2026 ASEE ANNUAL CONFERENCE June 21–24, 2026 | Charlotte Convention Center, NC
Host Sponsor UNC Charlotte
ASEE AMERICAN SOCIETY FOR ENGINEERING EDUCATION
Paper ID #51480

Using Mastery Learning Data to Predict Programming Course Performance

Victor Barbosa Slivinskis, Duke University
Ofosu Osei, Duke University
Dr. Genevieve M Lipp, Duke University

Genevieve Lipp received a B.S.E. in mechanical engineering from Duke University in 2010 and a Ph.D. in 2014 with a focus on nonlinear dynamical systems. She is an Assistant Professor of the Practice at Duke University and directs the First-Year Computing program.

Using Mastery Learning Data to Predict Programming Course Performance

Abstract

Mastery learning is based on the theory that nearly all students can succeed in a course given sufficient time and appropriate feedback and practice opportunities. It has been used in many disciplines to allow students to engage in the learning cycle until they demonstrate mastery. We use data from a custom mastery learning platform to predict which students will struggle in a graduate introductory programming course. Data from 100 autograded formative programming exercises, including grade, date graded, whether passed, timeliness, and number of submissions, are analyzed together with summative student performance on three tests, a final exam, and an overall course grade. Results include a rule-based model and machine learning model for predicting student performance, as well as practical recommendations for research-based mastery learning policies. A database tool is proposed to monitor student performance on formative tasks and alert them when a change of approach is advised.

Introduction

Mastery learning is grounded in the premise that nearly all students can achieve high levels of understanding when provided with sufficient time, high-quality instruction, and targeted feedback [1]. Decades of empirical work support this premise, demonstrating improvements not only in student achievement but also in student confidence and attitudes toward learning [2]. As a result, mastery learning frameworks have been increasingly adopted in computing and engineering education, particularly in introductory programming contexts where student preparation and pacing vary widely [3].

Despite its demonstrated benefits, mastery learning presents practical challenges in time-constrained academic environments. Bloom's original vision emphasized self-paced progression and multiple iterations of assessment and remediation, conditions that are difficult to fully realize within a fixed-length semester. In practice, instructors must balance opportunities for repeated practice with institutional schedules, grading logistics, and the cognitive load placed on students. In such settings, mastery-based courses risk leaving some students behind if emerging difficulties are not identified early enough for meaningful intervention.

This challenge is particularly pronounced in intensive graduate-level programming courses, where students enter with heterogeneous backgrounds and are expected to rapidly acquire foundational skills. The present study focuses on ECE 551, a required introductory programming

course for graduate engineering students at Duke University. The course follows *The Seven Steps* instructional framework developed by Hilton [4], emphasizing algorithmic thinking, code tracing, and iterative problem solving. Over a single 15-week semester, students progress from basic computational thinking to advanced topics in C and C++, including dynamic memory management, object-oriented programming, and core data structures and algorithms. Mastery of this material is essential, as many students begin interviewing for software development roles during or immediately after the course.

To support mastery learning at scale, ECE 551 employs a custom platform delivering approximately 100 autograded formative programming exercises. Each exercise allows multiple submission attempts and provides immediate feedback, enabling students to engage in repeated practice prior to summative assessments. Figure 1 illustrates the mastery cycle as presented to students for a single course unit. While this structure has led to strong overall learning outcomes, instructors have observed that some students begin to fall behind early in the semester, often without obvious warning signs until performance on the first major exam.

