Predicting at-risk Students to Facilitate Scaffolding Instructions

Qiong Cheng
University of North Carolina at Charlotte
qcheng1@uncc.edu

Mahitha Garikipati
University of North Carolina at Charlotte
mgarikip@uncc.edu

Smirthi Meenakshisundaram
University of North Carolina at Charlotte
smeenak1@uncc.edu

***Abstract*— This Innovative Practice Full paper presents our modeling methods to identify at-risk students early to scaffold for better intervention or interactions with students. We executed the data-driven practice in undergraduate and graduate Data Structures and Algorithms (DSA) courses for multiple semesters. Abstract nature in both courses can often be hard to teach and difficult for students to grasp. To resolve the difficulty and assess its efficacy, we propose predictive modeling methods to identify at-risk students, find student behavior patterns, and distinguish and target the areas that they need, to facilitate scaffolding instructions. We employed two feature selections and four predictive modeling methods, such as support vector machine (SVM), k-nearest neighbors (KNN), naive Bayes classifier (NBC), and random forest. Our comparative analysis on these predictive models indicates that random forest performed best in minimizing false negative (i.e., type II) error while not increasing false positive(i.e., type I) error significantly.**

Keywords—algorithms, machine learning, scaffolding instructions, adaptive learning, active learning

I. INTRODUCTION

Sequential, parallel, and distributed computing are the cores of computer science in the modern multi-core context, which involves courses that require programming and problem-solving skills in meaningful but different contexts. Both programming and problem solving are all about using logic to build upon the reusable, robust, and adaptable programming concepts. Data Structures and Algorithms (DSA_ is clearly one of such cornerstone computing courses, according to computer science curricula [1]. This course is a prerequisite for later advanced courses. Student experience in this course directly influences persistency/dropout rate.

From the instructor rational perspective, one of the learning outcomes of these undergraduate or graduate courses is designed to enable our students to develop transferable thinking as well as to promote student's problem-solving and programming skills. Through comparison and theoretical analysis, students get hands-on training on designing, implementing, or applying data structures and algorithms to solve problems more efficiently and effectively.

However, learning and teaching in this course are challenging. First, its abstract nature can often be difficult for students to grasp. Secondly, some students entering the undergraduate or graduate course are not well prepared, having problems in programming basics, and easily feeling frustrated or discouraged. Thirdly, contemporary students have grown up with massive distractions to entertainment electronics and social media, relatively low in motivation in general. Fourthly, widening student disparities is a current tendency.

Recently, some primary and secondary schools started to provide computing program(s). A number of high school students took AP computing courses and became relatively more experienced in coding [2-4]. However, there is still a large part of the "college-ready" students that have no participation in any computing-related activities at all before they arrive. It indicates a growing disparity in programming experience among students when entering programming courses.

Additionally, programming languages or requirements vary among schools or institutions. Knowledge background and prior experience in transfer students who enter classes more likely complicate the diversity, and thus, incoming students possess a wide range of prior knowledge and the confidence in learning programming, critical thinking, and problem-solving skills.

The increasing disparities lead to the alienation of the struggling students while boring those who are more experienced. It poses challenges to those core computing courses. At issue is the appropriateness of the one-size-fits-all pedagogical model. Thus, traditionally constructing knowledge via passive learning is not suggested.

An alternative is the adaptive learning model combined with scaffolding techniques. This adaptive learning model offers customized learning experiences to meet individual student needs, with a goal to helps all students to attain mastery of course content. In the undergraduate DSA course, we adopted adaptive preparation pedagogy for four semesters. In the graduate DSA course, we emphasized course projects adaptive to students' interests. Another common pedagogical practice is scaffolding with a combination of active learning techniques, such as algorithmic visualization simulation, collaborative study, problem-based learning, or project-based learning, and "flipped classroom".

In this paper, we study data-driven scaffolding. Our hypothesis is that identifying at-risk students and discovering behavior patterns would support scaffolding instruction in its design and execution. Namely, early detection of at-risk students gives us opportunities to contact them with personalized interaction and appropriate intervention. In our belief, this technique can be an effective approach for both undergraduate and graduate courses.

