Use of Machine Learning to Identify Predictors of Student Performance in Writing Viable Computer Programs with Repetition Loops and Methods

Candido Cabo, Ph.D.
Department of Computer Systems
New York City College of Technology
City University of New York
ccabo@citytech.cuny.edu

Abstract –The goal of this research to practice full paper is to identify which computer programming concepts/skills predict students' ability to write viable programs using repetition loops and custom methods in Java. We developed machine learning models (logistic regression and decision trees with *Scikit-Learn*) to predict student performance in writing computer programs. High scores in feature importance analysis of the concepts/skills used as inputs to the models were considered important for predicting the output (i.e., performance in writing viable programs using loops and methods). We found that: 1) The ability to write programs with repetition and methods relies on an adequate understanding of several previous pre-requisite concepts/skills; 2) The relative importance of the pre-requisite concepts/skills varies, but adequate understanding of selection structures, which is typically taught early in the semester, is critical for students to be able to write viable computer programs using repetition loops and methods later in the semester; 3) Machine learning models can be used as predictors of student ability to write viable computer programs; 4) The transparency and interpretability of white-box models, like logistic regression and decision trees, allows students and teachers to identify which pre-requisite concepts/skills need to be emphasized and reinforced to increase performance on a target concept/skill.

Keywords –Computer Programming Teaching and Learning, Java, Student Performance, Machine Learning, Learning Analytics, Educational Data Mining

I. INTRODUCTION

According to Bruner [1], "a theory of instruction must specify the ways in which a body of knowledge should be structured so that it can be most readily grasped by the learner." Moreover, "a theory of instruction should specify the most effective sequences in which to present the materials to be learned." Writing viable computer programs requires an understanding of several interrelated concepts and the ability to apply those concepts to solve problems [2]. Those concepts/skills should be presented to the student in an effective scaffolded sequence, so students can make adequate progress in their ability to write computer programs.

Concepts like assignment, data types, structures that control the flow of execution in computer programs like sequencing, selection (if/else), repetition (for/while), methods, and data structures like arrays are basic to computer programming, and are part of every course in computer programming. Computer programming curricula generally follow Bruner [1] theory of instruction: simpler concepts are presented first (assignment, data types), and complex concepts (selection, repetition, methods), which build on the simpler ones presented earlier, are introduced later. Still, writing viable computer programs using repetition loops and custom methods is a challenge for novice programmers.

We hypothesize that to structure the learning material into an effective sequence, we need to develop more insight on the interrelatedness between the different concepts/skills presented in computer programming courses. We believe that quantifying how the understanding of concepts/skills taught early in the semester influence (or do not influence) the ability to learn concepts/skills taught later in the semester would facilitate learning. The goal of this paper is to identify, using machine learning predictive models, which computer programming concepts/skills determine the ability of students to write viable programs using repetition and methods in Java. The models can also be used to predict, based on performance in concepts/skills already learned, student performance in yet to be learned concepts/skills. Those predictions allow students and teachers to identify areas of concern that need to be emphasized and reinforced to facilitate learning future target concepts/skills.

II. BACKGROUND AND PRIOR WORK

In addition to the structure and sequence of presentation of the learning materials, Bruner recognized that other factors, like

student "predispositions" and the "form and the pacing of reinforcements," could also affect learning [1]. Indeed, a review of the literature indicates that the challenges faced by students when learning computer programming are multifactorial, and relate not only to the difficulty of the curriculum but also to pedagogy and differences in student learning styles and attitudes [2][3]. All those challenges result in high drop out and failure rates in computer programming courses [2][3][4][5].

One approach to improve student success in programming courses is to find early predictors that could identify students at risk. The idea is that by predicting student performance early in the semester, there is still time for interventions to improve students' outcomes. A review of the literature on early predictors of success/failure in learning programming can be found in [4]. Despite the broad range of factors investigated, including performance in mathematics, performance in specialized tests, demographics, cognitive capacity, style and development, student attitude and motivation, no clear predictor has emerged, and the question remains unanswered.

Machine learning predictive models have been used extensively to identify students at risk early in the semester [6]. The approach in the majority of those studies is to use a combination of demographic, academic and assessment data collected early in the semester to predict student performance at the end of the semester. Those studies use different features as input to the models: Ahadi et al. [7] used gender, age, GPA, previous programming experience and programming behavior collected in the first week of the semester; Khan et al. [8] used gender, GPA, academic semester, attendance and the first 15% of the semester assessments; Bergin et al. [9] used the results of earlier mathematics examinations, the number of hours spent playing video games while taking the course, and a measurement of programming self-esteem. A comparison of the prediction ability of different machine learning algorithms is often included as part of the studies. For example, Ahadi et al. [7] and Khan et al. [8] found that decision trees is the best predictor of student performance at the end of the semester, while Bergin et al. [9] found Naïve Bayes models to make more accurate predictions.

Instead of aiming at predicting end-of-semester student performance, other studies aim at modeling and predicting performance in the next-concept to be learned based on performance on current concepts, an extension of what Corbett and Anderson [10] called "knowledge tracing". Piech et al [11], using a *Khan Academy* dataset, found that recurrent neural networks can predict whether or not a student will solve a particular problem correctly given their performance on prior problems. Wang et al. [12] modeled students' current knowledge state to make predictions about the acquisition of new concepts using recurrent neural networks trained in the *Hour of Code* dataset (*Code.org*). The model was able to learn a suitable representation of students' current knowledge without human intervention.

