A Robust Machine Learning Technique to Predict Low-performing Students

SOOHYUN NAM LIAO, University of California, San Diego, USA
DANIEL ZINGARO, University of Toronto Mississauga, Canada
KEVIN THAI, CHRISTINE ALVARADO, WILLIAM G. GRISWOLD, and LEO PORTER,
University of California, San Diego, USA

As enrollments and class sizes in postsecondary institutions have increased, instructors have sought automated and lightweight means to identify students who are at risk of performing poorly in a course. This identification must be performed early enough in the term to allow instructors to assist those students before they fall irreparably behind. This study describes a modeling methodology that predicts student final exam scores in the third week of the term by using the clicker data that is automatically collected for instructors when they employ the Peer Instruction pedagogy. The modeling technique uses a support vector machine binary classifier, trained on one term of a course, to predict outcomes in the subsequent term. We applied this modeling technique to five different courses across the computer science curriculum, taught by three different instructors at two different institutions. Our modeling approach includes a set of strengths not seen wholesale in prior work, while maintaining competitive levels of accuracy with that work. These strengths include using a lightweight source of student data, affording early detection of struggling students, and predicting outcomes across terms in a natural setting (different final exams, minor changes to course content), across multiple courses in a curriculum, and across multiple institutions.

CCS Concepts: • **Social and professional topics** → **Computing education**; **Computer science education**;

Additional Key Words and Phrases: Peer instruction, machine learning, at-risk students, prediction, multi-institution, cross-term, clicker data

ACM Reference format:
Soohyun Nam Liao, Daniel Zingaro, Kevin Thai, Christine Alvarado, William G. Griswold, and Leo Porter. 2019. A Robust Machine Learning Technique to Predict Low-performing Students. *ACM Trans. Comput. Educ.* 19, 3, Article 18 (January 2019), 19 pages.
https://doi.org/10.1145/3277569

This work was supported in part by NSF Award No. 1712508.

Authors' addresses: S. N. Liao, K. Thai, C. Alvarado, W. G. Griswold, and L. Porter, University of California, San Diego, 9500 Gilman Dr. La Jolla, CA, 92093, USA; emails: snam@eng.ucsd.edu, kthai1994@gmail.com, cjalvarado@eng.ucsd.edu, wgg@cs.ucsd.edu, leporter@eng.ucsd.edu; D. Zingaro, University of Toronto Mississauga, 3359 Mississauga Rd, Mississauga, ON, L5L 1C6, Canada; email: daniel.zingaro@utoronto.ca.

1946-6226/2019/01-ART18 $15.00
https://doi.org/10.1145/3277569

1 INTRODUCTION

University administrators and faculty continue to be concerned by student time-to-graduation and retention. Elevated time-to-graduation increases student financial burdens and stresses limited university resources. Low student retention, particularly in STEM fields, can lead to workplace shortages.

Course failure rates affect both time-to-graduation and student retention. Failing a course can increase a student's time to degree or cause the student to leave the major or the institution. In addition, there are affective costs to failure, such as reduced self-efficacy [5]. For the institution and the instructor, student failure means wasted resources and lost opportunity for student success. As a result, university personnel, instructors, and students share the goal of reducing failure rates [9, 43, 50].

Those seeking to lower failure rates must contend in particular with the recent spike in university enrollments and class sizes. According to the National Center for Educational Statistics [26], undergraduate enrollment in U.S. postsecondary institutions rose from 13.2 million students in 2000 to 17.0 million students in 2015, an increase of more than 25 percent. Computer science, the focus of this study, has seen even greater rates of increase in enrollment beginning in 2006 [14]. However, this growth in students is not matched by commensurate growth in instructional faculty, resulting in increased class sizes (or additional course offerings) for computer science courses in more than 47% of U.S. postsecondary schools. In a large class, it is more difficult for instructors to keep track of individual students' progress, and struggling students are more likely to slip through the cracks.

At the same time, technology-based teaching practices sometimes employed in large classes not only improve student outcomes in these classes but also generate a rich data set that can be used to identify struggling students automatically. In this study, we focus specifically on one such practice called Peer Instruction [15]. Peer Instruction is an active-learning pedagogy used in many disciplines [15, 25, 28, 31, 45, 48, 55] that revolves around students answering and discussing meaningful conceptual questions with their peers and the instructor. Often, Peer Instruction is paired with hand-held devices (e.g., clickers) that allow the instructor to quickly view student answers, and this clicker response data is automatically collected and stored. Our work builds on recent evidence that student performance on Peer Instruction questions is correlated with student outcomes [35] and predictive of students who are at risk in an introductory programming class [24].

In this work, we provide a method for using clicker data collected from Peer Instruction activities to automatically identify students at risk of failure so that these students can receive support and take corrective action. Our goals for this identification technique are (1) that it can predict at-risk students accurately and sufficiently early in a term that corrective steps are possible, (2) that data collection is sufficiently lightweight to encourage instructor adoption, and (3) that it is robust in the sense that (a) it works on courses across the curriculum and across institutions, and (b) a model from a previous term applies in subsequent terms despite minor changes to course content.

To the best of our knowledge, our study is the first to apply a single modeling methodology across a variety of courses within a field (including both lower- and upper-division courses) and across different institutions and to use prior-term data for predictions. In this study, we applied our methodology to data from five different courses at two different institutions. We applied a single modeling method using support vector machines [13] to perform a binary classification of whether or not students are expected to perform well on the final exam. For each course, a model is built using a prior term's student clicker data and final exam scores, which is then applied on data from the first three weeks of a future term. Our method generates models that are shown to correctly identify at least 62% of at-risk students when using an instructor-determined threshold of the bottom 40% of the class.

Table 1. Synthesis of Related Work (★: Present Paper)

Ref	Study Focus	Year	Presage Variables		In-progress Variables		Predictive†	Multiple Contexts	Model Tested with Different Cohort
			Performance and Background	Behavior & Attitude	Performance	Behavior and Attitude			
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
[17]	Institution	1968	✓		✓			✓	
[40]	Institution	1997	✓		✓				
[47]	Institution	2010	✓		✓				
[11]	Institution	2014	✓				✓		
[51]	Institution	2014	✓				✓		
[21]	Program	2001	✓						
[44]	Program	2003	✓						
[1]	Program	2009	✓						
[19]	Program	2011	✓						
[42]	Program	2011	✓		✓				
[27]	Course	1979	✓	✓			✓		
[41]	Course	1986	✓						
[53]	Course	2001	✓	✓					
[6]	Course	2006	✓	✓		✓	✓	✓	
[20]	Course	2006				✓			
[52]	Course	2013				✓			
[8]	Course	2015				✓			
[4]	Course	2015			✓	✓	✓		
[24]	Course	2016			✓		✓		✓
[2]	Course	2017			✓	✓	✓		
[10]	Course	2017			✓	✓	✓		✓
★	**Course**	**2018**			✓		✓	✓	✓
[12]	Module	1994			✓				

†Checkmark indicates predictive modeling, unmarked indicates descriptive modeling.

We continue in Section 2 by describing prior work on predicting students' learning outcomes from multiple disciplines. Our research questions are then provided in Section 3. Section 4 describes the context of our study, including a brief discussion of the Peer Instruction pedagogy and the role of the studied courses in the computer science curriculum. In Section 5, we describe our modeling process and evaluation of the data analysis. Section 6 provides our results, and Section 7 offers a discussion and exposition of broader implications.