We investigated the data-driven technique in the DSA gateway courses and further compared and analyzed four predictive modeling methods, such as support vector machine (SVM) [5], k-nearest neighbors (KNN) [6], naive Bayes classifier (NBC) [7-8], and random forest [9], in identifying at-risk students with a goal of facilitating scaffolding instructions. We used two feature selection methods to reduce the number of variables respectively in each model. Our comparative analysis indicates that random forest performed best in minimizing false negative (i.e., type II) error while not increasing false positive(i.e., type I) error significantly.

The rest of the paper describes our methods and results (section 3) after a primer on our academic context (Section 2). We then discuss its evaluation in section 4. Section 5 draws a conclusion.

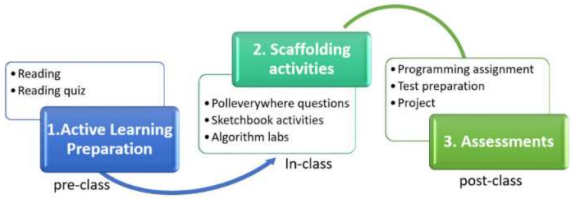

Figure 1 Modular learning cycle in an Algorithms and Data Structures course.

II. ACADEMIC CONTEXT

Both undergraduate and graduate Data Structures and Algorithms are core gateway courses that play a critical and intermediary role in computer science [1]. These courses are required for all students in our computing programs, and prerequisites to almost all of their senior-level courses. All undergraduate students who complete the course with an A, B, or C grade can move forward to their advanced courses.

Both courses are broad in topic coverage and sufficiently rigorous in computational logic and reasoning. It helps students to boost problem-solving and programming skills and prepares them to delve deeper into computing.

We organize these courses through a modular learning cycle:

Before class, students read book chapter(s) and work on preparation quiz questions. In face-to-face or remote classes, we scaffold instruction and lab activities and encourage teamwork. Through collaboratively solving a problem or answering questions in class, students are expected to practice and deepen their understanding by comparing and analyzing related concepts and constructing efficient data structures or algorithms. After class, students are required to independently work on programming assignment(s) or take a test to assess their understanding from low-order to high-order of Bloom's taxonomy.

The undergraduate course introduces fundamental data structure design and implementation as well as basic algorithmic techniques. It contains ten modules: generics and abstract data types (ADTs), algorithmic analysis, queues, stacks, lists, search and sorting, recursion, trees, hash tables/maps, and graphs. The graduate course focuses on algorithm analysis and diverse algorithmic design techniques as well as advanced data structures. It contains twelve modules: algorithm analysis, correctness analysis of algorithms, brute force and exhaustive search, divide and conquer methods, binary heap, transformation, reduction and input-enhancing methods in algorithm design, dynamic programming, greedy algorithms, advanced data structures, graph algorithms, linear programming, NP-completeness and approximations.

III. METHODS AND RESULTS

A. Scaffolding techniques

"Scaffolding" is a metaphor to capture the nature of support and guidance in learning [10-12]. It is a technique of breaking up the learning into chunks and then organizing an activity or a structure with each chunk. Instructional scaffolding allows instructors to guide and control learners' tasks and activities by assisting students step by step in order to complete a task or develop new understandings so that students can later complete similar tasks alone. In the core computing courses that require different levels of programming and problem solving, scaffolding involves a meta-programming approach to building software applications, which inspires student thinking of how the program could be developed practically.

Scaffolding is significantly important for enhancing student learning in these core computing courses since programming is not only about learning simple syntax constructs and their applications, but about honing practical problem-solving skills in meaningful contexts [13-17].

Existing scaffolding techniques have been used in different forms. From the perspective of teaching methodology, the scaffolding techniques help build up student confidence by overcoming difficulties through worked code demonstration or interactive auto-grading assessment tools with updated inputs or tasks. From the perspective of programming languages in general, there are in two folds: one form defines a small but essential subset of general-purpose programming languages, with which instructors can teach students basic programming concepts with wide applications while hiding complicated issues; the second form includes dragging-dropping based visual programming languages to reduce coding complexity [19]. From the perspective of reasoning, scaffolding problem solving includes two components: declaration- and procedure-based reasoning.

As mentioned in [20], student behaviors affect student satisfaction and instructor satisfaction. However, in the current literature, there are fewer studies investigating data-driven scaffolding instruction designs.