The idea of "knowledge tracing" assumes an interrelatedness of the concepts and skills taught in computer programming courses. Such an interrelatedness has been investigated in several reports. Cabo [13] used factor analysis to uncover interdependencies between computer programming concepts, and investigated how conceptual understanding relates to the ability to write viable programs. Cabo [14] found that performance in skills taught early in the course determine performance in skills taught later. Bosse et al. [15] found connections between topics in a computer programming course from interviews with instructors and students' diaries. Kaur et al. [16] used association analysis between topics to improve the order of content delivery in a computer programming course.

III. Research Questions

The overall goal of this study is to identify which computer programming concepts/skills are important pre-requisites to write viable programs using repetition loops and methods. The specific research questions are:

RQ1: Can we predict student performance in writing programs using repetition loops and methods based on their performance on earlier concepts/skills?

RQ2: Which are the pre-requisite concepts/skills needed to write viable programs that use repetition loops?

RQ3: Which are the pre-requisite concepts/skills needed to write viable programs that use custom methods?

IV. Methods

4.1. Participants and setting

Our institution is one of the most diverse institutions of higher education in the northeast United States: 29% of our students are African American, 34% are Latino, 20% are Asian or Pacific Islanders, and 10% are Caucasian. The College's fall 2019 enrollment was 17,036. Students enrolled in eight sections of a Java Fundamentals course between Fall 2014 and Spring 2019 were part of the study (n = 145). Students who officially dropped the class or stopped attending the class were excluded from the analysis.

4.2. Student performance in Java concepts/skills

We assessed students' understanding of the following programming concepts: assignment, selection (if/else), repetition (for/while), arrays and methods (first two categories in Bloom's taxonomy [17][18]). We also assessed students' ability to apply those concepts (third category in Bloom's taxonomy) to write viable computer programs using selection, repetition, arrays and methods. Assessments were graded on a 0-10 scale. Students who obtained a grade of >= 7 (equivalent to 70% or a C) in the average of problems in a given category were considered proficient in that category.

4.3. Machine learning prediction models

We used two machine learning predictive models, logistic regression and decision trees (*Scikit-Learn* [19]), to identify which computer programming concepts/skills are important pre-requisites to write programs using repetition and methods. We also evaluated the ability of those models to predict student performance in writing computer programs with repetition and methods.

Logistic regression (*sklearn.linear_model.LogisticRegression*) is a machine learning classification algorithm that is used to predict the probability of a categorical dependent variable (target) based on the values of independent variables (features). In this report, we constructed models that predict the probability of writing viable programs involving repetition and methods based on a number of independent variables that reflect performance in concepts and skills taught earlier in the semester. The features represent the results of student performance in the continuous range 0-10. The target variable is binary: 1 (able to write viable programs) or 0 (unable to write viable computer programs).

Decision trees are machine learning classification algorithms that predict the probability that a target variable belongs to a certain class based on True/False decisions on the values of feature variables made at each non-leaf node of the tree. The algorithm finishes when the decision flow reaches a leaf node. The decision tree algorithm used by Scikit-Learn (*sklearn.tree.DecisionTreeClassifier*) is the Classification and Regression Tree (CART) algorithm, which produces only binary trees. Based on student performance on early concepts/skills, a decision tree predicts whether a student will be able or not able to write viable programs. In constructing our model, instead of using a single decision tree, we used a bagging ensemble (*sklearn.ensemble.BaggingClassifier* in *Scikit-Learn*) of 100 base decision trees, where each tree uses a random sampling with replacement (bootstrap) of the training set. Bagging combines the predictions of the different decision trees to produce a final prediction by using the most frequent prediction of all base classifiers. We used cross validation folds of size 5 to determine the optimal maximum depth of the tree (which was 3 levels) to prevent overfitting.

For both models, the total number of samples was divided in a training and a testing set. The training set consisted of 70% randomly selected samples of the total set and it was used for model training and validation. The remaining 30% of the samples were used for testing the models.

4.4. Feature and target variables used in the models

To analyze and predict performance in writing programs using repetition loops, the features used by the models were the performance (range: 0-10) in earlier concepts (assignment, selection and repetition) and skills (selection). The target variable (repetition skills) was binary: 1 (ability to write viable programs using repetition) or 0 (inability to write viable computer programs using repetition).

To analyze and predict performance in writing programs using methods, the features used by the models were the performance (range: 0-10) in earlier concepts (assignment, selection, repetition, arrays and methods) and skills (selection, repetition and arrays). The target variable (method skills) was binary: 1 (ability to write viable programs using methods) or 0 (inability to write viable computer programs using methods).

4.5. Feature importance analysis

We used feature importance analysis to determine which concept/skill features were more important for predicting the output (i.e., performance in writing viable programs using loops and methods).

For logistic regression models, we used the value of the estimated coefficients for each feature as an estimate of its importance. Additionally, we also used Recursive Feature Elimination (*sklearn.feature_selection.RFE*), which ranks the importance of the features used as input to the model. Both methods produced the same ranking of feature importance (see below). For the decision tree models, we used the *feature_importances_* property of the model that is directly accessible in *Scikit-Learn*. The values of the feature importance are given as a percentage and add up to 100%.

4.6. Predictive performance analysis

To quantify the predictive ability of the models, we used standard measurements of accuracy, precision, recall, and AUC (area under the ROC curve), all applied to the test data [19].