2 BACKGROUND

Predictions of student outcomes at universities and colleges have been investigated at all levels: retention at the institution level, completion/graduation at the program level, success at the course level, and understanding at the module level within a course. Due to differing timescales and outcomes of interest, different predictors have been used at each level. Modeling techniques and how they are evaluated also vary widely. Table 1 provides a high-level comparison of the present study and the prior studies discussed below.

We note that predicting student outcomes has been a rich area of study for decades. Prior studies have focused on modeling students' performance to improve retention rates, predict exam scores, or provide timely aid to at-risk students. The earliest studies concentrated on immutable factors that students hold at the start of class, such as gender, age, and SAT scores [40]. Additional factors have been added in recent research including student attitudes [16], student behaviors during the term [2, 7, 54], and the use of in-class formative assessment data [24]. However, the vast majority of previous modeling techniques are either based on data that may be difficult for an instructor to obtain, applied to a single course, applied at a single institution, or modeled and tested with a single term of data.

2.1 Types of Information Used for Prediction

A variety of independent variables has been used in generating prediction models. To simplify presentation, we have chosen to classify these variables as presage variables or in-progress variables. Presage variables refer to those variables that are available or determined before modeling is initiated; for example, high school Grade Point Average (GPA) or math background. However, in-progress variables are measures gathered in the context of the outcome being assessed; for example, midterm grade or timings of assignment submissions. Columns 4–7 in Table 1 indicate the kinds of independent variables that were used in prior work.

We further subdivide presage variables into (1) performance and background variables (e.g., high school GPA, gender, age, socioeconomic status) and (2) behavioral and attitudinal variables (e.g., study habits, motivation). We similarly divide in-progress variables into (1) performance variables and (2) behavioral and attitudinal variables.

Research-based use of student data depends greatly on how the data is collected and its availability to researchers. For example, most performance and background presage variables are already collected automatically for other purposes and stored in registrar or admissions databases. That said, this data may not be readily-available for instructor and researcher use. Similarly, class-level in-progress performance variables (e.g., compilation errors, keystroke-level logging) are often not captured by standard learning management systems and, even when such data is available, might not be released by research boards for incorporation into research.

The rest of this section first examines how researchers evaluate the success of their studies and then surveys a variety of related projects. We then describe how our present study relates to prior work in predicting outcomes in computing.

2.2 Model Creation and Evaluation

The modeling methods for predicting student outcomes are generally statistical, ranging from simple correlation studies to machine-learning models such as regression, decision trees, and random forests.

Authors of some studies sought to demonstrate the robustness of their techniques by applying their methods at different institutions, on different courses, using different instructors, and so on. Robustness is particularly important at the course and module levels: this would suggest that one modeling technique could be used across the curriculum. Column 9 in Table 1 indicates whether authors of prior work successfully applied their methods in multiple contexts.

Model evaluation techniques vary in the literature as well. Some researchers use descriptive modeling, where relationships between independent and dependent variables are studied on the training set itself. This is useful for studying relationships in past data, but is not intended for making predictions about future occurrences. For such predictions, researchers use predictive modeling, where part of the data (the test set) is held back and used to evaluate the accuracy of the model. Column 8 in Table 1 indicates whether prior work employed predictive modeling. In these predictive approaches, the test set can be created in one of two primary ways. The first is to divide a single dataset into training and test sets. This requires only one dataset, but can be problematic as it does not necessarily show predictability for a new cohort (of students, of a course, etc.). The second is to gather (at least) two datasets from different cohorts and use one cohort to build the model and the second to evaluate the model. This latter approach is preferred as it shows predictability across cohorts and prevents a model from being overfitted to a single cohort. That said, this form of data collection is time-consuming, and making predictions across cohorts is complicated by natural variation between cohorts. The last column of Table 1 distinguishes between these predictive modeling methods.

Last, there is considerable variation in the metric used to evaluate the model, making it difficult to compare one model to another (and hence to directly compare or contrast prior work). Some models simply measure the correlation between predictive factors and outcomes, while other models generate predictions. Predictive model accuracies are often reported either as a single accuracy level (i.e., what percentage of all students are correctly classified as either at-risk or not-at-risk), or more generally as the area under a Receiver Operating Characteristic (ROC) curve, referred to as the AUC [36]. One important strength of this latter method is that ROC curves quantify the tradeoff between model sensitivity (i.e., the percentage of all at-risk students correctly classified by the model as at-risk) and specificity (i.e., the percentage of all not-at-risk students correctly classified by the model as not-at-risk).

2.3 Institution- and Program-Level Modeling

The top rows of Table 1 summarize some of the work discussed in this subsection. Of primary interest at the institution level is institutional retention. Many studies have emphasized the use of presage variables, which allow prediction even before a student matriculates [38]. While such work has met some success, it quickly became apparent that in-progress variables, such as first-term GPA, were generally more predictive than presage variables [17]. As a result, recent work often includes in-progress variables in addition to presage variables [40, 47].

We make a distinction among (1) descriptive modeling, (2) predictive modeling tested on a separate test set collected from the same cohort, and (3) predictive modeling tested on future cohorts. Although we suggest that (3) is core to the goal of building and using models to help students, none of the prior work in Table 1 at the institution and program level tested the models on future cohorts. As examples of (1), some researchers [40, 47] assess the quality of their regression models using the models themselves. Exemplifying (2), other authors [11, 51] generate predictive models and provide prediction results. However, those authors test their models on a test set pulled from the same cohort on which training occurs, which may cause the models to overfit to that cohort.

Program-level prediction has focused on program entrance criteria and early identification of struggling students. Unlike for institutional-level modeling, the prior work largely focuses on descriptive modeling rather than predictive modeling. For example, in the medical sciences, it is generally found that presage variables (e.g., entrance exams, high school GPA) are correlated with student performance in the program [21, 44], but it is not reported whether these variables can be used to predict student outcomes with any degree of accuracy.

2.4 Course-Level Modeling in Computing

There are numerous studies predicting student outcomes at the course level in other fields (e.g., biology, chemistry, and physics [39], among others). Fully detailing this broad area of study here would be space prohibitive. As such, we focus on work occurring in the context of the present study: computer science.

In computing, prediction has been focused on student performance in introductory computing courses. Modeling student outcomes began in the late 1970s with early work using presage data to predict student performance on assessments or final course grades [27, 41]. In-progress data in the form of attitudinal surveys have also been used to predict exam scores [53]. Both presage and in-progress variables have been combined effectively using regression analysis to predict student programming performance [6].

Other in-progress variables have also been used to determine student performance. In the mid 1990s, the in-progress variable of programming progress was used to aid intelligent tutoring systems [12]. Recent work identified the in-progress variable of clicker data resulting from Peer Instruction courses as a robust predictor of student performance on final exams [35].

Another primary source of data for prediction using in-progress variables is code snapshots of student programming activity. To gather these data, researchers used an instrumented programming environment that regularly records the student's current program in a log file. The approach was then to assess student knowledge by comparing two successive code snapshots. Early approaches examined compiler errors [20, 52], while later work also considered runtime errors [8]. Ahadi et al. improved on these works by employing machine learning; making predictions on a separate term; and adding the correctness of test cases and the number of steps taken to complete assignments as in-progress variables [2, 4]. Ahadi et al. assigned 24 programming assignments in the first week of the class and these assignments mirrored what was required of students on the final exam. Although such data may facilitate early-term modeling efforts, it requires significant out-of-class effort in the first week from students, which could be discouraging.