B. Data-driven Scaffolding

1) Participants and data sets

To examine the effectiveness of predictive models, we individually worked on undergraduate and graduate courses. In the graduate course, we had 324 consented students and 81 features over four semesters, Spring/Fall 2019, Spring 2020/2021; in the undergraduate course, we had 76 students and 31 features in Spring 2020 and 88 students in Fall 2020.

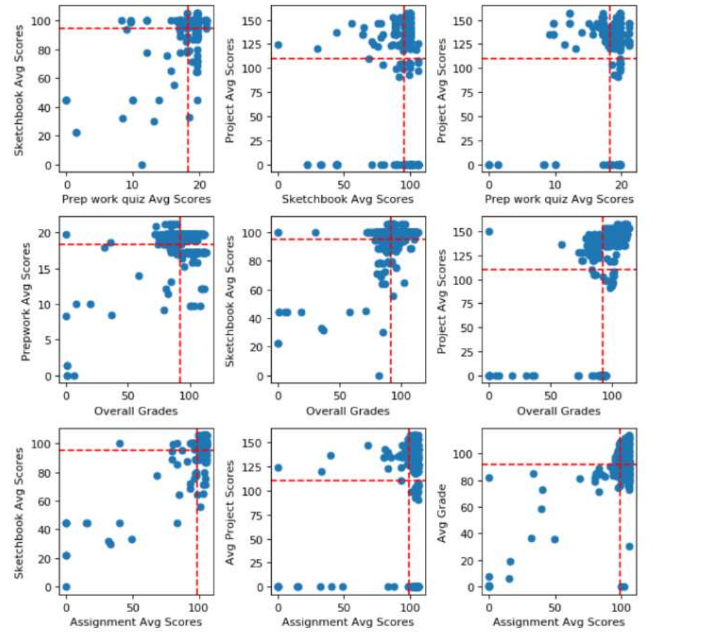

Figure 2 Student performance (graduate course over four semesters, Spring/Fall 2019, Spring 2020/2021). The dashed line is the average score overall participants. The first row corresponds to the scatter plots demonstrating the correlation among three assessment items, such as

sketchbook average scores, preparation work quiz average scores, and project average scores; the second row demonstrates the correlation of overall grades individually with these three assessment items; the third row corresponds to assignment average score individually with sketchbook average scores, average project scores, and average final grade.

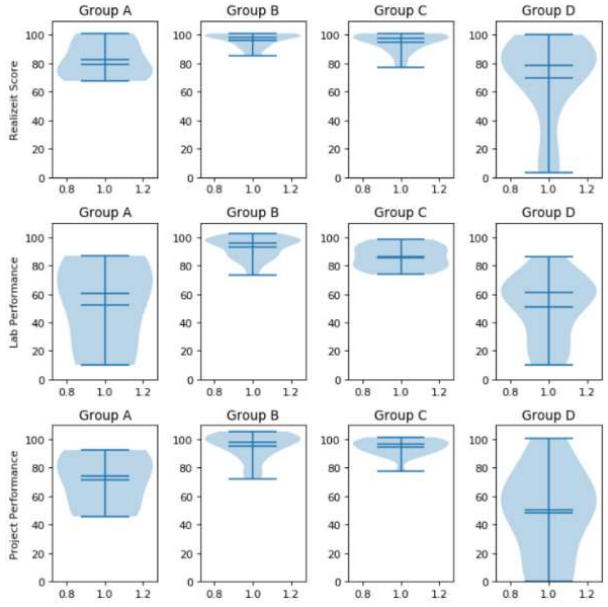

Figure 3 Performance patterns in undergraduate groups (Fall 2020). The first row is the distribution of each group in preparation work (We launched our preparation work in an integrated adaptive preparation platform, called Realize^{IT}), the second is student lab performance, and the third is student project performance. Students in group B were consistently excellent; students in group C worked hardest and progressed relatively most; students in group A dropped performance during the semester in this course; students in group D might be constantly problematic over semesters. Refer to detailed explanation in section 3.B.3.