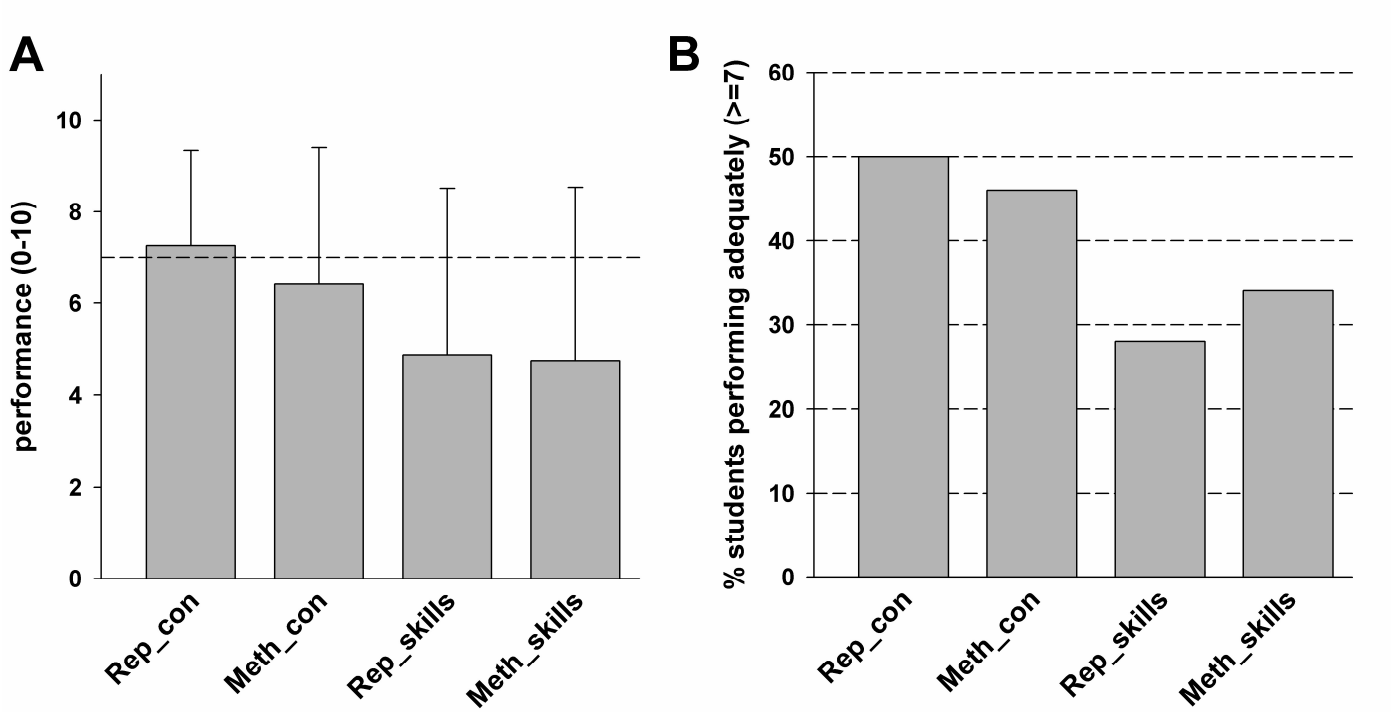

Figure 1. (A) Average performance in repetition and methods concepts (Rep_con, Meth_con) and skills (Rep_skills, Meth_skills) (range = 0-10). **(B)** Percentage of students' performing adequately (>= 7) in repetition and methods concepts and skills.

V. RESULTS

5.1. Student performance in repetition loops and methods

Figure 1A shows the average performance of students in concepts and skills related to the use repetition loops and custom methods in Java programs. The average performance is 4.86+/-3.64 in programs using repetition and 4.75+/-3.78 in programs using methods. About 28% of students are able to write viable programs that include repetition loops (Rep_skills, Figure 1B) and ~34% are able to use methods correctly (Meth_skills, Figure 1B). All in all, these results show that writing programs using repetition and custom methods is a challenge for our students.

5.2. Analysis of student performance writing programs with repetition loops

Figure 2 shows the results of the feature importance analysis when repetition skills was the target. When using logistic regression models (Figure 2A), the two most important features, using the regression coefficients (grey bars) and the ranking from Recursive Feature Elimination (the numbers inside the bars), are the performance in *selection concepts* (Sel_con) and *repetition concepts* (Rep_con). When using the decision tree bagging ensemble model (Figure 2B), the two most important features are *selection concepts* (Sel_con) and *selection skills* (Sel_skills). Both models agree that performance in *selection concepts* is the most important feature, but each model selects a different feature as second in importance. Note that feature importance values add up to 1 (100%) in Figure 2B, but not in Figure 2A, because the estimation of feature importance is based on the value of the regression coefficients.