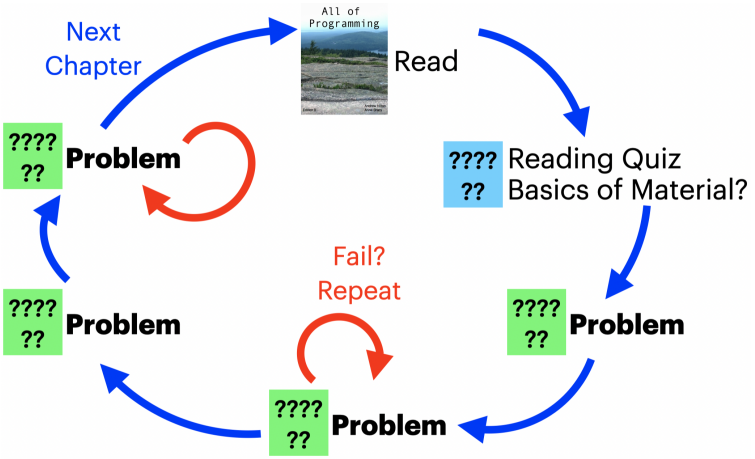

Figure 1: The mastery learning cycle for one chapter of the course, as explained to students.

These observations motivate a central question of this work: *Can behavioral data generated through mastery learning platforms be used to identify students who are likely to struggle, early enough to enable effective instructional intervention?* While prior work has explored mastery learning outcomes and student engagement patterns, fewer studies have focused on translating fine-grained formative assessment metadata into actionable, early-warning tools for instructors in graduate-level computing courses.

The objective of this study is to develop and evaluate a predictive model that identifies students at risk of struggling as early as possible in the semester, using only data generated through routine interaction with mastery-based programming exercises. Specifically, this work makes three contributions:

1. We demonstrate that behavioral signals derived from formative programming exercises—including submission timing, frequency, and deviation from typical engagement patterns—contain predictive information about later student performance.

2. We introduce an interpretable, rule-based *outlier score* that captures heterogeneous manifestations of struggle and complements standard machine learning features.
3. We present a deployable, database-backed system that integrates predictive modeling into instructional workflows, enabling instructors to monitor risk and initiate timely, personalized support.

Figure 2 illustrates how the proportion of students classified as struggling evolves over the course of the semester following each major assessment. By operationalizing mastery learning data in this way, the proposed approach seeks to bridge the gap between learning analytics research and day-to-day instructional decision-making, supporting the core mastery learning principle that all students can succeed when challenges are identified and addressed early.

