Shih, JL. et al. (Eds.) (2023). Proceedings of the 31st International Conference on Computers in Education. Asia-Pacific Society for Computers in Education

Investigating Programming Performance Predictability from Embedding Vectors of Coding Behaviors

Ikkei IGAWA[a]*, Yuta TANIGUCHI[a] , Tsubasa MINEMATSU[a] , Fumiya OKUBO[a] & Atsushi SHIMADA[a]
[a]*Kyushu University, Japan*
* igawa@limu.ait.kyushu-u.ac.jp

Abstract: Understanding students' coding behaviors is crucial for providing targeted support in programming education. Automatic analysis of coding behaviors using machines can address the limitations of manual monitoring. Previous studies focused on coding behavior representations without considering differences relative to a model answer. We propose embedding vectors that capture these differences, enabling the distinction between simple and complex code solutions. Evaluating these vectors by predicting assignment scores, we achieved over 15% higher accuracy compared to conventional methods. This approach has the potential to enhance teachers' understanding of students' coding behaviors and improve support in programming education.

Keywords: Programming education, educational data analysis, CodeBERT, programming performance prediction

1. Introduction

Understanding student coding behaviors in programming courses offers several advantages, including identifying struggling students and pinpointing areas of difficulty. This enables teachers to provide targeted support and effective feedback, enhancing students' understanding of programming (Leinonen et al., 2019). However, directly monitoring all students' programming activities is not feasible for teachers due to large class sizes and limited resources. Therefore, the use of machine-based automatic analysis of student coding behaviors becomes essential.

Previous studies have conducted the automation of programming log analysis (Mao et al., 2021). However, many of them do not consider the difference between student codes and a model answer when calculating coding behavior representations. By considering these differences, it becomes feasible to distinguish between students who have correctly solved a problem with simpler code closer to the model answer and those who have written more complex code. To address this, we propose the generation of coding behavior representations that consider the deviation from the model answer. Our objective is to utilize these representations, derived from the coding activity logs of students during programming assignments, to predict their performance accurately. By achieving accurate predictions of assignment scores based on coding behavior representations, we demonstrate that these representations effectively capture factors that affects their programming performances.

2. Proposal Method

To create embedding vectors that capture students' coding behaviors, including their ability to write code successfully and smoothly, as well as their tendency to explore different approaches through trial and error, we propose some embedding methods.

First, we utilize the pre-trained CodeBERT model (Feng et al., 2020) without fine-tuning to convert students code into embeddings. CodeBERT is a pre-trained bimodal model for PLP with Transformer-based neural architecture. Next, to compute an embedding vector for each student's coding behavior, we utilize the answer code provided for a given programming assignment. By considering the answer code as a reference, we quantify the characteristics of a student's coding behavior relative to this reference code. To accomplish this, we introduce the concept of answer-directed vectors (ADVs), which are vectors pointing from each code written by a student to the answer code.

After calculating the ADV for each student's code, we generate two vectors: the mean of the ADVs (ADV Mean) and the sum of the ADVs (ADV Sum). We expect that struggling students will have a larger ADV Sum due to their trial-and-error approach and significant code differences. The ADV Sum can be calculated like below:

$$\sum_{i=1}^{n} \left(\boldsymbol{e}_{\mathrm{AnswerCode}} - e^{i}_{\mathrm{Code}} \right),$$

Whereas $\boldsymbol{e}_{\mathrm{AnswerCode}}$ indicates the embedding vector of the answer code, and e^{i}_{Code} indicates the embedding vector of the *i*-th code written by a student to solve the assignment. Therefore, we can calculate the ADV Mean by dividing ADV Sum by *n*.

In this study, we evaluate the ADV Mean and ADV Sum as representative vectors for student coding behaviors, examining their effectiveness in capturing and representing the nuances of students' approaches to programming assignments.