2) Dependency of assessment items

The pedagogical design of a modular learning cycle drives our curiosity in the dependency of assessment items and their contributions to student overall grades (see Fig. 2). We first explored the pairwise correlation among three assessments involved in a modular learning cycle, such as preparation-work (also called prep-work) quizzes, sketchbook problems, and individual programming assignments. Then we studied the average effect of each assessment type on the overall grade. We hypothesize that a student who was well-prepared in each module would be well-motivated to work on sketchbook problems and then, most likely be successful in assignments and tests. Since programming assignments and projects hold about 40% of the final grade, student performance on these items correlated well with overall grades. In the exploration, we calculated the average score on each assessment type/component of interest for each student.

3) Student behavior patterns

Student overall performance may be determined by many other factors, for example, student motivation to learning, sense of belongingness and confidence, student learning styles, instructor's teaching styles, positive emotions, prior knowledge, time management, and instructional support. In this paper, we studied the relationship among student behavior patterns, prior knowledge, and final grades.

In the undergraduate course, our first pre-knowledge test occurred in the first week, and in the graduate course, it happened in the third week. The goal of this test is to evaluate students' prior knowledge in mathematics or programming. After comparing students' prior knowledge in the pre-test and their overall grades, we cast students into four behavior pattern groups. Group A had over-average prior knowledge but performed poorly; group B excelled in both prior knowledge and overall grade; group C had a poor prior knowledge but finally excelled in the class; group D failed in both (see Fig. 3).

Group B was consistently excellent and group D posited innate problems. Student performance in group A dropped down, which might indicate a change in these students, for example, a lifestyle change during the pandemic. Oppositely, students in Group C achieved the best progress on average. Their prep-work and in-class lab performance were worse than group B, but their performance ratio of programming assignments (also called mini-projects) over pre-knowledge test scores were highest, and learning progress was most pleasant. Note that programming assignments are independent work, but students are allowed to attempt for infinite times before the deadline, the auto-grading tool provides hints or timely feedback for each submission, and students have a fair opportunity to visit office hours to get help from TAs or instructors. All of these indicate that students in group C worked hardest and were motivated to perform for better.

As an instructor, we expect to promptly identify students in group A and D, so that necessary intervention can be executed to help these students and move them to B and C, respectively.

4) Predictive modeling methods to identify at-risk students

Computing courses at certain levels define as at-risk students differently. Our undergraduate data structures course requires a C at least, and a student is defined at-risk if the student would fail the course, namely his/her final grade is not greater than 70. Our graduate core course requires at least a B, and the threshold of defining an at-risk students has increased to 85.

We carried out predictive models individually on undergraduate and graduate courses as well. First, we collected consented data from canvas, cleaned, and normalized the data. Feature selection [21-23] is in one of preprocessing steps, effective in reducing the dimensionality. It removes irrelevant and redundant features and facilitates identifying the best set of features for a predictive model. Performing feature selection potentially increases the generalizability of the models and promotes the accuracy of the predictions.

In this study, we used two feature selection methods, Principal Component Analysis (PCA) [17] and correlation-based method [18], to reduce the number of variables used in each model.

PCA is originally an ordination technique to reduce feature dimensions to maximize the variance of selected features and minimize the distortion of distances between points upon projection into fewer dimensions by rigid rotation to derive successive orthogonal axes. Correlation help measure similarity or relevance between two features. There are two classical approaches to calculate the correlation coefficient, Pearson or Spearman rank-based correlation. The goal of the correlation-based feature selection is to find the most significant features which are less redundant.

To find the best predictive model, we have executed four different prediction methods to identify at-risk students:

K-Nearest Neighbor (KNN) is a non-parametric classifier. It classifies whether a student is at risk, by a majority vote of

its K neighbors. In this paper, we used two nearest neighbors. Naive Bayes Classifier (NBC) is a probabilistic classifier with an assumption of the (naive) independence between the predictive variables. It applies Bayes' theorem to calculate a conditional probability distribution of a student being at risk. Support Vector Machine (SVM) finds a hyperplane that separates at-risk students from no risky students and maximizes the soft margin of these two groups of students to the hyperplane. Random forests (RF), also called random decision forests, are an ensemble learning method. It constructs many decision trees that decide whether a list of student features indicates a fail at the training step. The final classifier is calculated as the mean/average prediction (regression) of the individual trees.

