UNIVERSITY OF CALIFORNIA SAN DIEGO

Early Identification of At-risk Students and Understanding Their Behaviors

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science (Computer Engineering)

by

Soohyun Nam Liao

Committee in charge:

Professor William G. Griswold, Co-Chair
Professor Leo Porter, Co-Chair
Professor Gedeon Deák
Professor Scott R. Klemmer
Professor Julian McAuley

2019

The dissertation of Soohyun Nam Liao is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____
Co-Chair

_____
Co-Chair

University of California San Diego

2019

DEDICATION

To Sean, the father of our beautiful children Nile and Willy

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to give sincere gratitude to both of my advisors, Bill and Leo. Without them I would have not made until the end of this journey. They trusted my potential as a researcher from the beginning, supported me whenever I needed them for whatever reasons, and guided me to grow as well-prepared faculty. Their emotional and financial support allowed me to start a family during my Ph.D. with ease as well.

I also thank my family who remotely supported me from South Korea, and my husband Sean who was always beside me and helped me worry about nothing but my Ph.D. research. I was able to keep up and stay positive thanks to their encouragement.

Lastly, I deeply appreciate all the co-authors of the publications I referred to in this dissertation - Daniel Zingaro, Christine Alvarado, Kevin Thai, Sander Valstar, and Michael A. Laurenzano. Particularly, I send special thanks to Dan who gave me a good introduction about the field of computing education and collaborated with me throughout my active research years in computing education.

Chapter 1, 2, 3, 8 and 9, in part, are reprints of the materials as they appears in four publications: 1) Proceedings of the Conference on International Computing Education Research. S. N. Liao, D. Zingaro, M. A. Laurenzano, W. G. Griswold, and L. Porter, ACM, 2016; 2) Proceedings of the 50th ACM Technical Symposium on Computer Science Education. S. N. Liao, D. Zingaro, C. Alvarado, W. G. Griswold, and L. Porter, ACM, 2019; 3) ACM Transactions in Computing Education. S. N. Liao, D. Zingaro, K. Thai, C. Alvarado, W. G. Griswold, and L. Porter, ACM, 2019; and 4) Proceedings of the 24th ACM Annual ACM Conference on Innovation and Technology in Computer Science Education. S. N. Liao, S. Valstar, K. Thai, C. Alvarado, D. Zingaro, W. G. Griswold, and L. Porter, ACM, 2019. The dissertation/thesis author was the primary investigator and author of these papers.

Chapter 4, in full, is a reprint of the materials as they appears in Proceedings of the Conference on International Computing Education Research. S. N. Liao, D. Zingaro, M. A.

Laurenzano, W. G. Griswold, and L. Porter, ACM, 2016. The dissertation/thesis author was the primary investigator and author of this paper.

Chapter 5, in full, is a reprint of the materials as they appears in ACM Transactions in Computing Education. S. N. Liao, D. Zingaro, K. Thai, C. Alvarado, W. G. Griswold, and L. Porter, ACM, 2019. The dissertation/thesis author was the primary investigator and author of this paper.

Chapter 6, in full, is a reprint of the materials as they appears in Proceedings of the 50th ACM Technical Symposium on Computer Science Education. S. N. Liao, D. Zingaro, C. Alvarado, W. G. Griswold, and L. Porter, ACM, 2019. The dissertation/thesis author was the primary investigator and author of this paper.

Chapter 7, in part, has been submitted for publication of the material as it may appear in Proceedings of the 24th ACM Annual ACM Conference on Innovation and Technology in Computer Science Education. S. N. Liao, S. Valstar, K. Thai, C. Alvarado, D. Zingaro, W. G. Griswold, and L. Porter, ACM, 2019. The dissertation/thesis author was the primary investigator and author of this paper.

VITA

2011	B. S. in Electronics and Communications Enginering *summa cum laude*, Hanyang University, Seoul, South Korea
2014-2018	Graduate Teaching Assistant, University of California, San Diego
2017	Associate-in Instructor, University of California, San Diego
2019	Ph. D. in Computer Science (Computer Engineering), University of California, San Diego

ABSTRACT OF THE DISSERTATION

Early Identification of At-risk Students and Understanding Their Behaviors

by

Soohyun Nam Liao

Doctor of Philosophy in Computer Science (Computer Engineering)

University of California San Diego, 2019

Professor William G. Griswold, Co-Chair
Professor Leo Porter, Co-Chair

As enrollments and class sizes in postsecondary institutions have increased, instructors have sought automated means to identify students who are at risk of failing a course. This identification must be performed early enough in the term to allow instructors to assist those students before they fall irreparably behind. In this sense, this dissertation proposes early identification methods of at-risk students and characterizes behaviors of such students.

The first part of this work describes a modeling methodology that predicts student final exam scores in the third week of the term by using different sets of easy-to-collect data including student clicker responses, prerequisite course grades, online quiz scores, and assignment grades.

The work uses different machine learning techniques, trained on one term of a course, to predict outcomes in subsequent terms. This allowed making predictions across terms in a natural setting (different final exams, minor changes to course content). We applied this modeling technique to five different courses across the computer science curriculum, taught by three different instructors at two different institutions. The results show consistent performance across multiple courses in a curriculum, and across multiple institutions. Also, prerequisite course grades and clicker responses are more predictive than online quiz and assignment grades.

