Automated Prediction of Novice Programmer Performance Using Programming Trajectories

Miguel A. Rubio(✉)

Department of Computer Science, University of Granada, Granada, Spain
marubio@ugr.es

Abstract. Online programming courses have become widely available and host thousands of learners every year. In these courses, participants must solve programming exercises by submitting partial solutions and checking the outcome. The sequence of partial solutions submitted by a student constitutes the programming trajectory followed by the student.

In our work, we define a supervised machine learning algorithm that takes as input these programming trajectories and predicts whether a student will successfully complete the next exercise. We have validated our model with two different datasets: the first one is a set of problems from the online learning platform Robomission with over one hundred thousand exercises submitted. The second one comprises one hundred thousand exercises submitted to the Hour of Code challenge.

The results obtained indicate that our model can accurately predict the future performance of the students. This work provides not only a new method to represent students' programming trajectories but also an efficient approach to predict the students' future performance. Furthermore, the information provided by the model can be used to select the students that would benefit from an intervention.

Keywords: Machine learning · Introductory programming · Novice programmer · Educational data mining · Block-based programming

1 Introduction

Online programming courses have emerged as a popular way to introduce students to programming [1]. These courses present several advantages: they are easily accessible, and students face interesting challenges. Unfortunately, it is not feasible to provide individual support to each student due to the large number of students enrolled in these courses. Automatic systems capable of providing adaptive support could enhance the students' experience and improve their success rate [2].

In order to develop these automatic systems, there is a need to develop models capable of detecting students that will likely fail [3–5]. These models could use the large datasets that students generate when completing programming tasks [6, 7]. Students usually submit several partial solutions before solving a task, creating a programming trajectory for each exercise [8, 9]. These programming trajectories can be analyzed by machine learning systems to find general patterns [10].

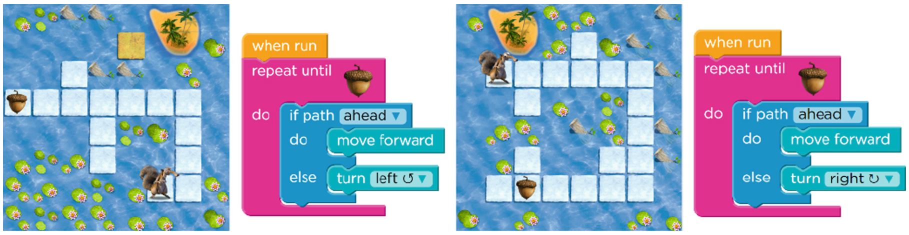
In this study we present a supervised machine learning model that predicts the student future programming performance. The model takes the programming trajectory followed by the student and estimates the probability of the student successfully completing the next exercise. The model has been validated using two different datasets obtained from two different online programming environments, Robomission [11], and the Hour of Code challenge from Code.org [12].

Our results indicate that this model can predict accurately whether a student will be able to successfully complete a programming exercise. The information provided by the model can be used to rank students in terms of their performance. Using this ranking one can automatically select a group student that would benefit most from an intervention.

2 Methods

2.1 Data

In this study we worked with two different datasets. The first dataset is a set of programming trajectories submitted by students while completing one exercise in the Hour of Code challenge [13]. Additionally, for each student the dataset contains information about whether the student successfully completed the next task. The exercises and their solutions are shown in Fig. 1. Piech et al. [8] describe this dataset in more detail. The second dataset comprises 85 programming tasks from the Robomission programming platform. Effenberfer [14] gives a thorough description of the dataset.



```
when run
repeat until
do   if path ahead
     do   move forward
     else turn left ↺
```

```
when run
repeat until
do   if path ahead
     do   move forward
     else turn right ↻
```

Fig. 1. Hour of code exercise 18 (left) and exercise 19 (right) and example solutions. To solve the exercise the student must program the squirrel to reach the acorn.

2.2 Proposed Model

Our goal is to generate a supervised machine learning algorithm capable of predicting whether the student will successfully complete the next exercise. To this end we will use the programming trajectories followed by the students T = $\{\psi_0, \psi_1 \ldots \psi_n\}$. Where ψ_0 is the state before the student starts to work, ψ_i are the code snapshots submitted by the student and ψ_n is the last snapshot.