Similarly, Castro-Wunsch et al. examined many machine-learning models, and applied the trained model to a test set from a subsequent term [10]. That work achieved accuracies between 67% and 72% and failure group accuracies (i.e., sensitivities) between 61% and 77% on the subsequent cohort depending on the model [10]. However, echoing the concern from Ahadi et al., Castro-Wunsch et al. trained their models using data from students completing a large number of programming assignments in the first 4 weeks of class. While it is possible to assign this many assignments in the first few weeks of the term, such an approach requires significant additional effort from the students.

Most closely related to the present study is recent work where student clicker responses were used for modeling student learning in an introductory computer programming class [24]. As with our present study, this data is available without any extra work for the instructor if the instructor has already adopted clicker questions in class. Treating a student's response to each clicker question as a predictive variable for the student's final exam score, Liao et al. [24] used principal component analysis to identify the most predictive clicker questions. The authors then built a linear regression model based on those questions to predict final exam score. The model was trained on students from one cohort and tested with students from the subsequent cohort, achieving 76% accuracy. That work, however, does not demonstrate the efficacy of the modeling approach across institutions and courses.

Considering the research in this subsection in its totality, what is missing is (1) a robust technique based on easily obtainable student data that is capable of making predictions for multiple courses across a curriculum and at multiple institutions, and (2) evaluation of these techniques as they would be applied in real-world instructor settings (e.g., using a model built in a previous term to make predictions for a subsequent term). These are the main contributions of the present article.

3 RESEARCH QUESTIONS

Based on gaps identified in the previous section, this article has two primary research questions:

RQ1: Can a single modeling technique be designed to reliably predict student outcomes across multiple institutions?

RQ2: Can that modeling technique also reliably predict student outcomes across a range of courses in the curriculum of a chosen discipline?

4 STUDY CONTEXT

In this section, we describe the Peer Instruction pedagogy and briefly explore its known benefits for students (which may partially explain its increased adoption in computing). Next, we provide context regarding the courses in the computer science curriculum that appear in this study.

4.1 Peer Instruction

Peer Instruction (PI) [15] is a student-centered active learning method. It was invented in the 1990s for introductory Physics courses, and was later adopted for computer science. It consists of the use of in-class multiple choice questions to which students respond once individually and then again following a group discussion. In many cases, including those in this study, students use clickers to submit their responses. Following the two responses, instructors lead a class discussion about the question to provide common understanding and instructor expertise on the relevant material.

The efficacy of PI has been shown in various disciplines including physics [15, 55], biology [48], mathematics [25, 28], and computer science [29–33, 45, 46]. These studies discovered that PI improves students' learning and their attitudes about learning [15, 55], enhances student participation and understanding [25], and fosters learning even when none of the discussion group members knows the correct answer [48]. In computing, PI has been shown to enhance student learning [46], lower course failure rates [29], and increase the retention rate of students in the major [33]. Moreover, PI is effective in a variety of computing course subjects, including programming, theory, and systems courses [23, 30].

By virtue of using PI coupled with clickers, students naturally generate clicker data in each lecture. In the present work, we leverage this automatically-collected data, and require no further data collection or generation.

4.2 Course Contexts

Table 2 describes the courses used in our study. They are all undergraduate courses, with the top three courses in the table at the lower division level, and Advanced Data Structures (Adv. DataStruct) and Computer Architecture (Architecture) at the upper division level.

All courses except for Architecture are programming-intensive courses. In those courses, students complete programming projects using a particular programming language. The CS1 courses focus on learning how to code, while CS2 and Adv. DataStruct focus on implementing software applications using concepts learned in class.

CS1 is traditionally the first introductory programming course for students. As such, the majority of students are in their first or second year of study. Our dataset includes two CS1 courses. One was taught in the Python programming language (CS1-Python) and the other was taught in the Java programming language (CS1-Java). CS2 is the programming course after CS1. Here, students learn how to write more modular code, often with object-oriented concepts. In addition, CS2 teaches fundamental data structures and how to use them to support interfaces and abstraction. The CS2 course for our study used the same language as the corresponding CS1: Java. Adv. DataStruct teaches more sophisticated data structures material than what is typically covered in CS2 and used the C++ programming language. The Computer Architecture course teaches students about computer systems, specifically how computer processors function.

In summary, the present article covers a wide range of courses with respect to course level (lower- or upper-division) and curricular material (programming, data structures, architecture).

Table 2. Courses Used in the Study

Course	Description	Lower/upper Division
CS1-Python	Learn basic grammar of the Python programming language. Course topics include variables, datatypes, functions, conditionals, and loops. Also learn introductory sorting techniques.	Lower
CS1-Java	Learn basic grammar of the Java programming language. Course topics include variables, methods, conditionals, and loops. Programming exercises concern simple image and audio processing.	Lower
CS2-Java	Learn essential data structures including arrays, lists, stacks and queues. Analyze runtime of different data structures. Compare different sorting algorithms.	Lower
Adv. DataStruct	Learn advanced data structures including trees, hashtables, skip lists, Huffman coding, and Graph Search in the C++ programming language.	Upper
Architecture	Introductory computer architecture. Learn ISAs, performance, single/multi-cycle processors, pipelining, and caches	Upper

To our knowledge, no prior work has predicted at-risk students across such a broad spectrum of courses.

5 STUDY METHODOLOGY

To answer our research questions, we developed a modeling technique and applied it to data from five courses across two institutions. In this section, we describe our dataset and how the collected data was processed to generate prediction models. Additionally, we describe how the generated models were statistically analyzed.

5.1 Data Collection

Our data were obtained from three instructors across two public universities in North America. Detailed information for each course is provided in Table 3. All courses from institution A are 10 weeks long, while CS1-Python at institution B is 12 weeks long. The study, including collection and analysis of this data, was approved by our Institutional Review Board (IRB).

Two of our courses are CS1 offerings at different institutions: CS1-Java from Institution A and CS1-Python from Institution B. In addition to using different programming languages, we note that topic coverage and ordering differ as well. Unfortunately, these differences limit comparisons between the two courses.

The data for each course consists of two different terms of data taught by the same instructor. The earlier term was used to train our prediction model and the later term to test that model. We will refer to the earlier term as the *training set* and the later term as the *test set*. Clicker responses were naturally collected in class, because all the instructors in our dataset taught their courses using Peer Instruction.