The second part of the work is to understand what factors contribute to those students' difficulties. If we were able to better understand the characteristics of such students, we may be better able to help those students. This work examines the characteristics of lower- and higher-performing students through interviews with students from an introductory computing class. We identify a number of relevant areas of student behavior including how they approach their exam studies, how they approach completing programming assignments, whether they sought help after identifying misunderstandings, how and from whom they sought help, and how they reflected on assignments after submitting them. Particular behaviors within each area are coded and differences between groups of students are identified.

This dissertation ends with stating some future work of automating the identification process for general public and crafting effective student intervention frameworks.

Chapter 1

Introduction

University administrators and faculty continue to be concerned by time-to-graduation and student retention. 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. Moreover, there are affective costs to failure, such as reduced self-efficacy [7]. 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 [12, 76, 86].

Recently, lowering failure rates incurs an additional challenge: university enrollments and class sizes are on the rise. According to the National Center for Educational Statistics [50], undergraduate enrollment in 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 dissertation, has seen even greater rates of increase in enrollment beginning in 2006 [18]. 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 postsecondary schools.

Research evidence has shown that in a large class, students are not as satisfied with the learning environment as in a small class [41]. This is probably because it is more difficult for instructors to keep track of individual students' progress, and struggling students are more likely to slip through the cracks.

Therefore, it is important to identify at-risk students early in the term. This would allow instructors to strategically assign existing instructional resources to the identified students and help them succeed. This approach requires rigorous investigation of students in order to properly identify at-risk students, understand why the identified students are at-risk, and provide effective intervention. Therefore in this dissertation, we demonstrate different methods of how to identify at-risk students early in the term and explore different behavioral characteristics of at-risk students.

Early identification of at-risk students could be highly valuable for instructors and students alike. Instructors could explore possible intervention strategies to help struggling students, and students who are made aware that they are likely struggling may be spurred to change study habits or seek additional assistance. Numerous prior literature in a variety of disciplines investigated different methods of early identification for decades. Prior work can be grouped based on at what scale predictions were made, how models were generated and evaluated, and what types of data were used.

Identifying at-risk students has attracted researchers' attention in different parts of institutions. They are institution-wide instructional researchers [14, 26, 72, 82, 87], program-wide instructional researchers [3, 34, 36, 75, 77], and individual instructors of different programs [4, 5, 8, 11, 13, 35, 54, 73, 89, 90]. The former two types of researchers aimed to prevent student dropouts of their institutions or programs, whereas most of the individual instructors aimed to ensure that identified at-risk students succeed in their courses.

Methods to create and evaluate models are also varied. Some literature used descriptive

modeling [3, 11, 26, 34, 35, 36, 72, 73, 75, 77, 82, 89, 90] to understand the relationships between independent and dependent variables. The others used predictive modeling [4, 5, 8, 13, 14, 54, 87], to actually make predictions on another set of students. However, the both approaches in general are not applicable to an ongoing course, because the dependent variable values (i.e. student learning outcome) are not available until the course is complete. Thus, this dissertation uses a cross-term approach, which creates model using data from one cohort in the past and evaluates the generated model on data from a subsequent cohort. By doing so, the model evaluation can be done on ongoing courses early in the term when their dependent variables are not ready. Chapter 4, 5, and 6 in this dissertation all used this cross-term approach.

For the types of data used, research from the 1970s through the 1990s focused primarily on static personal features (GPA, gender, etc.), often called presage variables [3, 8, 14, 26, 34, 36, 54, 72, 73, 75, 77, 82, 87, 90]. Recent work has begun using dynamic data, often called in-progress variables [4, 5, 13, 65], such as assignments and in-class performance to identify these students. In-progress data is sometimes collected naturally as a part of technology-based teaching practices employed in large classes. Although these practices are originally used to improve student outcomes, they generated a rich data set that can be used to identify struggling students.

This dissertation focuses specifically on one such practice called Peer Instruction [20] (PI). PI is an active-learning pedagogy used in many disciplines [20, 44, 55, 59, 80, 83, 93] that revolves around students answering and discussing meaningful conceptual questions with their peers and the instructor. Research evidence has shown that PI improves students learning and their attitudes about learning [20, 44, 83, 93]. Often, PI 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.

Porter et al. provided promising results that easy-to-obtain, in-class clicker data is correlated with final exam scores; however, that analysis was limited to a single term [65]. Next,

Ahadi et al. demonstrated that machine learning models based on a combination of static and assignment submission data could be used to predict students in the bottom half of final exam scores; however, that analysis was done in a CS1 course with a very large number of assignments in the first few weeks [5]. That latter work focused on modeling a single term, although the authors did show the potential to model student outcomes using the data from a previous course offering and applies the model to a subsequent offering despite minor changes to course content (i.e. cross-term identification).