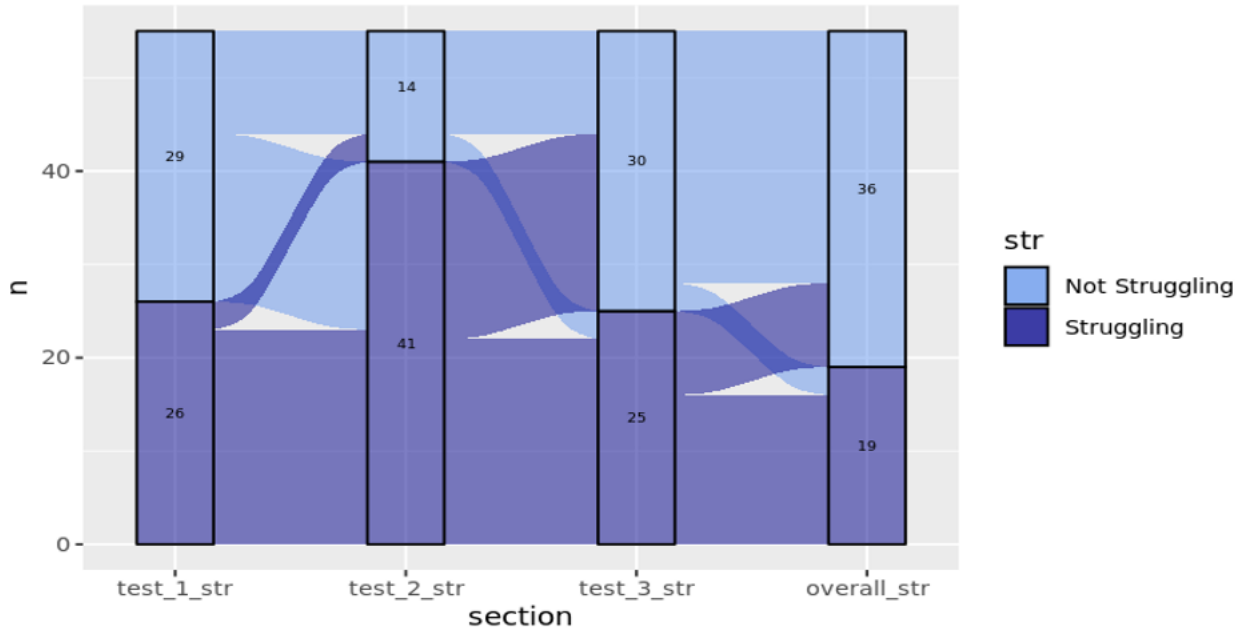

Figure 2: Classification of students as “struggling” or “not struggling” after each test.

Methods

Course Context and Participants This study was conducted in ECE 551, an intensive graduate-level introductory programming course at an R1 research institution in the United States. The course is required for several engineering graduate programs and enrolls students with diverse prior programming experience. Data were collected during a single 15-week semester (Fall 2023).

The initial dataset consisted of 57 enrolled students. Two students were excluded from analysis: one student withdrew from the course early in the semester, and one student received a medical exemption that prevented completion of the course requirements. The final analytic dataset therefore included 55 students.

All participants were graduate students enrolled in the course; no undergraduate students were

included. The study used retrospective analysis of de-identified instructional data generated as part of normal course operation. This work was reviewed by the authors' institutional review board, protocol [2024-0067], and all data were analyzed in de-identified form.

Outcome Definition Struggling was defined as scoring below 90 on Test 1; this threshold (90) reflects a conservative early-term where students typically need additional support.

Step 1: Data Preprocessing Student identifiers, assignment deadlines, and relevant metadata were retrieved from a mastery learning platform database. Autograder log data were processed to compute per-student behavioral statistics, including hours early or late relative to assignment deadlines and the number of submission attempts per assignment. These metrics were subsequently aggregated to engineer higher-level features, including cumulative statistics across assignments, indicators capturing potential "snowballing" effects in submission behavior, and measures of activity concentrated during specific times of day. All data processing and model development were implemented in Python (version 3.10).

In total, 24 features were generated and stored locally to support downstream pattern discovery and model training. Missing values were imputed using the `SimpleImputer` from `scikit-learn`, with categorical variables imputed using the mode and continuous variables using the mean. Non-numerical categorical features were encoded using `LabelEncoder`[5].

Step 2: Outlier Score Construction Exploratory analysis indicated that students who struggled on Test 1 did not exhibit uniform deficits across all behavioral features. Instead, difficulty tended to manifest as *atypical* engagement patterns in a small and variable subset of features, such as extreme lateness, unusually high numbers of submission attempts, or irregular temporal work patterns. Importantly, the specific features exhibiting atypical behavior differed across students. This heterogeneity motivated the development of an interpretable, rule-based *outlier score* designed to aggregate diverse manifestations of struggle into a single scalar indicator.

For each engineered behavioral feature, students were partitioned into struggling and non-struggling groups based on Test 1 performance. Feature distributions were compared using multiple complementary criteria capturing different modes of deviation. These included differences in central tendency (e.g., mean or median shifts), differences in within-group variability (standard deviation and interquartile range), and differences in the prevalence of extreme or outlying values. Population-level outliers were defined using a robust interquartile range (IQR) criterion computed across the full cohort, flagging values outside the interval

$$[Q_1 - 1.5 \cdot IQR,\ Q_3 + 1.5 \cdot IQR].$$

Additional indicators captured whether struggling students were more likely to occupy distributional extremes or fall outside empirically derived feature bounds.

These comparisons yielded a per-feature *quality profile* describing how strongly and in what manner each feature differentiated struggling from non-struggling students. Rather than relying on any single criterion, the model treated these signals as complementary, reflecting the fact that

difficulty may appear as shifted averages for some behaviors and as increased variability or extremity for others.

To compute an individual student's outlier score, each feature-level signal was combined using a weighted additive scheme. For a given feature, students received points when their behavior aligned with patterns observed more frequently among struggling students (e.g., exhibiting outlier behavior in a feature where struggling students showed elevated outlier prevalence). Weights controlling the relative influence of central tendency shifts, dispersion differences, and outlier prevalence were treated as tunable parameters.