Figure 3 shows an example of a decision tree to predict student performance in repetition loops. The decision process starts at the root (top) node. After evaluating the condition based on one value of one of the features (Selection_skills <= 8.5), it proceeds to the next layer depending on whether the condition is true (arrow pointing down-left) or false (arrow pointing down-right). The decision process continues until a leaf node (a node with no children nodes, no outgoing arrows) is reached. In addition to a condition, each node has four attributes: a *gini* value, *samples*, *value* and *class*. The *class* attribute in the leaf nodes indicates the prediction of the model: class 0 (not being able to write programs with repetition loops); class 1 (able to right programs with repetition loops). A node *samples* attribute represents how many training instances that node applies to. For example, for the most bottom-right node (level 3, right node), *samples* = 15 means that 15 training samples are in that node. The attribute *value* = [4, 11] means that 4 training

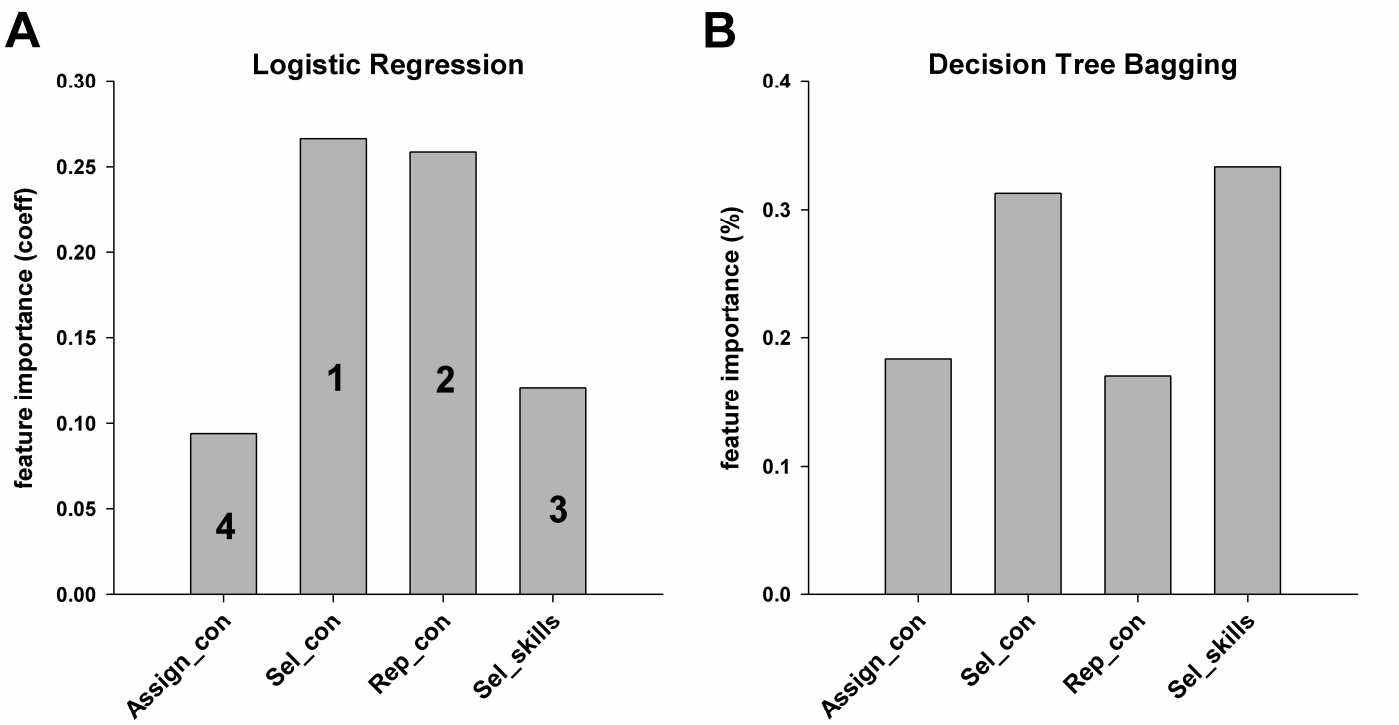

Figure 2. Feature importance analysis when predicting repetition skills. **(A)** Grey bars represent the values of the coefficients in a logistic regression model. The numbers in the bars represent feature importance ranking using Recursive Feature Elimination. **(B)** Feature importance expressed as a percentage when using a decision tree model.