Thus, the first identification method in this dissertation (Chapter 4) builds upon the strengths of both of these works by using easy-to-obtain, in-class clicker results to predict student outcomes across terms. By doing so, we provide cross-term modeling approach whose data is easy to generate (requiring little course change), and where the time-consuming collection of sensitive student demographics or background information is not required. A linear regression model was created using a PI CS1 course in Python. The model predicts a final exam score using in-class clicker question data collected from the first three weeks of a term then decides whether a student is "at-risk" or "not-at-risk" of failing the course.

This model accurately predicts 22% of students as at-risk and 48% as not-at-risk. Only 17% of students are miscategorized as not needing assistance when assistance was needed. In addition, recognizing that not all CS1 courses employ PI, we explore the use of the same questions as quiz questions rather than in-class clicker questions. We find that model accuracy declines only slightly, suggesting that our approach may be applicable to lecture-based courses as well.

The follow-up identification method (Chapter 5) builds on the study described in Chapter 4. The goal for this identification technique is that it is more robust than the previous model in the sense that it works on courses across the curriculum and across institutions. In this study, we used support vector machines [17] instead of linear regression to perform an one-step binary classification of whether students are expected to be at-risk or not-at-risk. This modeling method was applied to data from five different courses at two different institutions. For all courses, our

modeling method consistently correctly identified at least 62% of at-risk students when using an instructor-determined threshold of the bottom 40% of the class. To the best of our knowledge, this 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.

One common limitation of previous prediction studies, including the ones in Chapter 4 and 5, is that they focus on single data sources, rather than the ways in which multiple sources might work in concert. Also, many of those studies largely focus on introductory programming courses, leaving a gap in our understanding of the factors that might be predictive for later courses.

Thus in Chapter 6, we expand our data sources to include prerequisite computer science courses (when applicable), assignments, online quizzes, and clicker data from five different CS courses (both lower- and upper-division) across two separate institutions. In each case, we examine the value of each of these data sources in building a predictive model, where the model is trained on data from one term and then applied to students in a subsequent term. We find, as one might expect, that for upper-division courses, grades in prerequisite courses are particularly strong predictors. When prerequisite CS grades are available adding clicker data improves the model; when prerequisite CS grades are unavailable clicker data was the strongest predictor.

By contrast, assignments and online quizzes improve the model only marginally, compared to prerequisite courses and clicker data. In concert, prerequisite grades and clicker data provide particularly accurate predictions. As such, instructors wishing to identify low-performing students early in the term may wish to prioritize the collection of these data sources.

Despite such achievements above, identifying which students are likely to be at-risk is only half the battle: we must also endeavor to help these students. Unfortunately, designing effective interventions is challenging, and the literature on interventions has yielded few successes [15, 24]. Interventions that require extra work, such as supplemental help sessions or the opportunity to redo prior work, usually fail to reach at-risk students, who have too much on their plate

already [37, 48]; more general "learning how to learn" interventions are too far removed from the specific course material to be effective [24]. Moreover, one-size-fits-all interventions may fail in CS because we have reason to suspect that students in CS classes may succeed for different reasons than those in other disciplines [95].

A deeper understanding of the behaviors employed by at-risk and not-at-risk students in introductory computing would aid in developing effective interventions. To gain a broad understanding of the gamut of student behaviors, we conducted a series of semi-structured interviews with students in an introductory CS course at a large public university and qualitatively analyzed interview transcripts (Chapter 7). Our choice of qualitative research methods is informed by our goal of exploring students' individual contexts and identifying new variables of interest [19].

In these interviews, we inquired about student behaviors in the course, including how they prepare for exams and solve programming assignments. To analyze these interviews, we used an open-coding approach [28, 47] to identify and categorize student behaviors. A natural division appeared between behaviors associated with preparing for exams and behaviors associated with completing programming assignments. We believe some of the behaviors within these two categories are specific to learning computer science, whereas others fall under general study skills.

At the end of the course, with the knowledge of which students ultimately performed poorly or well, we analyzed the codes from interview transcripts and found that for some categories of behavior, the behaviors of at-risk and not-at-risk students varied in distinctive ways. To study the prevalence of these behaviours, a student survey was designed and administered with the results often matching with the interview findings.

Crafting effective interventions is left for future work, geared at encouraging positive behaviors common to not-at-risk students and/or discouraging behaviors common to at-risk students. We believe this dissertation paves the way for future intervention by providing a solution on how to identify at-risk students and implications on what kind of behaviors hinder their success

in a course.

This chapter, in part, is a reprint of the materials as they appears in four publications: 1) Proceedings of the Conference on International Computing Education Research. S. N. Liao, D. Zingaro, M. A. Laurenzano, W. G. Griswold, and L. Porter, ACM, 2016; 2) Proceedings of the 50th ACM Technical Symposium on Computer Science Education. S. N. Liao, D. Zingaro, C. Alvarado, W. G. Griswold, and L. Porter, ACM, 2019; 3) ACM Transactions in Computing Education. S. N. Liao, D. Zingaro, K. Thai, C. Alvarado, W. G. Griswold, and L. Porter, ACM, 2019; and 4) Proceedings of the 24th ACM Annual ACM Conference on Innovation and Technology in Computer Science Education. S. N. Liao, S. Valstar, K. Thai, C. Alvarado, D. Zingaro, W. G. Griswold, and L. Porter, ACM, 2019. The dissertation/thesis author was the primary investigator and author of these papers.