The training phase is straightforward: all the programming trajectories present in the training dataset are assembled into a tree. Different branches of the tree contain information about different programming trajectories. Figure 2 describes the process to integrate a new trajectory {ψ_0, ψ_1, ψ_5} into a tree. For each code snapshot present in the trajectory we check if there is a branch in the tree with matching snapshots. If there is such a branch, we follow it while the partial solutions match. As soon as we find a partial solution (ψ_5 in this case) that is not present in the branch, a new branch is created.

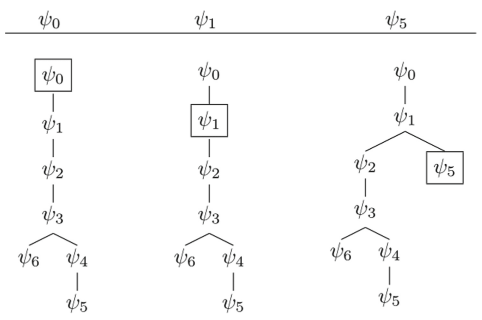


Fig. 2. Steps followed to integrate a new trajectory {ψ_0, ψ_1, ψ_5} into the tree. Two different leaves of the final tree present the same partial solution.

Once we have processed all the student trajectories to generate the tree, we store in each node the relevant parameters of the students that ended their programming trajectories in that node. In this study we stored the proportion of students that successfully completed the next exercise. After assembling the tree, we can estimate the probability that a new student with trajectory T_i will successfully complete the next exercise. If we want to classify the student, we only need to compare this probability with the threshold that we have selected.

We have selected the Receiver Operating Characteristic (ROC) curve [15] and the area under the curve (AUC) to measure the performance of the classifier. We have used a 10-fold crossvalidation [16] stratified over students to compute them. We will compare our model optimal performance with the results of a simple baseline model. Our baseline model expects the performance of both tasks, the one taken as input and the predicted one, to be the same.

3 Results

We start examining whether our model is successfully detecting students who fail the next exercise in the Hour of Code challenge. The left side of Fig. 3 shows that the ROC curve is systematically above the identity line (y = x). The area under the curve (AUC) of our model in this case is 0.77, with a 95% confidence interval (0.77–0.79). Both the AUC and the confidence interval are greater than 0.5, indicating that our model is performing better than a random classifier. Figure 3 also contain the main results for the baseline model and the optimal threshold. We can see that the baseline model is much closer to the bottom left corner of the figure than the optimal threshold.

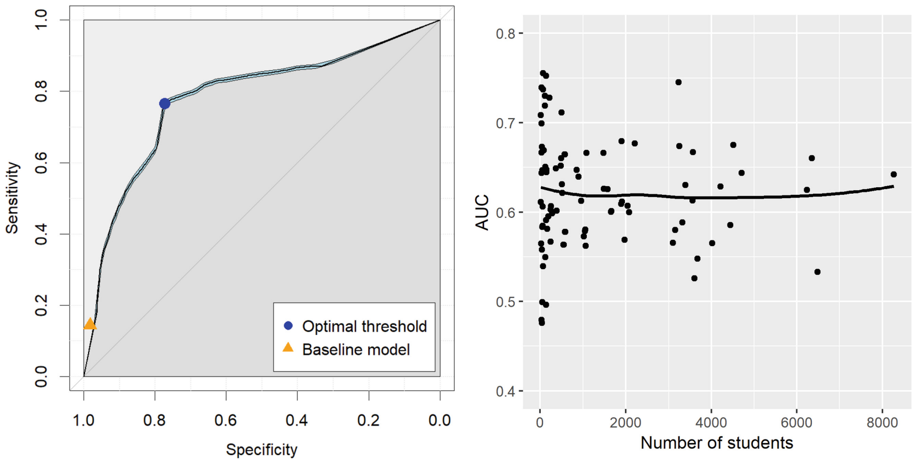


Fig. 3. Left: ROC curve obtained when classifying failing students in the Hour of Code exercise. The cyan region represents the 95% confidence interval. Right: AUC values for all the Robomission tasks vs. the number of students that completed each task. The line represents the loess regression of the data points.

