LetGrade: An Automated Grading System for Programming Assignments

K. N. Nikhila(✉) and Sujit Kumar Chakrabarti(✉)

International Institute of Information Technology, Bangalore, India
{nikhila.kn,sujitkc}@iiitb.ac.in

Abstract. Manually grading programming assignments is time consuming and tedious, especially if they are incorrect and incomplete. Most existing automated grading systems use *testing* or *program analysis*. These systems rely on a single reference solution and award no marks to submissions that differ from the reference solution. In this research, we introduce an automated grading model *LetGrade*. *LetGrade* is a supervised machine learning-based mechanism for automatically identifying the approach of solving and grading a student's submission. The method looks for a score of the similarities between the submitted solution and multiple correct solutions available to determine the solution's approach. The calculated similarity score is then entered into a pretrained supervised machine learning model that grades the submission. Our models were evaluated against datasets containing Python and C programming problems. The average variance in the grade predicted by the supervised machine learning model is consistently close to 0.5. This indicates that the models can accurately predict the grade within a 10% margin of error.

Keywords: Automated evaluation · Static analysis · Testing · Machine learning · Supervised learning

1 Introduction

Increasing enrolment in programming-based courses has led instructors to utilise automated tools in evaluating programming assignments so as to satisfy the demands of quality and objectivity on the one hand, and timeliness on the other. Most approaches rely heavily on testing. However, fundamental limitations of testing, which are well known in software engineering, also manifest in the context of automated evaluation. Moreover, testing can only verify output, so there are additional problems. Therefore, there has been a lot of work done that uses automated techniques based on static analysis.

A typical workflow of a static analysis based evaluation tool would involve a step that computes an estimated structural similarity between a submitted solution and the reference solution [11]. This similarity estimate will then directly translate into the marks awarded to a given solution. Although structural similarity correlates with the logical closeness of a submission to a reference solution, it does not linearly map to the grade. The relation between structural similarity and final marks appears to be at best something monotonic but non-linear. In the context of this paper, we define the step of creating a mapping between structural similarity scores and marks as *grading*. The mapping itself is called a *grading model*. The grading model is nothing but a supervised machine learning model that maps similarity scores to marks.

In this paper, we present our investigations done around devising the above grading model. We have tried a number of well-known regression models as our grading model. We have evaluated these models on submissions collected from online coding platform *HackerRank* [3] and our institute's *Introductory programming course in Python*. Problem-specific models and generalised models are able to predict the grade of incorrect solution with $\pm10\%$ tolerance.

The specific contributions of this paper is a novel approach to automated evaluation of programming assignments with focus on the grading model.

Section 2 introduces the Grading Model. Section 3 discusses related works and Sect. 4 concludes the paper.

2 Approach

2.1 Preliminaries

Some of the principles followed in designing our automated evaluation system are as follows:

1. **Static analysis + testing:** As mentioned in Sect. 1, testing has some inherent shortcomings as an underlying method for automated evaluation. The primary reason for this is that differences between two solutions which produce identical output can not be easily (and in some cases, not at all) detected through testing. Hence, apart from testing, our system also uses static analytic approach: a submission which is structurally more similar to a reference solution is a likely more correct solution.
2. **Support for multiple approaches:** We also observe that there are likely multiple correct approaches to solve a programming problem. It is our purpose to support multiple solution approaches to be considered without making it necessary for the instructor to know about them priorly.
3. **Grading:** We go with the assumption that the verdict of the human grader provides us with the ground truth in evaluation. Hence, our system involves a final step where we map similarity value computed by our system to actual grade by passing them through a grading model, which is the main topic of this paper.

2.2 Grading Model

In the manual evaluation, the grading scheme allows the maximum mark $a > 0$ and the minimum mark is 0 for each submission. The problem resembles the regression problem in supervised machine learning since domain experts assign marks in the interval $[0, a]$. The grade given by the domain expert based on the grading criteria is referred to as the *ground truth.*

We found a nonlinear relationship between *ground truth* and *similarity score.* Therefore, the similarity score and the method of implementation are the key features for building the model and grading the submissions. The feature *similarity score* is created during the approach matching step from the abstract representation of the program. Relationship between the *ground truth* and the *similarity score* is defined by Eq. 1.

$$M_i = f(S_i) \tag{1}$$

where M_i is the grade returned by the model and S_i is the highest similarity score for the i^{th} incorrect solution and correct solutions. We have used regression model, which is a supervised machine learning technique to learn f. The supervised machine learning models *Random Forest Regression* and *Support Vector Regression* with three distinct kernels *RBF, Poly* and *linear* were investigated. The models were trained and validated using a variety of data sets which included set of problems from different programming language. We use 10% of the data as test data and the rest 90% of the data to train the models in each training phase to verify the model's generalisation capacity. 4-fold cross-validation is used to train all of the models. The evaluation metrics *Explanatory Variance (EV)* and R^2-*Score* are used to compare the models.