Chapter 2

Background

2.1 Identification of At-Risk Students

Identification of student outcomes at universities and colleges have been investigated at all levels: early 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 time scales and outcomes of interest, different predictors have been used at each level. Modeling techniques and how they are evaluated also vary widely. Table 2.1 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 (called presage factors), such as gender, age, and SAT scores [72]. Additional factors have been added in recent research including student attitudes [23], student behaviors during the term [4, 10, 92], and the use of in-class formative assessment data [42]. However, the vast majority of previous modeling techniques are either (1) heavyweight, (2) applied to a single course, (3) applied at a single institution, or (4) modeled and

tested with a single term of data.

2.1.1 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 GPA or math background. On the other hand, 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 2.1 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 studies evaluate their success and then surveys a variety of related projects. We then describe how our present study relates to prior work in predicting outcomes in computing.

Table 2.1: Synthesis of Related Work (⋆: Chapter 3/4, ⋆⋆: Chapter 5)

Ref (1)	Study Focus (2)	Year (3)	Presage Variables		In-progress Variables		Pred.† (8)	Replicated (9)	Cross-term (10)
			Performance & Background (4)	Behavior & Attitude (5)	Performance (6)	Behavior & Attitude (7)			
[26]	Institution	1968	✓		✓			✓	
[72]	Institution	1997	✓		✓				
[82]	Institution	2010	✓		✓				
[14]	Institution	2014	✓				✓		
[87]	Institution	2014	✓				✓		
[36]	Program	2001	✓						
[77]	Program	2003	✓						
[3]	Program	2009	✓						
[34]	Program	2011	✓						
[75]	Program	2011	✓		✓				
[54]	Course	1979	✓	✓			✓		
[73]	Course	1986	✓						
[90]	Course	2001	✓	✓					
[8]	Course	2006	✓	✓		✓	✓	✓	
[35]	Course	2006				✓			
[89]	Course	2013				✓			
[11]	Course	2015				✓			
[5]	Course	2015			✓	✓	✓		
[4]	Course	2017			✓	✓	✓		
[13]	Course	2017			✓	✓	✓		✓
[16]	Module	1994			✓				
⋆	**Course**	**2016/9**			✓		✓	✓	✓
⋆⋆	**Course**	**2019**	✓		✓		✓	✓	✓

†Checkmark indicates predictive modeling, unmarked indicates descriptive modeling.

2.1.2 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.

Some studies have attempted to demonstrate the robustness of their techniques by replicating their results 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 2.1 indicates whether prior work attempted replication of their results.

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 2.1 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 approach 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 method 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 2.1 distinguishes between these predictive modeling methods.

Lastly, there is considerable variation in the metric used to evaluate the model, making it difficult to compare one model to another. 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 an ROC curve (AUC) [67]. 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).

2.1.3 Institution- and Program-Level Modeling

The top rows of Table 2.1 summarize some of the work discussed in this section. 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 [70]. 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 [26]. As a result, recent work often includes in-progress variables in addition to presage variables [72, 82].

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 2.1 at the institution and program level tested the models on future cohorts. As examples of (1), some researchers [72, 82] assess the quality of their regression models using the models themselves. In contrast, other authors [14, 87] generate predictive models, providing prediction results. These works test their models on a separate test dataset from the same cohort as in (2), which may cause the models to overfit to a single 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 [36, 77], but it is not reported whether these variables can be used to predict student outcomes with any degree of accuracy.

2.1.4 Course-Level Modeling

Various disciplines investigated predicting student outcomes at a course-level. For example, Sadler and Tai found evidence that student demographic information is related to student performance in introductory courses of biology, chemistry, and physics [71]. Moreover, in library and information science, both types of presage data - performance & background and behavior & attitude - are predictive to students learning outcome on multiple online courses in the program [52]. While these two work used presage data, prior work in psychology utilized in-progress data - page hits, discussion posts, discussion reads on a learning management system - to model student performance [68].

The vast majority of the course-level work is from computing discipline. In computing, which is the focus of this paper, 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 [54, 73]. In-progress data in the form of attitudinal surveys have also been used to predict exam scores [90]. Both presage and in-progress variables have been combined effectively using regression analysis to predict student programming performance [8]. Lastly, in-progress variables such as programming progress have been used to aid intelligent tutoring systems as well [16].

A primary source of data for prediction using in-progress variables is code snapshots of student programming activity. To gather these data, researchers use an instrumented programming environment that regularly records the student's current program in a log file. The goal is to assess student knowledge by comparing two successive code snapshots. Early methods examined compiler errors [35, 89], while later work also considered runtime errors [11]. 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 [4, 5]. 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 is certainly not lightweight (i.e. data collection requires additional effort on top of regular lecture materials) and could be discouraging to students. In addition, many machine-learning models were tested and the best one chosen; it is unclear how one would make such a decision in a different context without producing all of the models.

Similarly, Castro-Wunsch et al. examined many machine learning models, and applied the trained model to a test dataset from a subsequent term [13]. 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 [13]. However, using the method of Castro-Wunsch et al. is again complicated by having to run several models and selectively choosing the best-performing model. Finally, echoing another concern from Ahadi et al., Castro-Wunsch et al. trained their models using data from students completing 30 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 is again not lightweight and may not apply to other types of CS courses.