Table 3. Dataset Details

Course	Institution	Instructor	Training Set (Size)	Test Set (Size)
CS1-Python	B	Y	Fall 2013 (192)	Fall 2014 (142)
CS1-Java	A	X	Fall 2014 (373)	Fall 2015 (374)
CS2-Java	A	X	Spring 2014 (169)	Spring 2015 (176)
Adv. DataStruct	A	X	Fall 2013 (197)	Winter 2015 (191)
Architecture	A	Z	Fall 2015 (339)	Fall 2016 (266)

5.2 Data Preprocessing

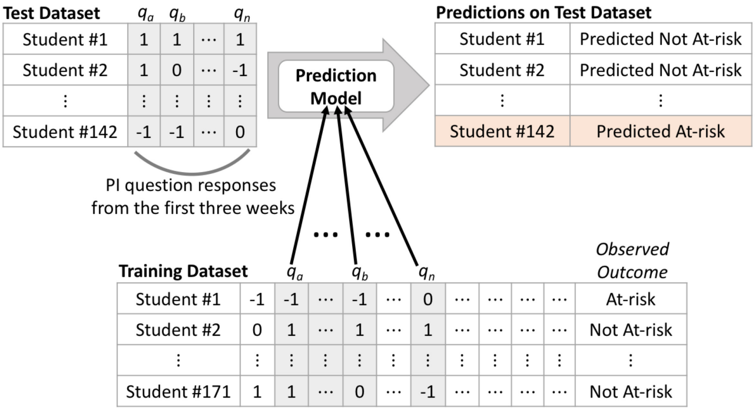
The data used in this study consists of student clicker responses (both individual and group) and raw final exam scores. The available clicker question data comprises every question that the instructor asked, along with all student responses, though only a subset of these questions is used for modeling and prediction, as described below. The data uses unique anonymous identifiers to protect students' identities.

- **Selecting clicker questions:** Our model learns to use *specific* clicker questions to predict course outcomes, so our data must consist only of questions that are present in both terms of a course. Thus, for each course, we discarded any question that was not present in both the training and test set.

  As our goal is to perform early identification of at-risk students, we also discarded clicker questions that were asked after week three in each course. Week three was chosen because the midterms were not administered until at least week four. (Later, in Section 6.4, we will explore how prediction accuracy changes in later weeks.) We also removed students who did not respond to any clicker questions in this three-week time frame.
- **Labeling question correctness:** For each clicker question, we assigned 1 for correct, -1 for incorrect, and 0 for unanswered. This assignment of a unique value (0) for unanswered questions differs from the approach taken by Liao et al. [24], who used data imputation to statistically predict missing clicker responses. Data imputation relies on patterns in available responses to guess missing data values. One concern with imputation is that the imputed responses may be inaccurate, particularly when a student answers few questions. More importantly, we believe that student non-response to a question is a signal in itself, distinct from a correct or incorrect response.
- **Defining the At-risk and Not-At-risk Groups:** Classifying students into two groups allows us to perform binary classification, where our model tries to predict whether a student in the test set will score in the top 60% or the bottom 40% of the final exam. Recent work uses a 50% cutoff [4, 10], i.e., simply predicting whether students are in the top or bottom half of the class, but we selected the 40% cutoff, because the instructors in our study felt that this was closer to the boundary where students were actually at-risk of failing the final or the course. However, we note that this cutoff was a choice, and the model could be trained to meet different thresholds, potentially with different accuracy levels.

5.3 Model Generation

Figure 1 illustrates our model generation process. At a high level, a machine-learning model is built based on the training set (student clicker data and exam scores in one term) and that same model is then used to predict student outcomes in the test set. Specifically, to build the model, we used support vector machines (SVMs) with the radial basis function kernel to train one prediction model for each course. We selected SVMs as they provided the most consistent AUC values across



Test Dataset

	q_a	q_b		q_n
Student #1	1	1	⋯	1
Student #2	1	0	⋯	-1
⋮	⋮	⋮	⋮	⋮
Student #142	-1	-1	⋯	0

Predictions on Test Dataset

Student #1	Predicted Not At-risk
Student #2	Predicted Not At-risk
⋮	⋮
Student #142	Predicted At-risk

Training Dataset

		q_a		q_b		q_n					*Observed Outcome*
Student #1	-1	-1	⋯	-1	⋯	0	⋯	⋯	⋯	⋯	At-risk
Student #2	0	1	⋯	1	⋯	1	⋯	⋯	⋯	⋯	Not At-risk
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Student #171	1	1	⋯	0	⋯	-1	⋯	⋯	⋯	⋯	Not At-risk

Fig. 1. Our machine-learning process.

our datasets and courses compared to other machine-learning models (logistic regression, decision tree, and random forest).

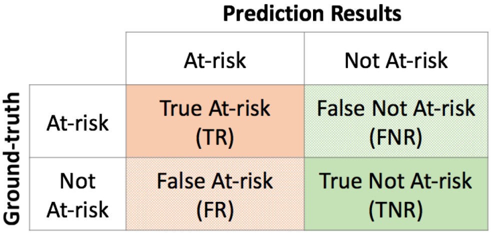
As accuracy may vary based on the particulars of choosing a machine-learning process, we provide more specific details of our methods both for future replication of these results and to inform instructors of how they might perform similar modeling. We note that ongoing work aims to provide instructors with a tool to fully automate this entire process.

After conducting the data preprocessing steps described in the previous subsection, we trained the models. We used a well-known package in R named kernlab [49] for model generation. The following steps describe our training process in detail:

(1) We generate a SVM model, using three modeling parameters: cost, sigma, and classweight. There is an inherent tradeoff between overfitting to the training set and making accurate predictions on the test set. Both cost and sigma influence this tradeoff. Classweight provides a mechanism for indicating the severity of different types of classification errors; for example, it enables a model to apply a larger penalty to misclassifying an at-risk student than a not at-risk student. As a model's accuracy depends on these parameters, we perform a design space search by using different combinations of those parameters. To tune those parameters, we partition the training set (data from the first term) into two subsets: 2/3 of the data as the "training subset" and the remaining 1/3 as the "validation subset." Using different parameters, we build models with the training subset and evaluate their performance based on the resultant AUC for the validation subset. When building these models, we use 10-fold cross validation to avoid model overfitting. Ultimately, we select the parameters (cost, sigma, classweight) that produce the highest AUC.
(2) Given these parameters of cost, sigma, and classweight, we build the SVM model using the entire training set (the first term of data), again using 10-fold cross validation. The resultant model is then used to evaluate performance on the test set (the second term of data).

5.4 Model Evaluation

The purpose of our modeling is to identify students at risk of failure as precisely as we can. While overall accuracy can give us a sense of the percentage of students who are misclassified by the model, it does not reveal the tradeoff between different types of errors that our model might make. Any binary decision model can produce two types of errors: it can incorrectly classify an at-risk



		Prediction Results	
		At-risk	Not At-risk
Ground-truth	At-risk	True At-risk (TR)	False Not At-risk (FNR)
	Not At-risk	False At-risk (FR)	True Not At-risk (TNR)

Fig. 2. Classification categories.

student as being not-at-risk, or it can incorrectly classify a student who is not-at-risk as being at-risk. These errors, along with the two possible correct outputs, are shown in Figure 2, which shows the confusion matrix for the possible outcomes of the classification versus the ground truth of each sample.

In this sense, sensitivity (i.e., recall) and specificity are more useful measures than simple prediction accuracy. Sensitivity measures how accurately a model can detect at-risk students, whereas specificity measures how accurately the model identifies not-at-risk students. The definitions of sensitivity and specificity are as follows:

$$Sensitivity = \frac{\text{TR}}{\text{TR} + \text{FNR}}, \qquad Specificity = \frac{\text{TNR}}{\text{TNR} + \text{FR}}.$$

The SVM model outputs the probability of each student being at-risk. By selecting different probability cutoffs for what is considered at-risk, we can trade off the sensitivity and specificity of a given model. For example, suppose that we wish to maximize sensitivity at the expense of specificity. Then, the solution is simply to predict that everyone is at-risk, yielding a sensitivity of 1.0 and specificity of 0.0. This, however, would be wasteful of limited instructor resources that could be deployed to help the students who were truly at-risk. For this reason, specificity is also important: it indicates how many not-at-risk students are properly identified as not needing assistance.

