program codes in our data, or the small size of our data. As a result, our method outperformed all baseline classification methods in the experiment with an accuracy of 94.8%.

Contract Cheater Detection

We used the same data again for the experiment to evaluate our method on contract cheater detection. However, we replaced student *A*'s typing session for one of the programming exercises *E* with another student *B*'s session to simulate the data from the case where student *A* is hiring a third person *B* to solve the exercise *E*. All other records from the student *B* are removed from the data since we are supposed not to have any other data from *B*. We applied our method to detect student *B*'s session from this simulated data, and we repeated this process for all possible combinations of student *A*, *B*, and exercise *E* for 27 students and 5 exercises.

Although this task is anomaly detection, we used a random forest classifier with a modified threshold on prediction. We labeled all student A's sessions as class 1 and all other sessions as class 0, and trained the classifier. We considered the session in test data as an anomaly when the probability of this session to be class 1 is under 90%. We collected all results from the contract cheater detection tasks for every combinations of *A*, *B*, and *E*, and the average precision and recall value are 85.9% and 84.4% respectively. This makes a false acceptance rate (FAR) of 15.6% and a false rejection rate (FRR) of 14.1%.

CONCLUSION

In this work, we proposed a new approach to detect contract cheaters with keystroke dynamics in online programming classes and developed a prototype system to evaluate our method. Unlike other approaches to detect academic dishonesty in online classes, our method can be used in introductory programming classes with novice students. We evaluated our system with a real university class and students, and showed that the system detects contract cheating cases with higher accuracy than the previous studies; the false acceptance rate (FAR) is 15.6% and the false rejection rate (FRR) is 14.1%. We expect our system to notify staff and instructors about suspicious cases for further investigation, and reduce contract cheating in online programming classes.

On the other hand, we found some limitations to our work, but we may be able to improve our system by resolving them as future work. First, our system has a relatively high false acceptance rate and false rejection rate, which makes it not usable as a sole verification method for filtering out contract cheaters. Since we used one of the traditional algorithms to handle anomaly detection cases and there are modern algorithms that are shown to work better in many cases, we may be able to improve our system's performance by utilizing one of those algorithms. Also, our system cannot detect the case where students are copying others' code but typing the code in the editor by themselves. Traditional plagiarism detection algorithms are designed for this case, but they are not very effective to use for novice programmers. We expect that we can detect this kind of case with keystroke dynamics analysis, too, since the lying person can be detected with the keystroke dynamics [7]. We have plans to use keystroke dynamics analysis to distinguish the typing session where the person is merely copying the content from other regular sessions.

ACKNOWLEDGEMENTS

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT (2017M3C4A7065962).

REFERENCES

[1] Steven Burrows and Seyed MM Tahaghoghi. 2007. Source code authorship attribution using n-grams. In *Proceedings of the Twelth Australasian Document Computing Symposium, Melbourne, Australia, RMIT University*. Citeseer, 32–39.

[2] Aylin Caliskan-Islam, Richard Harang, Andrew Liu, Arvind Narayanan, Clare Voss, Fabian Yamaguchi, and Rachel Greenstadt. 2015. De-anonymizing programmers via code stylometry. In *24th USENIX Security Symposium (USENIX Security), Washington, DC*.

[3] Daniele Gunetti and Claudia Picardi. 2005. Keystroke analysis of free text. *ACM Transactions on Information and System Security (TISSEC)* 8, 3 (2005), 312–347.

[4] Lingxiao Jiang, Ghassan Misherghi, Zhendong Su, and Stephane Glondu. 2007. Deckard: Scalable and accurate tree-based detection of code clones. In *Proceedings of the 29th international conference on Software Engineering*. IEEE Computer Society, 96–105.

[5] Marcus Karnan, Muthuramalingam Akila, and Nishara Krishnaraj. 2011. Biometric personal authentication using keystroke dynamics: A review. *Applied soft computing* 11, 2 (2011), 1565–1573.

[6] Donald L McCabe. 2005. Cheating among college and university students: A North American perspective. *International Journal for Educational Integrity* 1, 1 (2005).

[7] Merylin Monaro, Chiara Galante, Riccardo Spolaor, Qian Qian Li, Luciano Gamberini, Mauro Conti, and Giuseppe Sartori. 2018. Covert lie detection using keyboard dynamics. *Scientific reports* 8, 1 (2018), 1976.

[8] Terence Sim and Rajkumar Janakiraman. 2007. Are digraphs good for free-text keystroke dynamics?. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE, 1–6.