The extant research push of course-level modeling in computing is largely focused on making predictions in CS1 courses, with very little action in other courses [22]. Also, students taking CS1 courses are very likely in their first year of study, so such research cannot include student grades in prerequisite courses. Some work suggests that grades in prerequisite courses are useful for predicting student progression through a CS degree program [6], but that work does not consider dynamic predictors of performance.

Considering the research in this subsection in its totality, what is missing is (1) a robust and lightweight technique capable of making predictions for multiple courses across a curriculum

and at multiple institutions, (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), and (3) efficacy of prerequisite grades and dynamic predictors working together to predict outcomes in both introductory and more advanced courses. These are the main contributions of the present paper, for the subject of computer science.

2.2 Investigating Student Behaviors

Researchers have qualitatively studied the factors that motivate students to persist or drop out of introductory computing courses. Those studies found that time constraints, low prioritization of CS courses, ineffective study skills, difficulty, and motivation were characteristics often associated with students who drop CS1 [53, 88]. These prior studies uncover some of the same behaviors that we identify, but they focus only on those students who drop CS1 or leave computer science. In addition, there is a strong quantitative tradition that investigates student persistence and progress in introductory CS courses, focusing on factors such as motivation [78], achievement goals [95], measures of programming behavior [25, 35, 89, 11], and measures of students' social behaviors [9]. While these studies have informed techniques for identifying at-risk students, they do not give insight into the particular antecedents of these behaviours.

Finally, there are numerous qualitative studies that report on student behaviours and attitudes, such as study strategies of students when they are stuck [46], debugging or problem-solving strategies during programming [32, 49], students' perceptions of their ability [40], and students' perceptions of their behaviors [79]. That work, however, focuses on class-level data, not the particular experiences of individual students. In this sense, this dissertation is the first work, up to our knowledge, conducting qualitative studies on an CS course to understand why some students become at risk of failing a course.

2.3 Acknowledgements

This chapter, in part, is a reprint of the materials as they appears in four publications: 1) Proceedings of the Conference on International Computing Education Research. S. N. Liao, D. Zingaro, M. A. Laurenzano, W. G. Griswold, and L. Porter, ACM, 2016; 2) Proceedings of the 50th ACM Technical Symposium on Computer Science Education. S. N. Liao, D. Zingaro, C. Alvarado, W. G. Griswold, and L. Porter, ACM, 2019; 3) ACM Transactions in Computing Education. S. N. Liao, D. Zingaro, K. Thai, C. Alvarado, W. G. Griswold, and L. Porter, ACM, 2019; and 4) Proceedings of the 24th ACM Annual ACM Conference on Innovation and Technology in Computer Science Education. S. N. Liao, S. Valstar, K. Thai, C. Alvarado, D. Zingaro, W. G. Griswold, and L. Porter, ACM, 2019. The dissertation/thesis author was the primary investigator and author of these papers.

Chapter 3

Study Context

This chapter describes the context of datasets used in this dissertation. We state how those courses fit into the computer science curriculum and which pedagogical practice was adopted to those courses. Also, this chapter provides some key terminologies that are used throughout the paper.

3.1 Five Courses Used in the Present Paper

Table 3.1 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

Table 3.1: 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

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. While many computer science courses concentrate on how software works and how to design better software, this course provides students with an understanding of the hardware on which that software executes.

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

Having different course levels allows us to examine our modeling method with different student population. A variety of curricular material means different forms of assessments as well. For example, programming courses tend to have more programming assignments that takes up a good percentage of the final grades, while the architecture course only had written assignments. To our knowledge, no prior work has predicted at-risk students across such a broad spectrum of courses.

3.2 Peer Instruction

All the courses used in this dissertation adopted an student-centered active learning method named PI [20]. It was invented in the 1990s for introductory Physics courses, and was later adopted for computer science. For clarity, we outline the core components of a PI class, the different votes we collect from students, and possible interpretations of these votes.

In a PI course, instruction is centered around a series of questions that students solve in class. Although the number of questions varies, the courses studied here had 3-5 questions per lecture. As part of the process, students are asked to select their answer, often using in-class electronic response systems (clickers). Through a participation grade, students are rewarded for attending class and participating in PI questions, not on providing correct responses.

An individual PI cycle follows a well defined process:

- **Individual Vote**: Students are shown the question and asked to solve it individually. They then record their answer using the clicker, the results of which are transmitted to the instructor. As an approximation, this vote can be viewed as student understanding prior to in-class instruction.

- **Group Vote**: Students are then asked to discuss the problem in small groups and come to a consensus. They then respond again using the clicker. As an approximation, this vote can be viewed as student understanding after discussion. While it is possible that students could

vote the same as their peers without understanding the chosen response, prior research suggests that this is not a major concern [58].

- **Classwide Discussion**: Based on student responses, the instructor leads a classwide discussion about the question, aiming to both engage students in a discussion of why particular response choices were made and clarify to the class why certain responses are correct or incorrect.