Receiver Operating Characteristic (ROC) curves [18] are a well-established metric for model evaluation in machine learning, which plot both sensitivity and specificity in a two-dimensional plane. An ROC curve is more comprehensive than other standard metrics in that it provides various combinations of sensitivity and specificity with respect to classification probability thresholds. In addition to the ROC curves themselves, we provide the Area Under the ROC Curve (AUC) as an additional measure of model effectiveness.

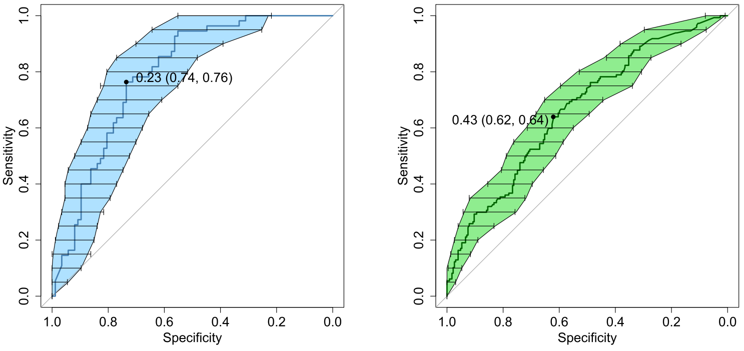
6 RESULTS

In this section, we evaluate the performance of our models for the courses included in this study. We begin by examining how well the models perform both across multiple institutions and across the CS curriculum. Then, we explore how modeling performance changes through the term, by adding clicker data from later weeks.

6.1 Multi-Institutional Analysis (RQ1)

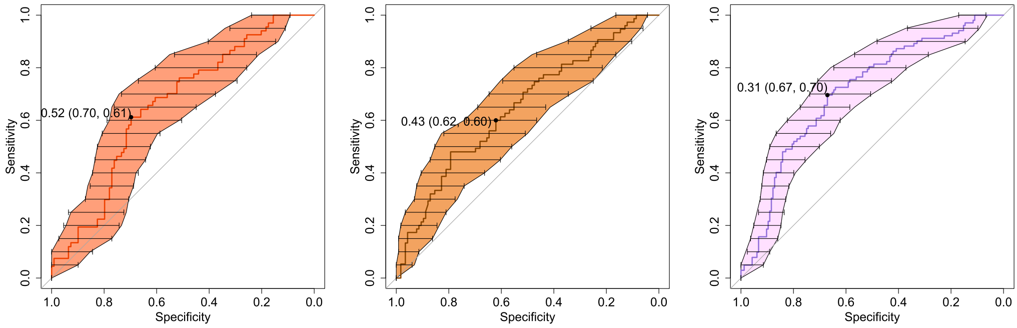
Addressing Research Question 1, we examine whether the modeling approach is successful across institutions. Figure 3 provides the ROC curves for the two CS1 courses taught at different institutions.

In Figure 3, the gray line is $y = x$ and represents what one would expect if the model were simply guessing randomly. As such, an ROC curve of $y = x$ would have an AUC of 0.5. The ideal ROC curve is one that hugs the upper left hand corner as this would indicate that the predictions are exceptionally accurate. The shaded areas of Figure 3 indicate the confidence intervals of specificities



(a) CS1-Python (Inst. B, AUC 0.79) (b) CS1-Java (Inst. A, AUC 0.67)

Fig. 3. Cross-institutional ROC curves.



(a) CS2-Java (AUC 0.67) (b) Adv. DataStruct (AUC 0.65) (c) Architecture (AUC 0.71)

Fig. 4. Cross-curricular ROC curves from institution A.

with respect to the given sensitivities. Each curve has a point labeled with three numbers. The first number is the probability threshold selected by the training dataset for the best combination of sensitivity and specificity. If this probability threshold (first number) is selected, then it produces the specificity (second number) and sensitivity (third number) for the test dataset.

From this figure, we can see that the ROC curve for both courses is significantly better than random. The ROC curve of CS1-Python (AUC = 0.79) is better than that of CS1-Java (AUC = 0.67). Indeed, both specificity and sensitivity are higher for CS1-Python than CS1-Java.

For each of these CS1 courses, we find that our modeling approach provides reasonable classification predictions. The AUC is well above 0.5 (0.79 and 0.67) and at the best probability threshold, the models offer specificities of 0.74 and 0.62 and sensitivities of 0.76 and 0.64. We discuss how these metrics compare to prior work in the next subsection.

Given the differences between institutions and courses, it is to be expected that the models would result in slightly different prediction accuracies. In Section 7, we discuss possible sources of this difference.

6.2 Analysis across Courses in the CS Curriculum (RQ2)

Focusing now on Research Question 2, we examined PI clicker data for four different courses across the CS curriculum. Figure 3(b) along with Figure 4 are the ROC plots of four different courses taught at the same institution covering material ranging from introductory to advanced computing topics.

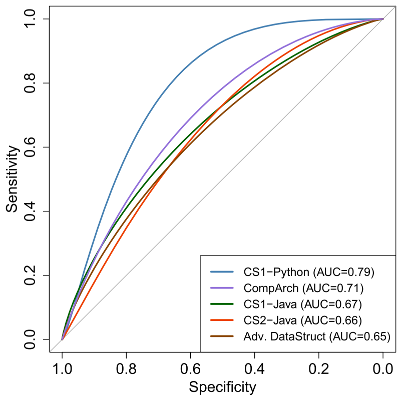


Fig. 5. Smoothed ROC curves.

Table 4. AUCs and 95% Confidence Intervals

Course	AUC	95% Confidence Interval
CS1-Python	0.79	0.72–0.86
CS1-Java	0.67	0.61–0.72
CS2-Java	0.66	0.58–0.74
Adv. DataStruct	0.65	0.57–0.73
Architecture	0.71	0.65–0.77

The results are highly encouraging as the models are able to predict student outcomes for each course reasonably well. Focusing on the best probability threshold for each model, we find that the model consistently achieves a sensitivity between 0.60 and 0.70 and a specificity between 0.62 and 0.70. Similarly, the AUC for each course is between 0.65 and 0.79.