Optimal weights and the final decision threshold were selected via randomized parameter search on the training data. Candidate parameter sets were evaluated by computing aggregate scores for all students and assessing classification performance under different thresholds. Depending on instructional priorities, the objective function emphasized either overall accuracy or recall of struggling students. The resulting outlier score thus represents a calibrated measure of similarity between an individual student's behavioral profile and patterns historically associated with academic difficulty.

The outlier score was designed to be interpretable and transparent, enabling instructors to understand which behavioral dimensions contributed to elevated risk. Rather than serving as a standalone classifier, this score was incorporated as an additional engineered feature for downstream machine learning models, allowing the predictive system to leverage both aggregated risk signals and the underlying behavioral data.

Step 3: Machine Learning Models To address class imbalance, the Synthetic Minority Over-sampling Technique (SMOTE; `scikit-learn`) [6] was applied to reduce the risk that the model learns defaults to predicting the majority class. This increased the dataset from 19 struggling and 36 non-struggling students to 36 students in each class. Because the dataset is small, SMOTE may not perfectly reflect the true distribution; in future work, we plan to compare against class-weighted and non-oversampled baselines. Principal Component Analysis (PCA) was then used to reduce the feature space from 24 to 10 dimensions, improving numerical stability and mitigating multicollinearity [7]. Before PCA, the strongest signals were concentrated in timeliness (hours early/late), submission intensity (attempt counts), and cumulative engagement patterns across assignments.

All preprocessing steps, including imputation and encoding, SMOTE, and PCA, were implemented within a single machine learning pipeline and were fit exclusively on the training data in each cross-validation fold, with the learned transformations applied only to the corresponding held-out fold, in order to prevent information leakage.

More than 20 distinct model architectures were evaluated, including Random Forests, Gradient Boosting methods, and Support Vector Machines, resulting in 46 total model configurations. Models were trained and evaluated using 15-fold stratified cross-validation (`StratifiedKFold`) [8]. The five highest-performing unique models, ranked by F1 score (Eq. 1), were selected to construct an ensemble voting classifier. These models included gradient boosting approaches (e.g., LightGBM and XGBoost), classical models (e.g., logistic regression), and tree-based models (e.g., Random Forests), as well as voting classifiers composed of

combinations of these approaches.

$$F_1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{1}$$

A hard voting strategy was employed using the five highest F1-scoring models (RandomForest [9], CatBoost [10], HistGradientBoosting [11], AdaBoost [12], and GradientBoosting [13]).

Step 4: Database Tool The proposed system employed a dual-database architecture to manage and process student performance data. A PostgreSQL database was used to simulate the production mastery learning platform database, while a SQLite database served as a downstream data store for engineered features and model predictions.

The PostgreSQL database replicated the structure and data types of the production environment, storing student identifiers, assignment information, and autograder logs. The SQLite database enabled efficient feature engineering and prediction workflows, maintaining a clear separation between raw source data and processed analytical outputs. This architecture was chosen to allow seamless transition to a production environment while supporting independent operation during development and testing.

Step 5: System Pipeline At the workflow level, the system integrated data processing and prediction capabilities through a command-line interface (CLI). The pipeline operated in several stages. First, raw data were extracted from the PostgreSQL database and transformed into engineered features stored in the SQLite database. Next, the trained ensemble model processed these features to generate predictions of student performance. Finally, the system identified and reported students predicted to struggle, through CLI commands.

Instructors could execute a `predict` command to view flagged students and determine whether to initiate interventions, such as email outreach. The CLI reports the number of flagged students using the Test 1 threshold (score < 90) and persists these predictions to the SQLite `Predictions` table for downstream reporting. Additional CLI commands allowed instructors to inspect detailed performance metrics for individual students and monitor aggregate class-level trends, enabling timely intervention and personalized support throughout the semester.