- **(Optional) Isomorphic Vote**: For some questions deemed to be particularly important for learning, the instructor asks a follow-on question after the classwide discussion. This isomorphic question is a different question on the same concept just discussed. The students respond to this question individually. As an approximation, the isomorphic vote can be viewed as student understanding at the conclusion of the PI process on that concept.

The efficacy of PI has been shown in various disciplines including physics [20, 93], biology [83], mathematics [44, 55], and computer science [56, 57, 59, 62, 63, 80, 81]. These studies discovered that PI improves students learning and their attitudes about learning [20, 93], enhances student participation and understanding [44], and fosters learning even when none of the discussion group members knows the correct answer [83].

In computing, PI has gained significant traction in recent years, with increased adoption by many instructors [59]. 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 [21, 39, 57]. PI in computing has been shown to enhance student learning [81], lowers course failure rates [56], and increases the retention rate of students in the major [63].

By virtue of using PI coupled with clickers, students naturally generate clicker data in each lecture. In addition, research evidence has shown that student clicker correctness is highly correlated with their final exam performance [66]. This dissertation leverages this automatically-

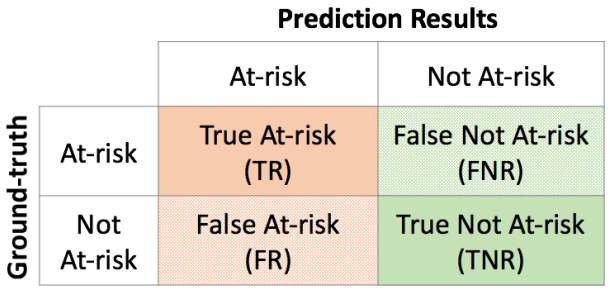


Figure 3.1: Classification Categories

collected data to make early identification.

3.3 Terminologies

This subsection defines some commonly-used terminologies to prevent confusion for the readers.

- **Early Identification**: Similar to prior work, we produce predictions using only data collected during the first 3 weeks of each term [66]. By making predictions this early in the term, by the end of week 3, the belief is that potential interventions may be possible even before the first midterm exam. The datasets used in this dissertation also had the first midterm in week 4 the earliest. Interchangeable with **Early Prediction**.

- **Predicting Student Outcomes**: It is used when identifying both At-Risk and Not At-risk students.

- **At-risk and Not At-risk Students**: At-risk (or Not At-risk) students means those students who are (or are not) at risk of failing the final or the course. Classifying students into two groups allows us to perform binary classification, where our model tries to predict whether a student in the test dataset will score in the top 60% or the bottom 40% of the final exam. Recent work uses a 50% cutoff [5, 13], 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. This flexibility allows instructors to choose thresholds based on how much resources they have.

- **Classifying Students**: Any binary decision model can produce two types of errors. It can incorrectly classify an at-risk 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 3.1, which shows the confusion matrix for the possible outcomes of the classification vs. the ground truth of each sample.

  - **True Not At-risk.** Our model correctly predicts that a student will be in the top 60% on the final exam.
  - **True At-risk** Our model correctly predicts that a student will be in the bottom 40% on the final exam.
  - **False Not At-risk.** Our model predicts that a student will be in the top 60% on the final exam, but they end up in the bottom 40%.
  - **False At-risk.** Our model predicts that a student will be in the bottom 40% on the final exam, but they end up in the top 60%.

3.4 Acknowledgements

This chapter, in part, is a reprint of the materials as they appears in four publications: 1) Proceedings of the Conference on International Computing Education Research. S. N. Liao, D. Zingaro, M. A. Laurenzano, W. G. Griswold, and L. Porter, ACM, 2016; 2) Proceedings of the 50th ACM Technical Symposium on Computer Science Education. S. N. Liao, D. Zingaro, C. Alvarado, W. G. Griswold, and L. Porter, ACM, 2019; 3) ACM Transactions in Computing

Education. S. N. Liao, D. Zingaro, K. Thai, C. Alvarado, W. G. Griswold, and L. Porter, ACM, 2019; and 4) Proceedings of the 24th ACM Annual ACM Conference on Innovation and Technology in Computer Science Education. S. N. Liao, S. Valstar, K. Thai, C. Alvarado, D. Zingaro, W. G. Griswold, and L. Porter, ACM, 2019. The dissertation/thesis author was the primary investigator and author of these papers.

Chapter 4

Early and Lightweight Identification Technique in CS1

As more instructors adopted technology-based teaching practices, more student data became available for analysis. PI is one of well-studied technology-based teaching practices. It collects students' clicker responses to multiple-choice questions posed by instructors in class. Thus, the work in this chapter utilized the clicker data from the first three weeks of a term to allow early identification of at-risk students. This method is also lightweight since it does not require any extra burden for instructors to collect the data, other than practicing typical PI. This work is the first which utilizes student clicker responses to predict student outcomes.

4.1 Method

4.1.1 Data Collection

The data are collected from a CS1-Python at a large North American research university during two consecutive fall terms ($n = 171$ and $n = 142$, respectively). Each offering was 12 weeks long and had three weekly 50-minute lectures. Student work included two term tests,

weekly pair-programming labs, two large programming assignments, and a final exam. The second term, but not the first, also included short, weekly programming exercises. Both terms were taught by the same instructor using the PI pedagogy [58, 60, 94]. As further explained below, PI focuses on students discussing conceptual questions and responding with electronic clickers. The PI materials used in this study are based on those of a prior study [65].