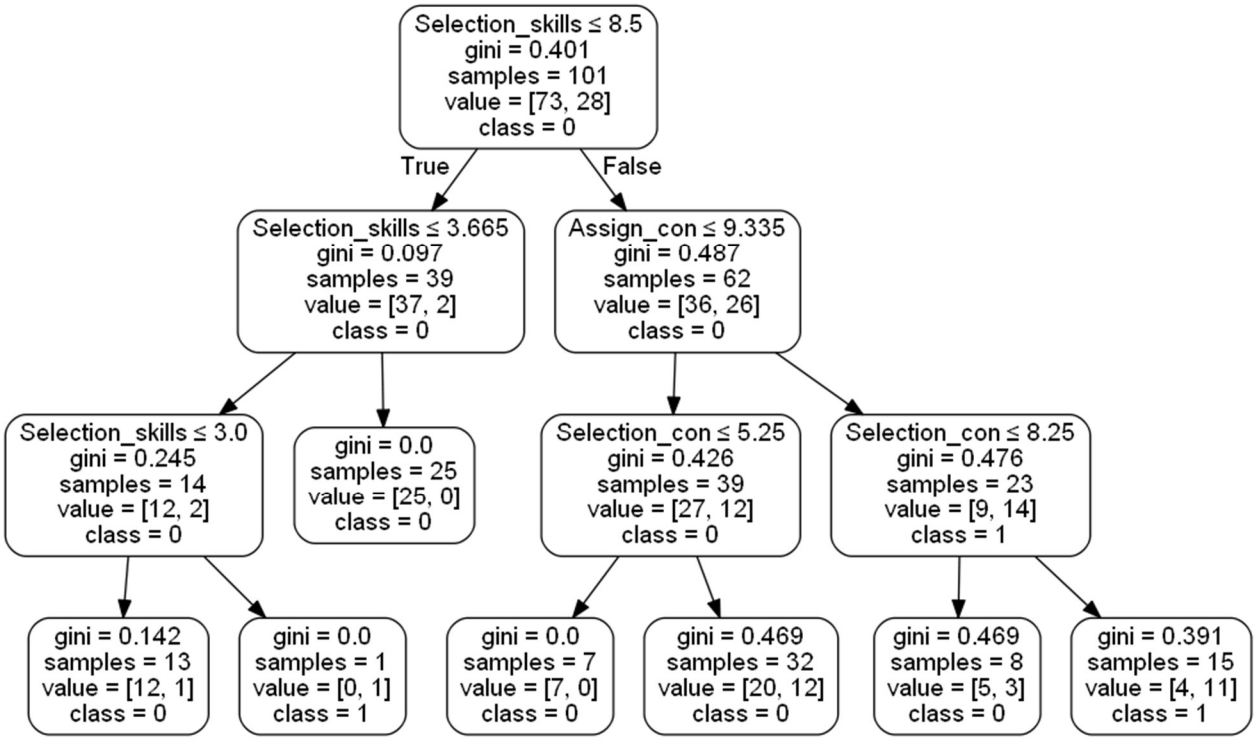

Figure 3. Decision tree to predict adequate (class = 1) or inadequate (class = 0) student performance in repetition skills. See text for explanation.

instances belong to class 0 and 11 training instances belong to class 1. The predicted class for the node is *class* = 1 because there are more training instances belonging to class 1 in that node. Note, however, that there is a probability associated with the decision. The probability that an instance that ended in that node belongs to class 1 is 73% (= 11/15) and that it belongs to class 0 is 27% (= 4/15). The *gini* = 0.391 (= $1 - (11/15)^2 - (4/15)^2$) attribute measures the "impurity" of the node (if the node contains training instances belonging to more than one class, the node is "impure" and the *gini* attribute is different from 0). The tree in Figure 3 also shows which features are relevant to the model: *selection skills*, *assignment concepts* and *selection concepts*, which is consistent with the three most important features in Figure 2B. The tree in Figure 3, which is just one of the 100 trees in the ensemble, does not use *repetition concepts* as one of the features in the decision tree.

5.3. Analysis of student performance writing programs with custom methods

Figure 4 shows the results of the feature importance analysis when method skills was the target. When using logistic regression models, the two most important features, using the regression coefficients (grey bars) and the ranking from Recursive Feature Elimination (the numbers inside the bars), are the performance in *method concepts* and *repetition skills* (Figure 4A). The same two features are selected as the most important by the decision tree bagging ensemble model (Figure 4B). The models differ in the selection of the third most important feature: *selection concepts* (logistic regression) and *array skills* (decision trees). As in Figure 2B, note that feature importance values add up to 1 (100%) in Figure 4B, but not in Figure 4A, because the estimation of feature importance is based on the value of the regression coefficients.

Figure 5 shows an example of a decision tree to predict student performance in writing programs that include custom methods. The notation and interpretation of the tree is similar to Figure 3. The tree in Figure 5 shows which features are relevant to the model: *methods concepts*, *repetition skills*, *array skills* and *array concepts*. Note that the three most important features are consistent with the results in Figure 4B, but *array concepts* are not the fourth most important feature. This is due to the fact that the results in Figure 4B represent the feature importance of the bagging ensemble of 100 trees, and the tree in Figure 5 is just one of the trees in the ensemble. Also note that performance in *array concepts* (level 2, third node from the left) is used to discriminate the class of just 3% (= 3/101) of the training samples.

5.4. Prediction of student performance writing programs with repetition loops and with methods

Table 1 shows the predictive performance for the different models and targets. When *repetition skills* is the target, both prediction models, logistic regression and decision trees, perform about the same. When *method skills* is the target, decision trees seems to perform slightly better. The prediction performance for *method skills* is higher than the prediction performance for *repetition skills*, regardless of the model.

VI. DISCUSSION

6.1. Next-Concept/Skill performance prediction (RQ1)

Next-concept/skill predictions estimate performance in future concepts/skills from performance on concepts/skills that had already been taught earlier in the semester. Those predictions provide students and instructors with an estimation of student readiness to learn future concepts/skills and make adequate progress in the course. If the prediction suggests that a student is not ready, the predictive models provide guidance on which concepts/skills need to be revisited and reinforced.

In this paper, we show that, by using machine learning algorithms, it is possible to predict student performance in writing viable computer programs with repetition loops and custom methods from their knowledge of concepts taught earlier in the semester (Table 1). The performance of the two machine learning algorithms used in this study, logistic regression and decision trees, is similar when predicting performance in *repetition skills*. However, when predicting *methods skills*, decision tree models perform better than logistic regression models. Other studies also found that decision trees perform better than other algorithms when predicting end-of-semester student performance early in the semester [7][8].