3. Experiment

3.1 Experiment Settings

To assess the effectiveness of the generated vectors in representing students' coding behaviors, we conducted an experiment to predict students' programming assignment scores. We utilized WEVL (Taniguchi et al., 2022), an online coding application used in programming courses at our university, to analyze the programming logs collected during a Python programming exercise course in 2020. Throughout the course, students received weekly programming assignments to be completed within a week.

We categorized students into two groups based on their scores: "Comp." (students who obtained full scores) and "Incomp." (students who did not). Then, we tested the predictability of the vectors by predicting which students were in Incomp., using machine learning. For binary classification, we employed a Random Forest with a maximum depth of 5. We then performed a *k*-fold cross-validation with *k*=3 and calculated the average scores of the four types of prediction performance metrics: recall, precision, accuracy, and f-measure.

In this experiment, we examined the "concat" assignment given during the middle stages of the course. The number of programming logs was 6,301, Among the 63 students who engaged with the "concat" assignment, 32 students belonged to the Incomp. group, while the remaining 31 students belonged to the Comp. group.

3.2 Results

Table 1 shows the prediction results of each vector in a programming assignment given in the middle stage of the course. For comparison, we used edit distance and embedding distance to calculate how the student code was different from the answer code. First, we can see that the vector generated by summing up ADV exhibits the highest scores in all the performance metrics. Second, we can see that the sum method outperforms the mean method in prediction scores in almost all the metrics.

Table 1. *Results of prediction on the concat assignment. The highest values for each performance metric are highlighted in bold. The chance rate of the prediction is 0.5079.*

		Recall	Precision	Accuracy	F−measure
Edit Distance	Mean	0.4453	0.5039	0.4762	0.4584
	Sum	0.4913	0.5371	0.5397	0.5105
Embedding Distance	Mean	0.5030	0.5571	0.5397	0.5204
	Sum	0.5939	0.5563	0.5397	0.5643
ADV	Mean	0.6212	0.6474	0.6349	0.6272
	Sum	**0.7698**	**0.7098**	**0.6984**	**0.7215**

4. Discussion

Looking at the Table 1, we can see that the sum method consistently outperformed the mean method across almost all methods. Considering the reason why this happened, a possible explanation is that students who are less proficient in programming and struggle with coding may require repeated trial and error to arrive at a solution. This increase in the number of attempts results in a greater sum. If we consider distance or vector size as penalties, we can accumulate them by employing the sum method, enabling the identification of students struggling with coding. Indeed, both Edit Distance Sum and Embedding Distance Sum are proportional to the number of attempts, but their scores did not increase as much as ADV Sum. This indicates that the direction of the ADV, as well as its size, contributes to prediction. In summary, our findings suggest that the ADV Sum represents factors that significantly affect student performance.

5. Conclusions

This paper presented methods using CodeBERT to convert students' coding behaviors into 768-dimensional vectors. The ADV Sum method, incorporating differences between student and answer code. CodeBERT-based representations outperformed edit distance, providing valuable insights for educators to understand and support students' learning needs in programming education. Enhanced feedback based on these insights can improve student learning outcomes.

Acknowledgements

This work was supported by JST CREST Grant Number JPMJCR22D1, JSPS KAKENHI Grant Number JP18H04125, JP21K17863 and JP22H00551, Japan.

References

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., . . . others (2020). CodeBERT: A pre-trained model for programming and natural languages. arXiv preprint arXiv:2002.08155 .

Leinonen, J. (2019). Keystroke data in programming courses (Doctoral dissertation, University of Helsinki). Retrieved from http://hdl.handle.net/10138/337131

Mao, Y., Shi, Y., Marwan, S., Price, T. W., Barnes, T., & Chi, M. (2021). Knowing when and where: Temporal-ASTNN for student learning progression in novice programming tasks. International Educational Data Mining Society.

Taniguchi, Y., Minematsu, T., Okubo, F., & Shimada, A. (2022). Visualizing source-code evolution for understanding class-wide programming processes. Sustainability, 14 (13), 8084.