The course content and lecture materials between the two terms are quite comparable as they were taught by the same instructor. We examined the PI clicker questions used in each course offering and found that 88% of all clicker questions appeared in both terms. Student responses to early matched questions (first three weeks), along with final exam scores, were used as the training and test data. Throughout the analysis, the former term is used as training data and the latter as test data.

4.1.2 Data Analysis and Modeling

In this section, we describe the process for creating and applying the model. The steps include: partitioning the data, preprocessing the data, using Principal Components Analysis to reduce data dimensionality, building the model, using the validation set to determine a classification threshold, and applying that model and threshold on the test set. Details are provided to encourage replication.

Data Set Partitioning

The data from the first term was split into two subsets: training set and validation set. The validation set is used to optimize the number of predictors in the trained model and to determine thresholds to best classify students. Only after the model is fully constructed and appropriate thresholds determined is it applied to the test set (students in the second term). Table 4.1 illustrates the size of each set for our model. The size ratio of the training set to the validation set is 2:1.

Table 4.1: Data Set Size After Preprocessing

Term 1		Term 2
Training Set	Validation Set	Test Set
117	54	142

Data Preprocessing

Two standard steps are conducted to prepare the data for analysis. First, to account for differences in difficulty of exams across terms, we convert raw final exam scores to scaled scores. The scaling is done by first determining each exam score's z-score, then scaling all z-scores between the maximum and minimum z-score. Final exam scores of the training set, validation set, and test set are thereby scaled to the range [0,1].

The second step addresses clicker responses that are missing. A student may fail to answer some clicker questions (e.g., by being absent, arriving late, forgetting to click, or leaving early), but our modeling approach does not handle missing data. If we simply omit students who fail to answer one or more questions, then we would lose the data of the vast majority of students in the class. We therefore use data imputation. Data imputation is essentially informed guessing: we guess how the student would have responded based on their responses and the responses of other students. The accuracy of such guesses is impacted by the number of questions that the student legitimately answered. (We investigate the impact of students who commonly miss class in Section 4.2.4.) Data imputation is performed using the well-established R mi package [85].

Dimensionality Reduction

As many clicker questions were asked each week, using all responses would lead to a model that overfits the data. To prevent this overfitting, we use Principal Components Analysis (PCA) to reduce the number of dimensions in the data using the R Caret package [38]. PCA extracts a given number of predictors (i.e., composites of clicker questions) that best represent the data. We explore using different numbers of predictors to determine the best number for our data

by optimizing the model accuracy with regard to the validation set (the test set is not used in this process).

Building the Linear Regression Model

We create a linear regression model using the principal components chosen by PCA and predict a scaled final exam score for each student. The prediction model was created and tested using the R Caret package [38]. In the training phase, we use k-fold cross-validation (k=10) to optimize the regression model parameters. The trained model also provides variable importance information, which describes the predictive power of each clicker question.

Classifying Students

The output of the model, when applied to either the training or test data, is a predicted final exam score. This prediction can then be used to classify students. Each classified categories of students are named as what we defined in Chapter 3.3

Given that we have predicted final exam scores, we could simply apply the original threshold that defines at-risk students to the predicted final exam results. However, due to error variance in predicted scores, this choice may not be ideal in that it could result in a large number of False At-risk or False Not At-risk students. As such, we instead use the validation set (again, not the test set) to determine an appropriate threshold for classifying students. We refer to this second threshold as the classification threshold or intervention threshold. We explore tradeoffs inherent in determining this threshold in Section 4.2.1.

To summarize, there are two relevant thresholds in our modeling approach. The first is the percentage of students that should be classified as at-risk (this can be based on common exam outcomes and final grades). The second is the threshold that we use to classify students as in need of assistance, taking into account the likelihood of misclassifying students.

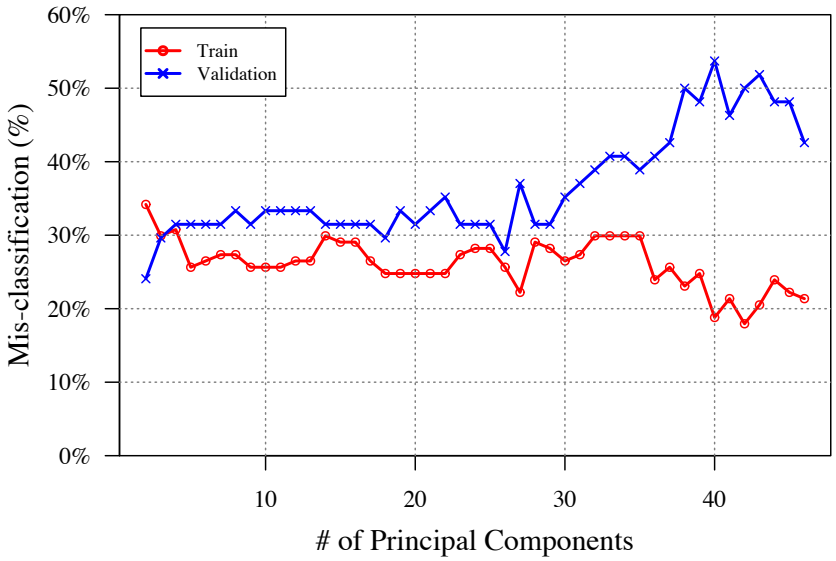


Figure 4.1: Impact on Accuracy of the Number of Principle Components

4.2 Results

In this section, we predict student outcomes across terms. We describe steps taken to build the identification model and apply it to identify at-risk students. We then explore the model further to learn about influential PI votes, class attendance, and applicability to lecture classes.