The predictive performance is higher for *method skills* models than for the *repetition skills* models (Table 1). This is not surprising because the *method skills* models use eight input features (Figure 4), and *repetition skills* models use just four (Figure 2). In predicting the final outcome of a programming exercise based on successive submissions of the exercise, Wang et al. [12] also found that the predictions improve with the number of submissions. The more information we have about the current knowledge that a student has, the easier it is to predict performance in the next-concept/skill.

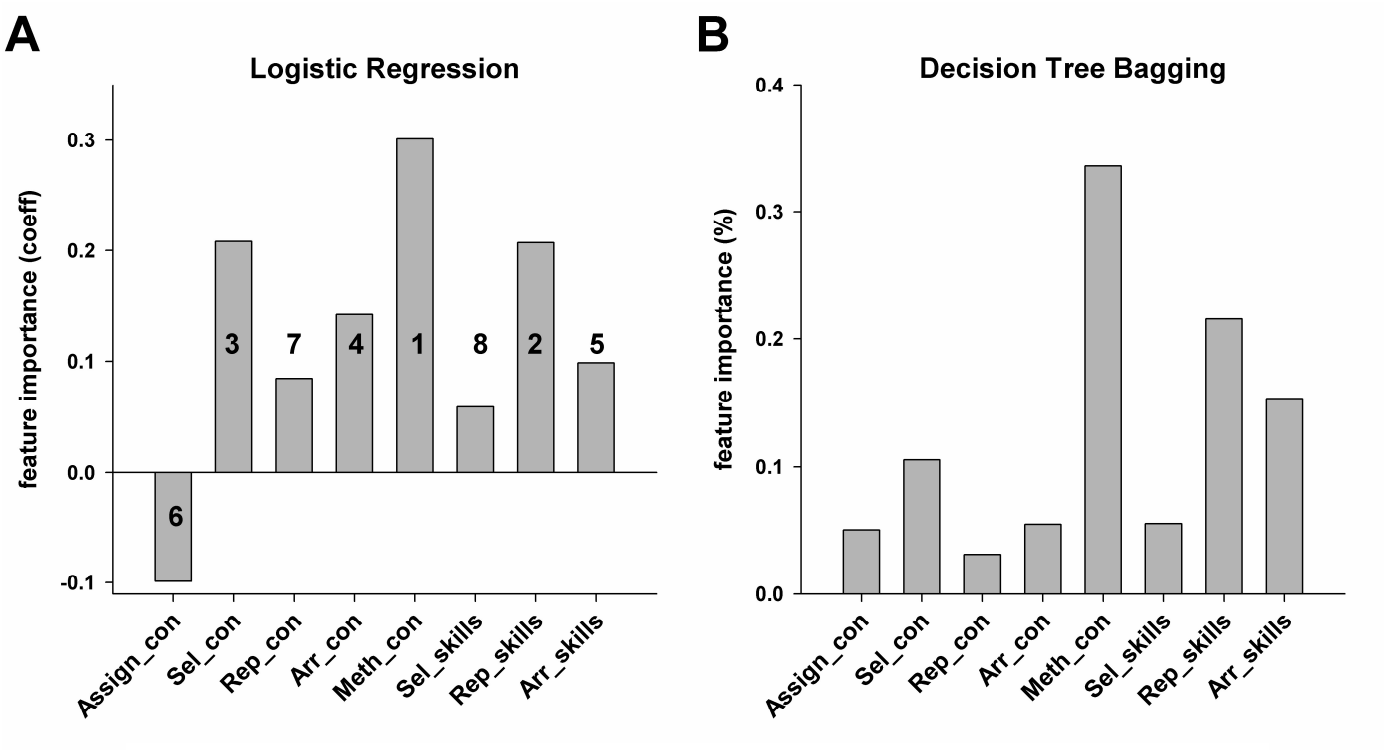

Figure 4. Feature importance analysis when predicting method skills. **(A)** Grey bars represent the values of the coefficients in a logistic regression model. The numbers in the bars represent the importance ranking using Recursive Feature Elimination. **(B)** Feature importance expressed as a percentage when using a decision tree model.