In each computational experiment, we preprocessed data (cleaning and normalizing data) and then executed a feature selection and predictive model, followed with a 4-fold cross-validation, where we calculated confusion matrix, true positions (TP), true negatives (TN), false positives (FP), and false negatives (FN).

To evaluate each model, we also measured accuracy, precision, sensitivity, specificity, F1.5 score, and ROC-AUC score. In these models, a negative result means the student is not at risk and will pass the course; a positive result means the student is at-risk and failed the course.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Sensitivity = \frac{TP}{TP + FN}$$

$$Specificity = \frac{TN}{FP + TN}$$

where:

TP is the number of students who failed and were identified as at-risk. TN is the number of students who passed and were not identified as at-risk. FN is the number of students who failed the course but were not identified by the models as at-risk. FP is the number of students who passed the course but were identified by the models as at-risk.

We built our predictive modeling methods, based on two feature selections, followed by four predictive models for the graduate course and three for the undergraduate course. Predicative results refer to Tables 1-4. Among these eight models in Tables 1 and 2, we observe that random forests outperformed. Among six models in Tables 3 and 4, we observe that PCA-KNN outperformed.

We used the best model to identify at-risk students at the early time of a semester, based on prior-knowledge assessment results. Over a semester, we compared and analyzed the performance and behavior patterns of predicated at-risk students and non-at-risk students individually on 31 assessment items. Viewing from the plot in Figure 4, we observed that predictive modeling distinguished behavior patterns between at-risk students and no-risk students.

Identifying at-risk students at an early time is facilitative for us to redesign intervention strategies. Based on students' motivation level and needs, we further scaffold instructions with an emphasis on problem-solving and reasoning of individual students.

5) Evaluation

We conducted a pre- and post-activity survey and measured student perceptions through 1) confidence and belongingness surveys respectively at the beginning and end of the semester, which is in the format of 5-point Likert scale questions and written comments, and 2) the reflections on learning and lessons learned.

As inspired by the learning theory of practice community [24-25], we believe that students are more highly motivated when they develop a sense of belonging to the institution or course, construct learner identities, make determined efforts to learning, and subsequently, they gain confidence [26]. Belongingness has been conceived as a student's sense of being accepted, valued, included, and encouraged by teachers and peers, along with a sense of self and that they are an important part of the life and activity of the class [26].

Survey results from consenting students demonstrated that the adaptive learning pedagogy assisted in increasing the interest and strengthening the student's sense of belonging: 84% felt more interested in working on programming questions as a result of the course; 81% felt more interested in working on a challenging program; 72% believed that the selection of their field of study was wise and compatible with their attributes; 94% felt the field of Computer Science was welcoming of them; 75% felt this was where he or she belonged. Additionally, through ANOVA analysis we compared confidence results before and after the course was taken and discovered that our students' levels of confidence were significantly sustained.

Table 1. Predictive results of 4-fold cross validation in graduate courses over four semesters.

Method	PCA-SVM	Corr-SVM	PCA-NBC	Corr-NBC	PCA-KNN	Corr-KNN	PCA-RF	CORR-RF
Recall	0.73	0.82	0.35	0.46	0.62	0.70	0.79	0.98
Accuracy	0.796	0.870	0.796	0.67	0.86	0.916	0.85	0.94
Precision	0.48	0.63	0.47	0.29	0.65	0.84	0.59	0.98
Sensitivity	0.81	0.884	0.90	0.72	0.926	0.96	1.0	1.0
Specificity	0.73	0.828	0.359	0.46	0.625	0.70	0.80	0.33
F1.5	0.80	0.870	0.796	0.67	0.86	0.916	0.85	0.98
AUC	0.772	0.85	0.63	0.59	0.77	0.836	0.83	0.99

Table 2. Predictive results of using the discovered model in table 1 to test our prior-knowledge assessment data in graduate courses over four semesters.