Results

Following model training and evaluation, a hard voting classifier composed of Random Forest, CatBoost, HistGradientBoosting, AdaBoost, and Gradient Boosting models was selected as the final predictive model. All reported performance metrics reflect averages across 15-fold stratified cross-validation. These results come from one cohort (N=55), so we treat them as initial evidence rather than a claim of broad robustness.

The model achieved an F1 score of 88.3%, indicating a strong balance between precision and recall when identifying struggling students. This balance is particularly important in instructional settings, where false negatives may delay needed support, while false positives may result in

Metric	Result
Accuracy	89.3%
Precision	90.8%
Recall	86.2%
F1 Score	88.3%

Table 1: Cross-validated performance metrics for the final voting classifier.

unnecessary intervention. The achieved accuracy of 89.3% suggests that the model reliably distinguishes between struggling and non-struggling students using only formative assessment metadata.

Figure 3 shows the confusion matrix for the voting classifier, aggregated across cross-validation folds. The matrix indicates that most struggling students were correctly identified, with relatively few false negatives. While some non-struggling students were misclassified as struggling, this conservative bias aligns with the pedagogical goal of prioritizing early support over missed intervention opportunities.

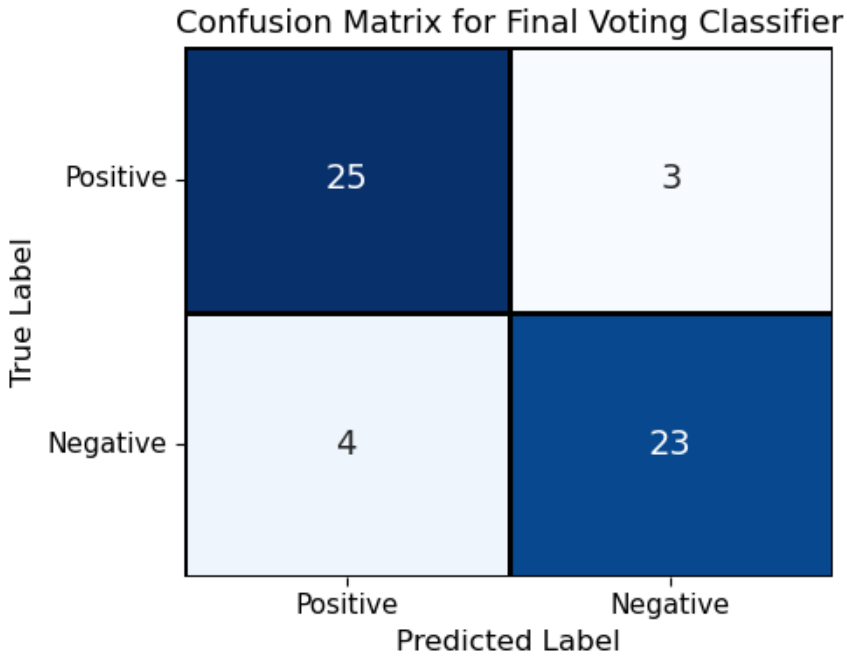

Figure 3: Confusion matrix for the ensemble voting classifier.

Receiver operating characteristic (ROC) curves were computed to further assess discriminative performance. As shown in Figure 4, the model achieved an area under the curve (AUC) of 0.95, indicating excellent separation between struggling and non-struggling students across a range of classification thresholds. This high AUC suggests that the model's predictions are robust to threshold selection and may be adapted to different instructional intervention strategies.