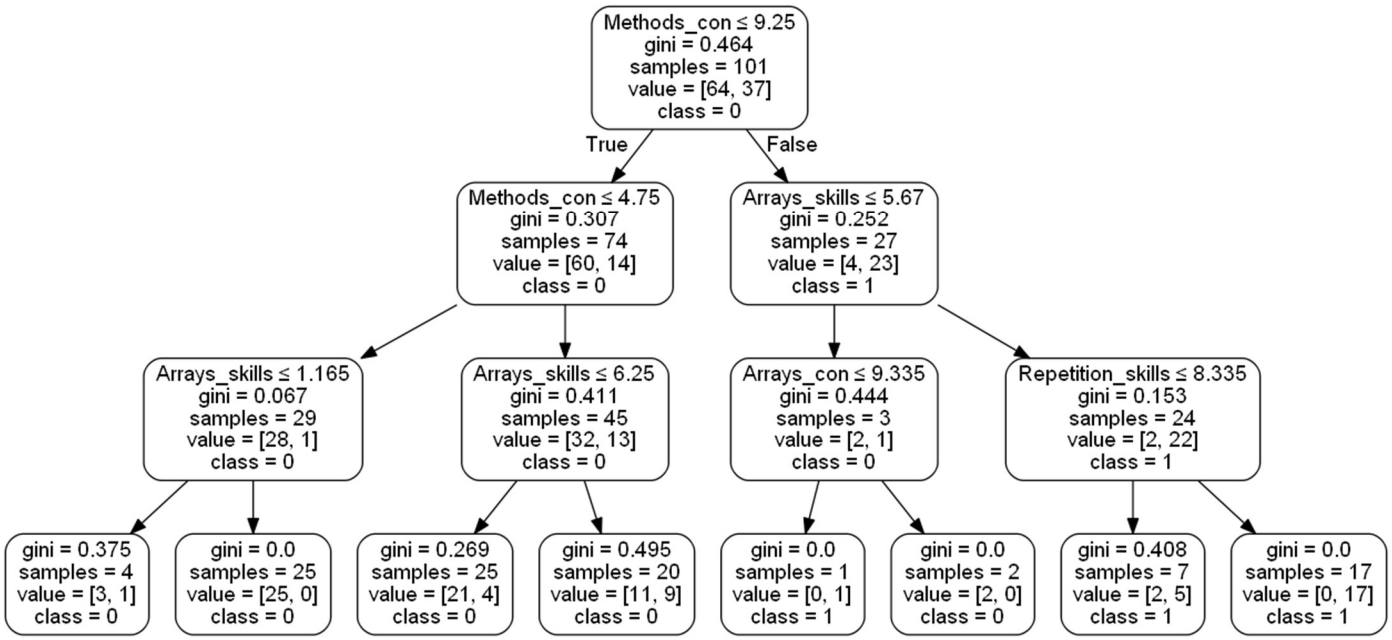

Figure 5. Decision tree to predict adequate (class = 1) or inadequate (class = 0) student performance in method skills. See text for explanation.

Table 1. Predictive Performance of the Models						
Model	**Target**	**Accuracy**	**Precision**	**Recall**	**F1-score**	**AUC**
Logistic Regression	Repetition skills	0.75	0.73	0.75	0.74	0.79
Decision Tree	Repetition skills	0.75	0.73	0.75	0.74	0.77
Logistic Regression	Method skills	0.83	0.82	0.83	0.82	0.81
Decision Tree	Method skills	0.84	0.85	0.84	0.85	0.90

6.2. Pre-requisite concepts to write Java programs with repetition loops and methods (RQ2, RQ3)

We believe that in an educational context, understanding how models make predictions is as important as the accuracy of those predictions. The interpretability and transparency of logistic regression and decision tree models makes it possible to analyze the relative importance of the input features in predicting a target. If a feature is important in predicting a target, then we consider that the concept/skill represented by the feature is a pre-requisite to the concept/skill represented by the target. Identifying important features provides students and instructors with guidance on which pre-requisite concepts/skills need to be improved to increase performance in a target concept/skill.

Different algorithms may lead to the identification of different pre-requisite concepts/skills needed for a target concept/skill because different algorithms make different assumptions about the data. For example, when using logistic regression to predict performance in *repetition skills*, the two most important features identified by the model are *selection concepts* (Sel_con) and *repetition concepts* (Rep_con) (Figure 2A). However, when using decision trees, *selection concepts* (Sel_con) and *selection skills* (Sel_skills) are the two most important concepts (Figure 2B). Despite their differences, both models agree that performance in *selection concepts* is a major determinant of performance in *repetition skills*. That is not surprising given that learning *selection concepts* includes learning boolean expressions, which are necessary to create repetition loops [15]. What is more surprising is that, for the decision tree model, performance in *repetition concepts* (Rep_con) is the least important feature for predicting *repetition skills* (even thought it contributes about 20%, Figure 2B), which may suggest a misalignment in the way we teach the theory and practice of repetition loops. However, as noted above, *repetition concepts* were ranked as second in importance when using the regression model (Figure 2A).

The most important feature to predict *method skills* is performance in *method concepts* (Meth_con) for both the logistic regression and decision trees models (Figure 4), which suggests a good alignment in the way we teach the theory and practice of methods. Teaching *method concepts* should address the major challenges to define and use methods effectively: 1) understanding the flow of execution from a calling method to the called method and back; 2) managing the passing of arguments to the methods and the return of a value from a method. Both models also agree that the second most important feature in predicting method skills is *repetition skills* (Rep_skills) (Figure 4). While it is not obvious why *repetition skills* should be a pre-requisite to *method skills*, *repetition skills* reflect an ability to apply computer concepts to writing viable computer programs. That transfer ability could also facilitate the application of *method concepts* to writing computer programs with methods (i.e., to develop *method skills*).

As we mentioned earlier, writing viable computer programs requires an understanding of several interrelated concepts and the ability to apply those concepts to solve problems [2]. Moreover, adequate performance in all (not just a few) conceptual categories is important to be able to write viable computer programs [13]. So, while it is helpful to rank the importance of the different input features, Figures 2 and 4 show that all features, to a greater or lesser extent, contribute to the prediction of, and are important pre-requisites for a given target skill.

VII. Limitations

There are elements that are important to write viable computer programs, like the ability to understand word problems [20] or learning attitudes like persistence and attention to detail [1], that have not been incorporated in the models. Further studies are necessary to determine if our results hold in a different context [21], and apply to other student populations.

VIII. Conclusions

1) Logistic regression and decision tree models can be used to predict student performance in writing viable computer programs with repetition loops and custom methods; 2) The transparency of white-box models, like logistic regression and decision trees, allows students and teachers to identify which pre-requisite concepts/skills need to be improved to increase performance in a target concept/skill; 3) *Selection concepts* is the most important pre-requisite for writing programs that use repetition loops (*repetition skills*); 4) *Method concepts* and *repetition skills* are the most important pre-requisites for writing programs that use custom methods; 5) Learning a new concept or skill builds on an adequate understanding of not just one but several previous pre-requisite concepts and skills.