Method	PCA-SVM	Corr-SVM	PCA-NBC	Corr-NBC	PCA-KNN	Corr-KNN	PCA-RF	CORR-RF
Recall	0.82	0.87	0.45	0.45	0.62	0.78	0.93	1.0

Accuracy	0.86	0.95	0.80	0.83	0.90	0.95	0.96	1.0
Precision	0.6	0.88	0.5	0.60	0.79	1.0	0.85	1.0
Sensitivity	0.86	0.97	0.89	0.92	0.97	1.0	0.96	1.0
Specificity	0.83	0.87	0.45	0.45	0.62	0.78	0.94	1.0
F1.5	0.86	0.95	0.80	0.83	0.90	0.95	0.96	1.0
AUC	0.84	0.92	0.67	0.69	0.85	0.89	0.93	1.0

Table 3. Predictive results of 4-fold cross validation in undergraduate courses (Spring 2020).

Method	**PCA-SVM**	**Corr-SVM**	**PCA-NBC**	**Corr-NBC**	**PCA-KNN**	**Corr-KNN**
True Negative	56	54	49	51	57	57
False Positive	1	3	8	6	0	0
False Negative	2	2	1	3	13	7
True Positive	21	21	22	20	10	16
Recall	0.91	0.91	0.96	0.87	0.43	0.70
Accuracy	0.96	0.94	0.89	0.89	0.84	0.91
Sensitivity	0.98	0.95	0.86	0.89	1.00	1.00
Specificity	0.91	0.91	0.96	0.87	0.43	0.70
F 1.5	0.93	0.90	0.87	0.84	0.53	0.77
Precision	0.95	0.88	0.73	0.77	1	1

Table 4. Predictive results of using the discovered model in table 3 to test our prior-knowledge assessment data in undergraduate courses (Fall 2020).

Pre-skills	**PCA-SVM**	**Corr-SVM**	**PCA-NBC**	**Corr-NBC**	**PCA-KNN**	**Corr-KNN**
True Negative	43	46	34	30	54	54
False Positive	14	11	23	27	3	3
False Negative	10	12	9	6	16	22
True Positive	13	11	14	17	7	1
Recall	0.57	0.48	0.61	0.74	0.30	0.04
Accuracy	0.70	0.71	0.60	0.59	0.76	0.69
Sensitivity	0.75	0.81	0.60	0.53	0.95	0.95
Specificity	0.57	0.48	0.61	0.74	0.30	0.04
F1.5	0.54	0.48	0.51	0.58	0.37	0.06
Precision	0.48	0.50	0.38	0.39	0.7	0.25
AUC	0.66	0.64	0.60	0.63	0.63	0.50

IV. DISCUSSION

The Coronavirus pandemic has introduced uncertainty into all aspects of society, including for schools, families, and student life. Some families faced more medical or financial challenges than others. Some students were deprived of opportunities to concentrate on their studies. They had to work multiple part-time jobs to support their family. Many students reported depression or other mental disorders due to the lack of real social life for a year. These difficulties might explain a part of behavior patterns in groups A and D (refer to figure 3).

In the paper, undergraduate data was collected from Spring 2020 and Fall 2020 and graduate data from Spring/Fall 2019, and Spring 2020/2021. Features in our predictive models were selected mainly from course content assessments. Introducing student-oriented social and geographic data would be helpful for us to build models that capture the evolving nature in pandemic or post-pandemic.

V. CONCLUSION

Sequential, parallel, or distributed computing is the core of computer science, which involves courses at different programming levels. Abstract nature in programming can often be hard to teach and difficult for students to grasp. Contemporary students have grown up with massive distractions, relatively low in motivation in general. Teaching in these courses for these generations has particular challenges, like abstract conceptualization, high enrollment, and an emerging explosion of learner diversity in pre-knowledge, confidence in computing systems, time management, problem-solving, and programming skills. To address these problems, we have explored diverse constructivist teaching in both flipped classrooms and remote/online learning. We combined scaffolding with student-focused active learning techniques.

We used two feature selections and four predictive modeling methods to identify at-risk students at the early time. Our data analysis revealed that out of four predictive modeling methods, such as support vector machine (SVM), k-nearest neighbors (KNN), naive Bayes classifier (NBC), and random forest, random forest performed best in minimizing false negative (i.e., type II) error while not increasing false positive(i.e., type I) error significantly. The data-driven approach helps identify at-risk students at an early time and enables instructors to design appropriate interventions or interactions with students

This innovative practice is encouraging. The future work can be to integrate students' racial data and analyze whether the technique helps reduce possible racial inequities generated from the early educational outcomes.