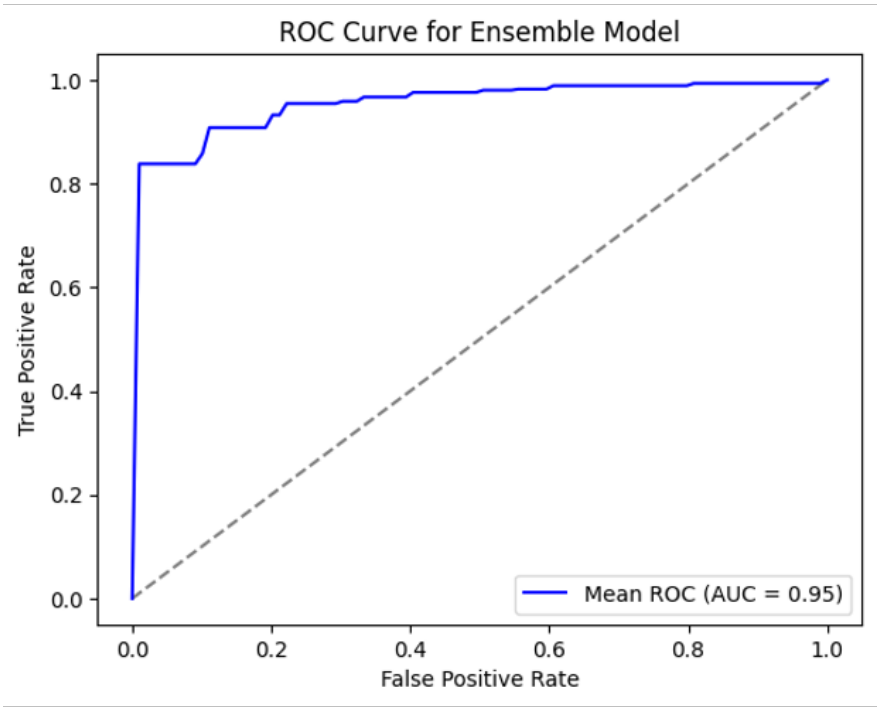

Figure 4: Receiver operating characteristic (ROC) curve for the voting classifier.

Operational Output: Flagged-Student Report

In addition to cross-validated performance, the system produces a simple report for instructors through the CLI. After scoring, predictions are persisted in the SQLite database and displayed as a flagged-student table, allowing instructors to review before deciding whether to initiate outreach. In this setting, predictions can be generated after the first few assignments, using only formative behavior logs collected before Test 1.

Student ID	Predicted status
S_022	Struggling
S_074	Struggling
S_055	Struggling
S_033	Struggling
S_041	Struggling
S_032	Struggling
S_057	Struggling
S_034	Struggling
S_019	Struggling
S_063	Struggling
S_031	Struggling

Table 2: Example excerpt of the prediction report produced by the early-warning CLI tool (anonymized).

```
% python early_warning_cli.py predict
Predicting with best model...
Predictions stored successfully in the Predictions table.
Summary: N=55 | flagged struggling=11 | threshold=90 | cutoff=026_ch07_rq
Table: Early-warning flags (students predicted to struggle)
```

NetID	Status
22	❌ Struggling
74	❌ Struggling
55	❌ Struggling
33	❌ Struggling
41	❌ Struggling
32	❌ Struggling
57	❌ Struggling
34	❌ Struggling
19	❌ Struggling
63	❌ Struggling
31	❌ Struggling

Figure 5: Example CLI output showing the number of flagged students and an excerpt of the flagged-student table (anonymized).

Discussion

This work demonstrates that behavioral data generated through a mastery learning platform can be leveraged to identify students at risk of struggling in an intensive graduate-level programming course. By combining engineered features derived from formative assessment behavior with a rule-based outlier score and ensemble machine learning models, we achieved strong predictive performance while maintaining interpretability and practical relevance for instructional decision-making.

One key finding is that no single behavioral feature consistently distinguished struggling students. Instead, difficulty manifested through heterogeneous patterns of deviation across multiple dimensions, including timeliness, submission frequency, and cumulative workload behavior. This observation motivated the construction of the outlier score, which proved valuable as a unifying indicator of risk. Rather than relying on strict thresholds for individual features, the outlier score captures the intuition that students who struggle often do so in different ways, but with a shared tendency toward atypical engagement patterns. Incorporating this score as a feature improved downstream model performance and aligned well with instructors' qualitative observations of student behavior.