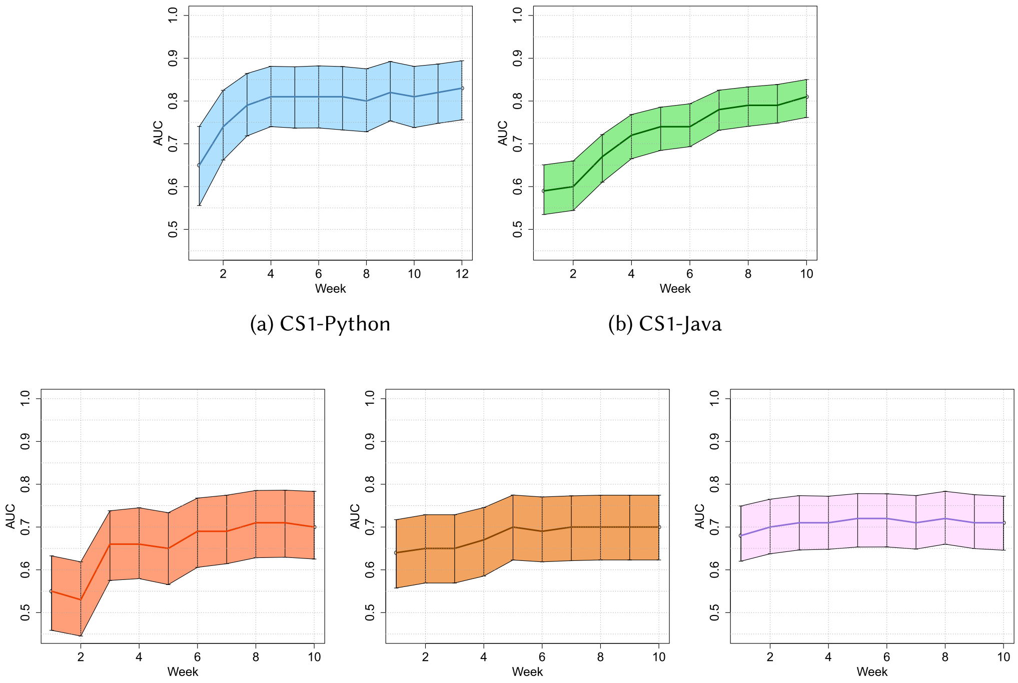
There is no truly comparable study in the literature against which to compare these results. However, our results are consistent with recent studies predicting student outcomes across terms using similar in-progress data in introductory computing courses [4, 10, 24]. Recall from Section 2.4 that one of these studies [24] used data that was similarly lightweight to our approach whereas another study used data that was more heavyweight [10]. As such, our approach is more lightweight and more broadly applicable than prior work while maintaining similar levels of prediction Performance.

6.3 Overall Analysis

Figure 5 and Table 4 provide a summary of results for all five courses when modeling clicker data from the first three weeks of the term. Note that the ROC curves have had their confidence intervals removed and are smoothed to facilitate comparison across the five curves. From the legend of Figure 5, we see that the AUCs of all courses are greater than 0.6, and that the five curves are all above the $y = x$ line. On average, the AUC and 95% confidence interval of the courses are 0.70 and 0.63–0.76, respectively. In addition, the lower bounds of the confidence intervals for all courses are higher than 0.5, indicating that all of our modeling results are statistically different from random guessing.

6.4 Prediction Accuracy over Time

Although the focus of this study is early prediction of student outcomes, we additionally wanted to understand how prediction quality changed over time. Figure 6 provides prediction quality



(a) CS1-Python

(b) CS1-Java

(c) CS2-Java

(d) Adv. DataStruct

(e) Architecture

Fig. 6. AUC and 95% confidence intervals, adding more clicker data.

(AUC values) given progressively more data. For each week, the AUC value is obtained by using all clicker data up to that week (e.g., the AUC values at week 6 result from modeling clicker data from week 1 to week 6). The shaded area of Figure 6 describes each AUC's 95% confidence interval.

The results show that the AUC generally improves as we add more clicker data. This is unsurprising, as predictions are made in the context of additional data, and data collected later in the term is temporally closer to the exam than is earlier data. However, it is somewhat surprising that the prediction quality often plateaus before the end of the term. For example, in Architecture, the AUC only marginally improves over time; in CS1-Python, AUC increases until week 4, but the improvement slows at that point. Although this is likely deserving of further study, recent work on CS1 shows that the bulk of questions on a final exam require content from early in the term, but that only a small fraction of these questions require content from late in the term [35]. It may be that there is a similar effect in courses later in the curriculum; however, other possible explanations exist including: Learning Edge Momentum—that course material builds on itself making early material critical for later success [37], and stumbling blocks—that students encounter an early difficulty and later struggle to overcome it [3]. We believe further inquiry is needed to better explain this phenomenon.

Overall, the change in accuracy over time highlights a potential challenge for educators wishing to intervene for students predicted to do poorly. More time in the course often improves prediction accuracy but also reduces the time with which to alter a student's trajectory and increases the time during which misconceptions might compound. Week 3 was identified in prior work [35] as offering reasonable accuracy early in the term; our data supports this finding.

7 DISCUSSION

In this section, we discuss some implications of our modeling method and key avenues for future work.

7.1 Applicability of Prediction Methodology

This article proposes a prediction method based on easily obtainable student data that is capable of detecting at-risk students early in the term. Importantly, our method enables an instructor to make predictions before midterm exams are typically taken. Prior research suggests strong, positive correlations between the midterm exam and final exam in typical CS1 courses [34], and we suspect that this finding continues to hold throughout the CS curriculum. Accurate prediction prior to midterm exams is therefore advantageous: it opens the possibility for an intervention to chart a new course for the student prior to a weighty assessment.

Our prediction method uses only student clicker responses from lectures, so is relatively lightweight compared to methods that use other sources of data (e.g., presage factors) that may not be accessible to instructors. One concern is that our method requires the use of clickers and the PI pedagogy: What is one to do if they do not use PI? For two reasons, we suggest that this is not a significant problem. First, we note that Peer Instruction has in fact gained significant traction in computer science in recent years, with increased adoption by many instructors [31]. Second, we note that this increased reliance on PI has led to PI curricula being developed and made freely-available for many courses, including CS1, CS2, Operating Systems, Theory of Computation, and Data Structures [22]. This is advantageous here, as prior work suggests that PI questions can be productively used as quiz questions in an otherwise non-PI course [24]. For example, PI questions could be asked at the start or end of class using clickers, or as online exercises before or after class to eliminate clickers entirely. Our prediction method can then be applied to that data. Further work is required to replicate accuracy of predictions in non-PI contexts.

It is unrealistic to assume that a course will remain constant from term to term. Natural changes include variation among students, assignments, exams, scheduling, and ordering of content. Ideally, educational research methods should be adaptable to the realities of teaching, not the other way around. It is therefore important that a prediction method go beyond modeling a single course to modeling present offerings using data from past offerings. We have demonstrated in this article that our prediction method is well-suited to such realistic course contexts.

A powerful feature of our methodology is that it allows for instructors to set their own threshold for what they categorize as "at-risk." This enables instructors with substantial resources to potentially intervene for a larger number of students or for instructors with more limited resources to focus on a smaller group of at-risk students.

Last, we highlight our prediction performance across our two institutions and multiple CS courses. Rather than possibly being confined to a particular course context, we have demonstrated modeling facility across courses, instructors, and institutions.

7.2 Differences in Modeling Performance Across Courses

One of the larger challenges in evaluating our results is determining why one course offered better early prediction accuracy than another. There are a number of confounding factors that may all play a role in how predictable one course is versus another, including timing of important topics, student differences, instructor experience with PI, and how closely the final exam aligns with the PI questions. We suspect that the quality of the PI questions is a particularly important determinant of prediction quality. For example, some PI questions are likely better than others in terms of identifying students who hold misconceptions of core concepts, and it may be those questions that disproportionately improve prediction accuracy. We otherwise hesitate to posit theories on which factors are more or less critical in the courses we have studied here, though we suspect that many of these factors interact to influence the extent to which we can make accurate predictions.

7.3 Learning about Students from the Model

One of the promising elements of this work is that our models are successfully extracting some measure of student understanding from student responses to PI questions. Prior work has used Bayesian classification [35] and model variable importance [24] to reverse engineer the questions that were most meaningful for predicting student outcomes. Unfortunately, it is more difficult to extract meaning from advanced statistical and machine-learning models such as the model used in the present article. We are encouraged by ongoing work in machine learning to extract meaning from models, and believe advances in that area could lead to instructors extracting useful knowledge about their students and questions from the model results.