REFERENCES

[1] Bruner, J.S. *Toward a Theory of Instruction*. Harvard University Press, Cambridge, MA, 1966.

[2] Robins, A. (2010). Learning edge momentum: a new account of outcomes in CS1. Computer Science Education, 20(1), 37–71.

[3] Gomes, A., Mendes, A.J. (2007) Learning to program – difficulties and solutions. International Conference on Engineering Education (ICEE 2007), September 3-7, 2007, Coimbra, Portugal.

[4] Robins A, Rountree J, and Rountree N. (2003). Learning and Teaching Programming: A Review and Discussion. Computer Science Education 13: 137-172.

[5] Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M. & Paterson, J. (2007). A survey of literature on the teaching of introductory programming. SIGCSE Bulletin 39 (4), 204–223.

[6] Rastrollo-Guerrero, J.L.; Gómez-Pulido, J.A.; Durán-Domínguez, A. Analyzing and Predicting Students' Performance by Means of Machine Learning: A Review. *Appl. Sci.* 2020, *10*, 1042. doi: 10.3390/app10031042.

[7] Ahadi A, Lister R, Happala H, Vihavainen A, "Exploring Machine Learning Methods to Automatically Identify Students in Need of Assistance." Proceedings of the 2015 ACM Conference on International Computing Education Research (ICER), August 9 – 13, 2015, Omaha, Nebraska, USA. doi: 10.1145/2787622.2787717.

[8] Khan, A. Al Sadiri, A. R. Ahmad and N. Jabeur, "Tracking Student Performance in Introductory Programming by Means of Machine Learning," *2019 4th MEC International Conference on Big Data and Smart City (ICBDSC)*, Muscat, Oman, 2019, pp. 1-6, doi: 10.1109/ICBDSC.2019.8645608.

[9] Bergin, S., Mooney, A., Ghent, J., and Quille, K. Using Machine Learning Techniques to Predict Introductory Programming Performance. International Journal of Computer Science and Software Engineering (IJCSSE) 4: 323-328, 2015.

[10] Corbett, A.T., Anderson, J.R. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Model User-Adap Inter* 4, 253–278 (1994). doi: 10.1007/BF01099821.

[11] C. Piech, J. Bassen, J. Huang, S. Ganguli, M. Sahami, L. J. Guibas, and J. Sohl-Dickstein. Deep knowledge tracing. In Advances in Neural Information Processing Systems, pages 505–513, 2015.

[12] Wang, L.; Sy, A.; Liu, L.; Piech, C. Learning to Represent Student Knowledge on Programming Exercises Using Deep Learning*Proceedings of the 10th International Conference on Educational Data Mining,* Wuhan, China, Jun 25-28, 2017, pp. 324-329.

[13] Cabo, C., Quantifying Student Progress Through Bloom's Taxonomy Cognitive Categories in Computer Programming Courses *Proceedings of the 2015 ASEE Annual Conference & Exposition*, Seattle, Washington, June 2015. doi:10.18260/p.24632.

[14] Cabo, C. (2019) "Student Progress in Learning Computer Programming: Insights from Association Analysis," *2019 IEEE Frontiers in Education Conference (FIE)*, Covington, KY, USA, 2019, pp. 1-8, doi: 10.1109/FIE43999.2019.9028691.

[15] Bosse, Y., D. F. Redmiles and M. Gerosa, "Connections and Influences Among Topics of Learning How to Program," *2019 IEEE Frontiers in Education Conference (FIE)*, Covington, KY, USA, 2019, pp. 1-8, doi: 10.1109/FIE43999.2019.9028393.

[16] R. Kaur, T. Brown, G. Walia, M. Singh and M. Reddy, "Using Association Rule Mining Algorithm to Improve the Order of Content Delivery in CS1 Course," *2019 IEEE Frontiers in Education Conference (FIE)*, Covington, KY, USA, 2019, pp. 1-5, doi: 10.1109/FIE43999.2019.9028452.

[17] Bloom, Benjamin S., Max B. Englehart, Edward J. Furst, Walter H. Hill, and David R. Krathwohl. (1956) Taxonomy of Educational Objectives, the Classification of Educational Goals, Handbook I: Cognitive Domain, edited by Benjamin S. Bloom. New York: McKay.

[18] Krathwohl, David R. (2002). A Revision of Bloom's Taxonomy: An Overview. *Theory into Practice* 41, no. 4: 212-18.

[19] Geron, A. Hands-on Machine Learning with Scikit-Learn and TensorFlow. O'Reilly Media, Inc. Boston, MA. 2017.

[20] C. Cabo, "Fostering Problem Understanding as a Precursor to Problem-Solving in Computer Programming," *2019 IEEE Frontiers in Education Conference (FIE)*, Covington, KY, USA, 2019, pp. 1-9. doi:10.1109/FIE43999.2019.9028664.

[21] Fincher, S., Lister, R., Clear, T., Robins, A., Tenenberg, J. & Petre, M. (2005). Multi-institutional, multi-national studies in CSEd Research: some design considerations and trade-offs. Proceedings of the First international Workshop on Computing Education Research (ICER '05), 111–121.