The right side of Fig. 3 shows the AUC obtained for each task in the Robomission dataset versus the number of students that attempted each task. We performed a loess regression [16] looking for a correlation between AUC and the number of students. From the graph we can conclude that there is no such correlation. However, the variability of AUC values depends on the number of students. When the number of students is below 500 the AUC values show high variability. For values over 500 the variability decreases markedly.

4 Conclusions

In this study we present a machine learning algorithm able to predict the future performance of novice programmers using their programming trajectories in just one exercise. The output of the model can be used to rank students according to their predicted performance. The data used by the model can be easily obtained in online programming environments.

We have validated our model using two different datasets from two online learning platforms. Our results indicate that the model can classify students with reasonable accuracy. We have also found that the average performance of our model seems to be independent from the number of students attempting the task.

References

1. Nguyen, A., Piech, C., Huang, J., Guibas, L.: Codewebs: scalable homework search for massive open online programming courses. In: Proceedings of the 23rd International Conference on World Wide Web, pp. 491–502. Association for Computing Machinery, Seoul, Korea (2014)
2. Ihantola, P., et al.: Educational data mining and learning analytics in programming: literature review and case studies. In: Proceedings of the 2015 ITiCSE on Working Group Reports. pp. 41–63. ACM New York, NY, USA (2015)
3. Liao, S.N., Zingaro, D., Laurenzano, M.A., Griswold, W.G., Porter, L.: Lightweight, early identification of at-risk CS1 students. In: Proceedings of the 2016 ACM Conference on International Computing Education Research, pp. 123–131. ACM, New York (2016)
4. Liao, S.N., Zingaro, D., Thai, K., Alvarado, C., Griswold, W.G., Porter, L.: A robust machine learning technique to predict low-performing students. ACM Trans. Comput. Educ. **19**, 18 (2019)
5. Castro-Wunsch, K., Ahadi, A., Petersen, A.: Evaluating neural networks as a method for identifying students in need of assistance. In: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, pp. 111–116. ACM, New York (2017)
6. Glassman, E.L.: Clustering and visualizing solution variation in massive programming classes (2016)
7. Rivers, K., Koedinger, K.R.: Data-driven hint generation in vast solution spaces: a self-improving python programming tutor. Int. J. Artif. Intell. Educ. **27**, 37–64 (2017)
8. Piech, C., Sahami, M., Huang, J., Guibas, L.: Autonomously generating hints by inferring problem solving policies. In: Proceedings of the Second (2015) ACM Conference on Learning @ Scale, pp. 195–204. Association for Computing Machinery, Vancouver, BC, Canada (2015)
9. Jiang, B., Li, Z., Stamper, J.: Programming pathway clustering using Tree Edit Distance. In: CSEDM 2018: Educational Data Mining in Computer Science Education Workshop, pp. 76–84. ACM, The University at Buffalo, New York (2018)
10. Hosseini, R., Brusilovsky, P., Yudelson, M., Hellas, A.: Stereotype modeling for problem-solving performance predictions in MOOCs and traditional courses. In: Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization. pp. 76–84. ACM, New York (2017)
11. Effenberger, T., Pelánek, R.: Towards making block-based programming activities adaptive. In: Proceedings of the Fifth Annual ACM Conference on Learning at Scale. Association for Computing Machinery, London, United Kingdom (2018)
12. Wilson, C.: Hour of code: we can solve the diversity problem in computer science. ACM Inroads. **5**, 22 (2014)
13. Bau, D., Gray, J., Kelleher, C., Sheldon, J., Turbak, F.: Learnable programming: blocks and beyond. Commun. ACM **60**, 72–80 (2017)
14. Effenberger, T.: Blockly programming dataset. In: 3rd Educational Data Mining in Computer Science Education (CSEDM) Workshop (2019)
15. Robin, X., et al.: pROC: an open-source package for R and S + to analyze and compare ROC curves. BMC Bioinformatics **12**, 77 (2011)
16. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. SSS. Springer, New York (2009). https://doi.org/10.1007/978-0-387-84858-7