7.4 Suggestions for Future Modeling

There are three particular features of our modeling methodology that we encourage researchers to replicate and extend.

- Authentic real-world modeling: A considerable body of prior work uses a single term of data to both validate the model and make predictions. However, these predictions are necessarily made once the course is complete: they can inform modeling efforts but cannot be used to help struggling students. Moreover, they do not take into account the dynamic nature of courses as they shift from term to term. We suggest that, along with machine-learning improvements, the community focus on cross-term predictions. We have demonstrated in the present article the efficacy of present-day machine-learning algorithms, and it is time to use what we have learned to help students.
- Cross-institution: Within computer science, we are starting to see best-practice pedagogical materials being adopted at multiple institutions. See `peerinstruction4cs.org` for one example of this type of sharing. It is therefore increasingly likely that similar materials will come to be used at multiple schools. This bodes well for modeling student outcomes, as data from one institution may possibly be used to make predictions at others. That is, an institution may not require "starting from scratch," but can leverage prior-term predictions from other schools.
- Cross-curriculum: The computer science education research community has invested decades of research into understanding the progression of CS1 students. There is much less known about students in courses that follow CS1. We have demonstrated that our modeling does succeed in CS1 but also in several follow-on courses. There is therefore much to gain by studying courses that have not received as much research attention to this point.

8 CONCLUSION

Identifying at-risk students early in the term of a course offers benefits to both instructors and students. Instructors learn which students need more help, and students may receive that assistance in time to improve their course outcome. Reducing failure rates has institutional benefits by potentially reducing class sizes (fewer repeating students), improving retention of students in a major, and reducing time-to-graduation. In this work, we propose a prediction methodology using support vector machine binary classification to identify at-risk students early in the term. We demonstrate that this methodology can be effective at predicting students across different terms of the same course for courses at different institutions, by different instructors, and across the computer science curriculum. As the methodology requires only data naturally collected when teaching using Peer Instruction, the barrier for faculty to adopt and use this approach is relatively low.

Increased student enrollment means that instructors are unlikely to know where individual students stand. Coupled with that enrollment pressure, however, is the availability of powerful machine-learning algorithms and in-situ data generated by students as a byproduct of learning activities. The present article demonstrates a single modeling methodology that uses such data to automatically make predictions of student performance in a variety of computer science courses. Given the model accuracies in multiple real-world settings, ability to predict at-risk students early in the term, and lightweight demands on instructors to collect the data necessary to make such predictions, the clear next step for future work is to identify how to help these at-risk students.

ACKNOWLEDGMENTS

The authors sincerely thank the reviewers and the associate editor for their helpful suggestions on this work.

REFERENCES

[1] Yousef Mohamed Abdulrazzaq and Khalil Ibrahim Qayed. 2009. Could final year school grades suffice as a predictor for future performance? *Med. Teach.* 15, 2–3 (2009), 243–251.

[2] Alireza Ahadi, Arto Hellas, and Raymond Lister. 2017. A contingency table derived method for analyzing course data. *Trans. Comput. Educ.* 17, 3, Article 13 (2017), 13:1–13:19.

[3] Alireza Ahadi and Raymond Lister. 2013. Geek genes, prior knowledge, stumbling points and learning edge momentum: Parts of the one elephant? In *Proceedings of the 9th Annual International ACM Conference on International Computing Education Research.* 123–128.

[4] Alireza Ahadi, Raymond Lister, Heikki Haapala, and Arto Vihavainen. 2015. Exploring machine-learning methods to automatically identify students in need of assistance. In *Proceedings of the International Conference on Computing Education Research.* 121–130.

[5] A. Bandura. 1977. Self-efficacy: Toward a unifying theory of behavioral change. *Psychol. Rev.* 84, 2 (1977), 191–215.

[6] Susan Bergin and Ronan Reilly. 2006. Predicting introductory programming performance: A multi-institutional multivariate study. *Comput. Sci. Educ.* 16, 4 (2006), 303–323.

[7] Adam S. Carter, Christopher D. Hundhausen, and Olusola Adesope. 2017. Blending measures of programming and social behavior into predictive models of student achievement in early computing courses. *Trans. Comput. Educ.* 17, 3 (2017), 12:1–12:20.

[8] Adam S. Carter, Christopher D. Hundhausen, and Olusola O. Adesope. 2015. The normalized programming state model—Predicting student performance in computing courses based on programming behavior. In *Proceedings of the International Conference on Computing Education Research.* 141–150.

[9] Jennifer M. Case. 2015. A different route to reducing university drop-out rates. In *The Conversation.* Retrieved from https://theconversation.com/a-different-route-to-reducing-university-drop-out-rates-40406.

[10] Karo Castro-Wunsch, Alireza Ahadi, and Andrew Petersen. 2017. Evaluating neural networks as a method for identifying students in need of assistance. In *Proceedings of the Technical Symposium on Computer Science Education.* 111–116.

[11] Nihat Cengiz and Arban Uka. 2014. Prediction of student success using enrollment data. *Proceedings of the Workshops held at Educational Data Mining: Workshop Approaching Twenty Years of Knowledge Tracing.*

[12] A. T. Corbett and J. R. Anderson. 1994. Knowledge tracing - Modeling the acquisition of procedural knowledge. *User Model. User-Adapt. Interact.* 4, 4 (1994), 253–278.

[13] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Mach. Learn.* 20, 3 (1995), 273–297.

[14] CRA Enrollment Committee Institution Subgroup. 2017. Generation CS: Computer science undergraduate enrollments surge since 2006. Computing Research Association. Retrieved from http://cra.org/data/Generation-CS/.

[15] C. H. Crouch and E. Mazur. 2001. Peer instruction: Ten years of experience and results. *Amer. J. Phys.* 69, 9 (2001), 970–77.

[16] Michael de Raadt, Margaret Hamilton, Raymond Lister, Jodi Tutty, Bob Baker, Ilona Box, Quintin Cutts, Sally Fincher, John Hamer, Patricia Haden, Marian Petre, Anthony Robins, Simon, Ken Sutton, and Denise Tolhurst. 2005. Approaches to learning in computer programming students and their effect on success. *Res. Dev. Higher Educ.: Higher Educ. Chang. World* 28 (2005), 407–414.

[17] Edward M. Elias and Carl A. Lindsay. 1968. The Role of Intellective Variables in Achievement and Attrition of Associate Degree Students at the York Campus for the Years 1959 to 1963. Technical Report PSU -68 -7. Pennsylvania State University.

[18] J. A. Hanley and B. J. McNeil. 1982. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology* 143, 1 (1982), 29–36.

[19] Norhayati Ibrahim, Steven A. Freeman, and Mack C. Shelley. 2011. Identifying predictors of academic success for part-time students at polytechnic institutes in Malaysia. *Int. J. Adult Vocat. Educ. Technol.* 2, 4 (2011), 1–16.

[20] Matthew C. Jadud. 2006. Methods and tools for exploring novice compilation behaviour. In *Proceedings of the International Conference on Computing Education Research.* 73–84.

[21] David James and Clair Chilvers. 2001. Academic and non-academic predictors of success on the Nottingham undergraduate medical course 1970-1995. *Med. Educ.* 35, 11 (2001), 1056–1064.