ACKNOWLEDGMENT

We are grateful to Dr. Mohsen Dorodchi for discussion.

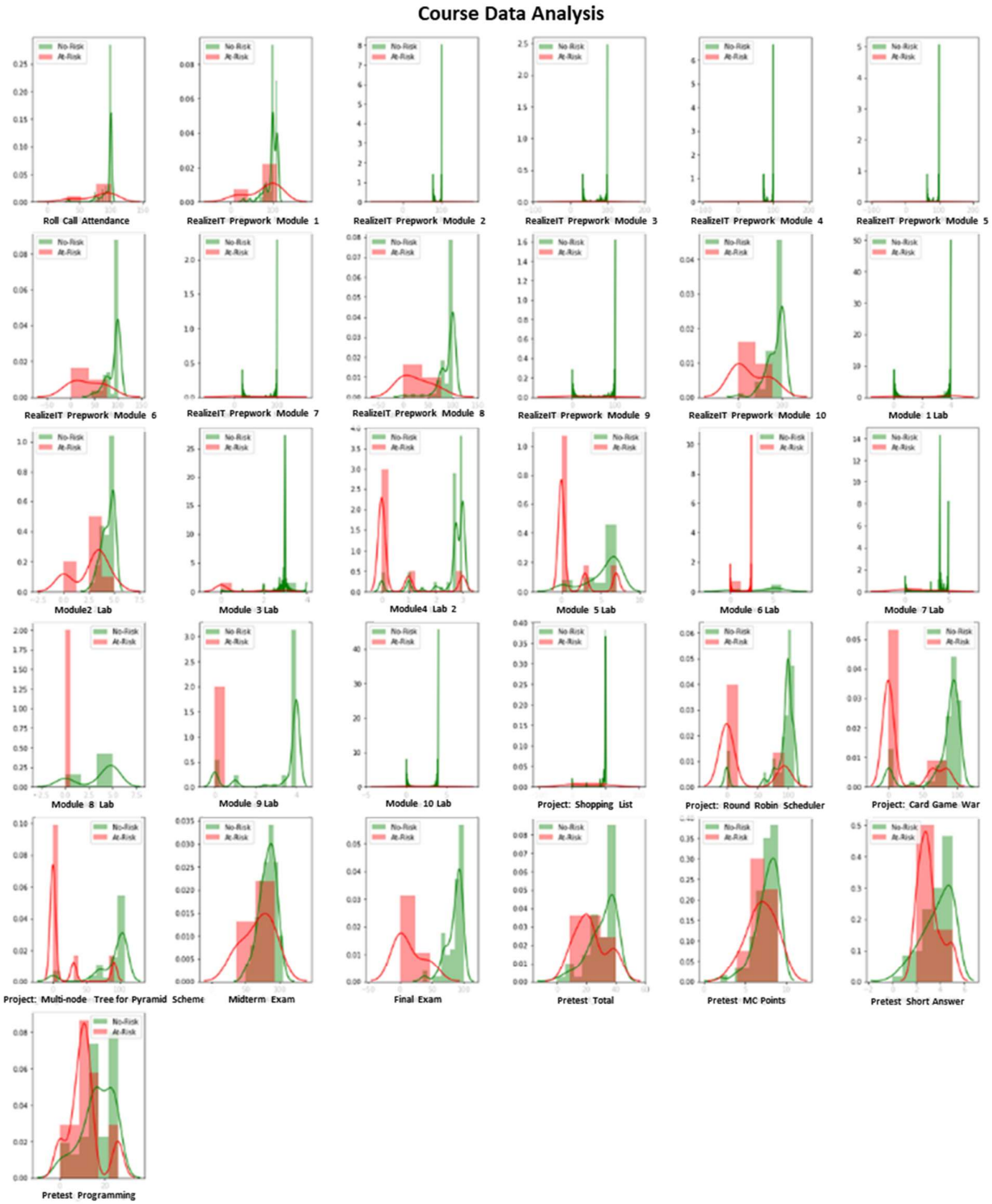

Figure 4 Plotting the data to visualize their distribution of 31 predictor variables over at-risk students (grade <70) and non-risk students. Orange is for at-risk students and green for non-at-risk students.