The ensemble voting classifier achieved an F1 score of 88.3%, indicating balanced precision and recall in identifying struggling students. This balance is particularly important in an educational setting, where false negatives may delay needed support, while false positives may unnecessarily concern students or instructors. The use of multiple model families within the ensemble suggests that predictive signal is distributed across both linear and non-linear relationships in the data. Importantly, the strong performance was achieved using only formative assessment metadata,

without relying on demographic variables or prior academic history, supporting the ethical and practical appeal of the approach.

Beyond predictive accuracy, the proposed database-backed pipeline highlights a path toward operationalizing learning analytics in real instructional contexts. The separation of raw production data from engineered analytical features supports reproducibility, data governance, and scalability. The command-line interface enables instructors to query predictions and inspect student-level trends without requiring expertise in machine learning, lowering the barrier to adoption. This design emphasizes decision support rather than automation, preserving instructor judgment while providing timely, data-informed insights.

Several limitations should be noted. The dataset is relatively small and drawn from a single course offering, which may limit generalizability. Although SMOTE was used to address class imbalance, synthetic oversampling may not fully capture the true distribution of struggling behaviors. Additionally, while PCA improved numerical stability, it reduced direct interpretability of individual features, which may be a consideration for instructors seeking fine-grained explanations. Future work should explore alternative dimensionality reduction or regularization approaches that preserve feature semantics.

There are several promising directions for future research. Longitudinal validation across multiple semesters would help assess model robustness and temporal stability. Integrating textual or code-level features from student submissions may further enhance predictive power. Finally, controlled studies examining the impact of instructor interventions triggered by model predictions would be essential to evaluate whether early identification translates into improved student outcomes, closing the loop between analytics and pedagogy.

Together, this study provides evidence that mastery learning data contain actionable signals of student struggle well before summative assessments. When combined with thoughtful feature engineering, interpretable heuristics, and robust machine learning models, these data can support timely, personalized interventions that align with the core principles of mastery learning.

Conclusion

Behavioral data from mastery-based formative programming exercises contain meaningful early signals of student difficulty in graduate-level computing courses. By combining an interpreability-focused outlier score with an ensemble machine learning approach, the proposed system achieved strong cross-validated performance while remaining practical for instructional deployment.

Importantly, the model relies only on routine platform interaction data, avoiding dependence on demographic or historical academic information. This supports the feasibility of integrating early-warning analytics directly into mastery learning environments to inform timely, targeted instructional support.

Future work will focus on validating the approach across additional course offerings and institutions, as well as determining how early in the semester reliable predictions can be made. We also plan to evaluate the real-world impact of instructor interventions triggered by model alerts, closing the loop between predictive analytics and improved student outcomes.

More broadly, this work demonstrates a pathway for translating fine-grained mastery learning data into actionable decision-support tools that help realize Bloom's vision: enabling all students to succeed through timely identification and support.

References

[1] B. S. Bloom, "Learning for mastery," Regional Education Laboratory for the Carolinas and Virginia, Tech. Rep., 1968.

[2] C.-L. C. Kulik, J. A. Kulik, and R. L. Bangert-Drowns, "Effectiveness of mastery learning programs: A meta-analysis," *Review of Educational Research*, vol. 60, no. 2, pp. 265–299, 1990.

[3] B. McCane, C. Ott, N. Meek, and A. Robins, "Mastery learning in introductory programming," in *Proceedings of the Nineteenth Australasian Computing Education Conference*, 2017, pp. 1–10.

[4] A. D. Hilton, G. M. Lipp, and S. H. Rodger, "Translation from problem to code in seven steps," in *Proceedings of the ACM Conference on Global Computing Education*, 2019, pp. 78–84.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. VanderPlas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[6] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[7] I. T. Jolliffe, *Principal Component Analysis*. New York: Springer, 2002.

[8] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, 1995, pp. 1137–1143.

[9] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[10] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: Unbiased boosting with categorical features," in *Advances in Neural Information Processing Systems*, vol. 31, 2018, pp. 6638–6648.

[11] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems*, vol. 30, 2017, pp. 3146–3154.

[12] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[13] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.