[22] Cynthia Lee, Leo Porter, Beth Simon, and Daniel Zingaro. 2012. Peer instruction for computer science. Retrieved from http://www.peerinstruction4cs.org.

[23] Cynthia Bailey Lee, Saturnino Garcia, and Leo Porter. 2013. Can peer instruction be effective in upper-division computer science courses? *Trans. Comput. Educ.* 13, 3 (2013), 12:1–12:22.

[24] Soohyun Nam Liao, Daniel Zingaro, Michael A. Laurenzano, William G. Griswold, and Leo Porter. 2016. Lightweight, early identification of at-risk CS1 students. In *Proceedings of the International Conference on Computing Education Research.* 123–131.

[25] Adam Lucas. 2009. Using peer instruction and I-clickers to enhance student participation in calculus. *Prob. Resour. Issues Math. Undergrad. Studies* 19, 3 (2009), 219–231.

[26] National Center for Education Statistics. 2016. Total undergraduate fall enrollment in degree-granting postsecondary institutions, by attendance status, sex of student, and control and level of institution: Selected years, 1970 through 2026. National Center for Education Statistics. https://nces.ed.gov/programs/digest/d16/tables/dt16_303.70.asp.

[27] Charles G. Petersen and Trevor G. Howe. 1979. Predicting academic success in introduction to computers. *Assoc. Educ. Data Syst.* 12, 4 (1979), 182–191.

[28] Scott Pilzer. 2001. Peer instruction in physics and mathematics. *Prob. Resour. Issues Math. Undergrad. Studies* 11, 2 (2001), 185–192.

[29] Leo Porter, Cynthia Bailey Lee, and Beth Simon. 2013. Halving fail rates using peer instruction: A study of four computer science courses. In *Proceedings of the Technical Symposium on Computer Science Education.* 177–182.

[30] L. Porter, C. Bailey-Lee, B. Simon, Q. Cutts, and D. Zingaro. 2011. Experience report: A multi-classroom report on the value of peer instruction. In *Proceedings of the Annual Joint Conference on Innovation and Technology in Computer Science Education.* 138–142.

[31] Leo Porter, Dennis Bouvier, Quintin Cutts, Scott Grissom, Cynthia Lee, Robert McCartney, Daniel Zingaro, and Beth Simon. 2016. A multi-institutional study of peer instruction in introductory computing. In *Proceedings of the Technical Symposium on Computer Science Education.* 358–363.

[32] Leo Porter, Saturnino Garcia, John Glick, Andrew Matusiewicz, and Cynthia Taylor. 2013. Peer instruction in computer science at small liberal arts colleges. In *Proceedings of the Annual Joint Conference on Innovation and Technology in Computer Science Education.* 129–134.

[33] Leo Porter and Beth Simon. 2013. Retaining nearly one-third more majors with a trio of instructional best practices in CS1. In *Proceedings of the Technical Symposium on Computer Science Education.* 165–170.

[34] Leo Porter and Daniel Zingaro. 2014. Importance of early performance in CS1: Two conflicting assessment stories. In *Proceedings of the Technical Symposium on Computer Science Education.* 295–300.

[35] Leo Porter, Daniel Zingaro, and Raymond Lister. 2014. Predicting student success using fine grain clicker data. In *Proceedings of the International Conference on Computing Education Research.* 51–58.

[36] D. M. Powers. 2011. Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation. *J. Mach. Learn. Technol.* 2, 1 (2011), 37–63.

[37] Anthony Robins. 2010. Leaning edge momentum: A new account of outcomes. In *Computer Science Education, 20(1).* 37–71.

[38] John E. Roueche. 1967. Research studies of the junior college dropout. *Amer. Assoc. Junior Coll.* (1967), 1–5.

[39] Philip M. Sadler and Robert H. Tai. 2007. Advanced placement exam scores as a predictor of performance in introductory college biology, chemistry and physics courses. *Sci. Educat.* 16, 2 (2007), 1–19.

[40] William E. Cohen Sadler and Frederic L. Kockesen Levent. 1997. Factors affecting retention behavior: A model to predict at-risk students. In *Proceedings of the Association for Institutional Research Annual Forum.*

[41] Vicki L. Sauter. 1986. Predicting computer programming skill. *Computers & Education* 10, 2 (1986), 299–302.

[42] Sami Shaban and Michelle McLean. 2011. Predicting performance at medical school: Can we identify at-risk students? *Adv. Med. Educ. Pract.* 2 (2011), 139–148.

[43] Karedn Shakerdge. 2016. High failure rates spur universities to overhaul math class. In *The Hechinger Report*. Retrieved from http://hechingerreport.org/high-failure-rates-spur-universities-overhaul-math-class/.

[44] Shahireh Sharif, Larry Gifford, Gareth A. Morris, and Jill Barber. 2003. Can we predict student success (and reduce student failure)? *Pharm. Educ.* 3 (2003), 1–10.

[45] B. Simon, M. Kohanfars, J. Lee, K. Tamayo, and Q. Cutts. 2010. Experience report: Peer instruction in introductory computing. In *Proceedings of the Technical Symposium on Computer Science Education*. 341–345.

[46] Beth Simon, Julian Parris, and Jaime Spacco. 2013. How we teach impacts student learning: Peer instruction vs. lecture in CS0. In *Proceedings of the Technical Symposium on Computer Science Education*. 41–46.

[47] Larry D. Singell and Glen R. Waddell. 2010. Modeling retention at a large public university: Can at-risk students be identified early enough to treat? *Res. Higher Educ.* 51, 6 (2010), 546–572.

[48] Michelle K. Smith, William B. Wood, Wendy K. Adams, Carl E. Wieman, Jennifer K. Knight, Nancy Guild, and Tin Tin Su. 2009. Why peer discussion improves student performance on in-class concept questions. *Science* 323, 5910 (2009), 122–124.

[49] Alex Smola, Kurt Hornik, Achim Zeileis, and Alexandros Karatzoglou. 2003. Kernel-based machine-learning lab. Retrieved from https://www.rdocumentation.org/packages/kernlab/versions/0.9-25.

[50] Louise Tickle. 2015. How universities are using data to stop students dropping out. In *The Guardian*. Retrieved from https://www.theguardian.com/guardian-professional/.

[51] Bruno Trstenjak and Dzenana Donko. 2014. Determining the impact of demographic features in predicting student success in Croatia. In *Proceedings of the International Convention on Information and Communication Technology, Electronics, and Microelectronics*. 1222–1227.

[52] Christopher Watson, Frederick W. B. Li, and Jamie L. Godwin. 2013. Predicting performance in an introductory programming course by logging and analyzing student programming behavior. In *Proceedings of the International Conference on Advanced Learning Technologies*. 319–323.

[53] Brenda Cantwell Wilson and Sharon Shrock. 2001. Contributing to success in an introductory computer science course—A study of twelve factors. In *Proceedings of the Technical Symposium on Computer Science Education*. 184–188.

[54] Annika Wolff, Zdenek Zdráhal, Drahomira Herrmannova, and Petr Knoth. 2014. Predicting student performance from combined data sources. *Educ. Data Mining* 524 (2014), 175–202.

[55] Ping Zhang, Lin Ding, and Eric Mazur. 2017. Peer instruction in introductory physics: A method to bring about positive changes in students' attitudes and beliefs. *Phys. Rev. Phys. Educ. Res.* 113, 1 (2017), 10.

Received May 2018; revised August 2018; accepted August 2018