REFERENCES

[1] Draft, S. (2013). Computer Science Curricula 2013. ACM and IEEE Computer Society, Incorporated: New York, NY, USA.

[2] Gelfand, N., Goodrich, M. T., & Tamassia, R. (1998, March). Teaching data structure design patterns. In Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education (pp. 331-335).

[3] Blum, L., & Cortina, T. J. (2007). CS4HS: an outreach program for high school CS teachers. ACM SIGCSE Bulletin, 39(1), 19-23.

[4] Hazzan, O., Gal-Ezer, J., & Blum, L. (2008, March). A model for high school computer science education: The four key elements that make it!. In Proceedings of the 39th SIGCSE technical symposium on Computer science education (pp. 281-285).

[5] Wang, L. (Ed.). (2005). Support vector machines: theory and applications (Vol. 177). Springer Science & Business Media.

[6] Denoeux, T. (2008). A k-nearest neighbor classification rule based on Dempster-Shafer theory. In Classic works of the Dempster-Shafer theory of belief functions (pp. 737-760). Springer, Berlin, Heidelberg.

[7] Liu, B., Blasch, E., Chen, Y., Shen, D., & Chen, G. (2013, October). Scalable sentiment classification for big data analysis using naive bayes classifier. In 2013 IEEE international conference on big data (pp. 99-104). IEEE.

[8] Dai, W., Xue, G. R., Yang, Q., & Yu, Y. (2007, July). Transferring naive bayes classifiers for text classification. In AAAI (Vol. 7, pp. 540-545).

[9] Belgiu, M., & Drăguţ, L. (2016). Random forest in remote sensing: A review of applications and future directions. ISPRS journal of photogrammetry and remote sensing, 114, 24-31.

[10] Hogan, K. E., & Pressley, M. E. (1997). Scaffolding student learning: Instructional approaches and issues. Brookline Books.

[11] Meunier, F. (2020). Data-Driven Learning: From Classroom Scaffolding to Sustainable Practices.

[12] Hamad, M., & Abdelsattar Metwally, A. (2019). Using Technology towards Promoting Online Instructional Scaffolding: Literature Review. Arab World English Journal (AWEJ) Special Issue: The Dynamics of EFL in Saudi Arabia.

[13] Kim, M. C., & Hannafin, M. J. (2011). Scaffolding problem solving in technology-enhanced learning environments (TELEs): Bridging research and theory with practice. Computers & Education, 56(2), 403-417.

[14] Ge, X., & Land, S. M. (2003). Scaffolding students' problem-solving processes in an ill-structured task using question prompts and peer interactions. Educational technology research and development, 51(1), 21-38.

[15] Xun, G. E., & Land, S. M. (2004). A conceptual framework for scaffolding III-structured problem-solving processes using question prompts and peer interactions. Educational technology research and development, 52(2), 5-22.

[16] Sharma, P., & Hannafin, M. J. (2007). Scaffolding in technology-enhanced learning environments. Interactive learning environments, 15(1), 27-46.

[17] Martinez, A. M., & Kak, A. C. (2001). Pca versus lda. IEEE transactions on pattern analysis and machine intelligence, 23(2), 228-233.

[18] Hall, M. A. (1999). Correlation-based feature selection for machine learning.

[19] Margulieux, L. E., & Catrambone, R. (2021). Scaffolding problem solving with learners' own self explanations of subgoals. Journal of Computing in Higher Education, 1-25.

[20] Long, W. (2017). Knowing behavior patterns helps teaching and learning.

[21] Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. Journal of machine learning research, 3(Mar), 1157-1182.

[22] Kira, K., & Rendell, L. A. (1992). A practical approach to feature selection. In Machine learning proceedings 1992 (pp. 249-256). Morgan Kaufmann.

[23] Dash, M., & Liu, H. (1997). Feature selection for classification. Intelligent data analysis, 1(1-4), 131-156.

[24] Wenger, E. (2010). Communities of practice and social learning systems: the career of a concept. In Social learning systems and communities of practice (pp. 179-198). Springer, London.

[25] Wenger, E. (1999). Communities of practice: Learning, meaning, and identity. Cambridge university press.

[26] Murray, G. (2014). The social dimensions of learner autonomy and self-regulated learning. Studies in Self-Access Learning Journal, 5(4), 320-341.