3 Related Work

This section discusses the prior art in various automated evaluation methods available for evaluating programming assignments and position our work w.r.t. them.

A majority of the current automated evaluation tools used today are mainly test-case based [1,4,9]. In most of these, the evaluator generates a set of test cases either manually or automatically to evaluate the programs. Each test case carries a particular weight of marks and calculates the number of test cases passed and assigns grades [5,6,13]. Another category of automated grading methods considers the semantic similarities among the programs based on abstract representation of the program like *abstract syntax tree* (AST), *control flow graph* (CFG), *data flow graph* (DFG)) and *program dependence graph* (PDG) etc. [2,7,12]. In addition, we have recently seen the evolution of automated grading using machine learning [8,10].

The work presented in our paper builds upon several of the above pieces of work. We use testing to segregate the correct solutions from the partially correct/incorrect ones. We apply feature based similarity estimation in our work both at the similarity estimation stage and marking stage similar to [10]. However, this is done as a step in a larger process. Further, we estimate structural similarity w.r.t. multiple possible reference solutions, rather than a unique one provided by the instructor. Unlike in all pure testing or pure static analysis based approaches, we use a combination of testing, static analysis and machine learning to award the final marks to a solution. This also distinguishes our contribution w.r.t. all evaluation approaches surveyed above.

4 Conclusion

In this work, we have addressed the difficulty in grading the incorrect programming problems and introduced a system for grading the same. Our method LetGrade are capable to grade the incorrect programs like a human grader. The average variance in the grade predicted by the supervised machine learning model is consistently close to 0.5. This indicates that the models can accurately predict the grade within 10% margin of error. This method simplifies the time-consuming work of the instructors in the programming course. We would like to extend the system to other programming languages.

References

1. Douce, C., Livingstone, D., Orwell, J.: Automatic test-based assessment of programming: a review. J. Educ. Res. Comput. (JERIC) **5**(3), 4-es (2005)
2. Goswami, N., Baths, V., Bandyopadhyay, S.: AES: automated evaluation systems for computer programing course. In: Proceedings of the 14th International Conference on Software Technologies, pp. 508–513 (2019)
3. HackerRank: Hackerrank (April). http://hackerrank.com
4. Jackson, D.: A software system for grading student computer programs. Comput. Educ. **27**(3–4), 171–180 (1996)
5. Joy, M., Griffiths, N., Boyatt, R.: The boss online submission and assessment system. J. Educ. Resour. Comput. (JERIC) **5**(3), 2-es (2005)
6. von Matt, U.: Kassandra: the automatic grading system **22**(1) (1994). https://doi.org/10.1145/182107.182101
7. Naudé, K.A., Greyling, J.H., Vogts, D.: Marking student programs using graph similarity. Comput. Educ. **54**(2), 545–561 (2010)
8. Singh, G., Srikant, S., Aggarwal, V.: Question independent grading using machine learning: the case of computer program grading. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 263–272 (2016)
9. Singh, R., Gulwani, S., Solar-Lezama, A.: Automated feedback generation for introductory programming assignments. In: Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 15–26 (2013)
10. Srikant, S., Aggarwal, V.: A system to grade computer programming skills using machine learning. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1887–1896 (2014)
11. Verma, A., Udhayanan, P., Shankar, R.M., KN, N., Chakrabarti, S.K.: Source-code similarity measurement: syntax tree fingerprinting for automated evaluation. In: The First International Conference on AI-ML-Systems, pp. 1–7 (2021)
12. Wang, T., Su, X., Wang, Y., Ma, P.: Semantic similarity-based grading of student programs. Inf. Software Technol. **49**(2), 99–107 (2007)
13. Wick, M., Stevenson, D., Wagner, P.: Using testing and Junit across the curriculum. ACM SIGCSE Bull. **37**(1), 236–240 (2005)