4.2.1 Constructing the Model

PCA Variable Selection

We use the training and validation set to determine the appropriate number of principal components to use in building the model. We define model accuracy as the correct classification rate. Figure 4.1 illustrates the misclassification rate of the training set and the validation set with respect to the number of principal components. The misclassification rate is the proportion of students for whom a prediction of being in At-risk or Not At-risk would be incorrect. This figure demonstrates that fewer than 30 principal components provides good validation set accuracy. Additional components may improve the accuracy of the model for the training set, but at the

potential expense of the validation and test sets. With a view toward choosing fewer rather than more components, and a desire to optimize performance on the training and validation sets, we chose nine principal components for our identification model.

Intervention Threshold

We next set the threshold for the percentage of the class on which to intervene. This intervening score could simply be set to the same point at which we consider students to be at-risk. However, the choice of intervention threshold has a large impact both on model accuracy and on the number of students who are identified as potentially needing help. The higher the threshold, the more False At-risk (students receiving help who do not need it) and the more instructor resources are spent helping those students. In turn, the lower the threshold, the more False Not At-risk (students not getting help who need it).

One can consider the two threshold extremes to better understand the tradeoffs present in selecting an intervention threshold. First, one might choose a threshold of 0%. This threshold would cause no students to be identified for an intervention, meaning that we never help a student who does not need help (zero False At-risk), but also never help students who do need help (maximum False Not At-risk). Second, one might choose a threshold of 100%, in which case we help everyone that needs help (zero False Not At-risk), but also help everyone else (maximum False At-risk). The question then becomes: how should we balance these tradeoffs? The answer ultimately comes down to instructor discretion based on available resources and the cost of intervention.

As an instructor cannot know the results for their present class, they can use student data from the prior term to help choose an appropriate intervention threshold. To help visualize the impact of the instructor's threshold decision, Figure 4.2 shows the impact of the intervention threshold for the first term data (training set and validation set combined). In this figure, we see that as we increase the threshold from 0% to 40%, our method tends to more accurately predict

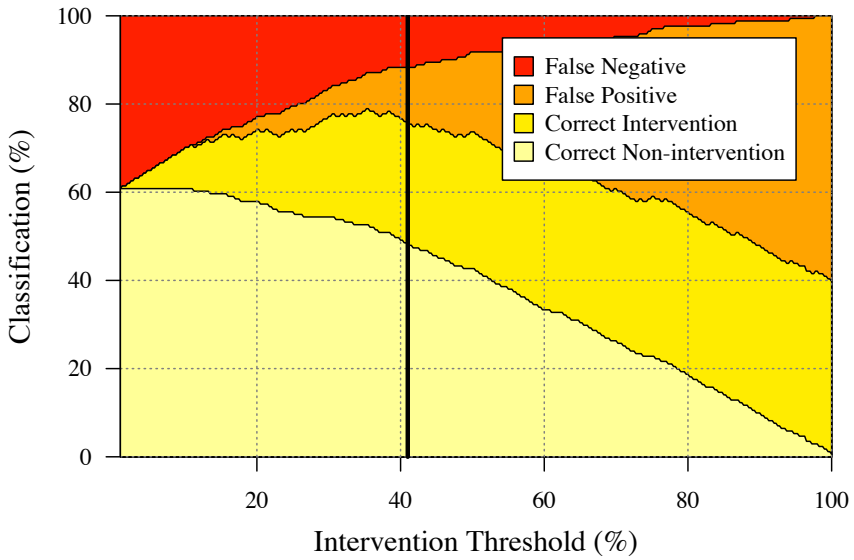


Figure 4.2: Intervention Threshold for Training and Validation Set

at-risk students. However, as we continue to raise the threshold, we introduce False At-risk at an increasing rate.

For the remainder of this analysis, we chose the intervention threshold with the highest accuracy (i.e, that minimized the misclassification rate). Other threshold decisions to minimize False At-risk or False Not At-risk would be possible and would be at the instructor's discretion.

We use the validation set alone (not with the training set) to determine the intervention threshold with the highest accuracy; that threshold is 41%. This means that to identify students who are in the bottom 40% of the class, we will conclude that students whose predicted final exam score is in the bottom 41% will be at-risk. Although the small difference between 40% and 41% may suggest that the alternative threshold is unnecessary, when we use different model parameters we found larger thresholds (e.g. 55%). We next examine our model for the training data in the context of this threshold and then apply this threshold to the test data.

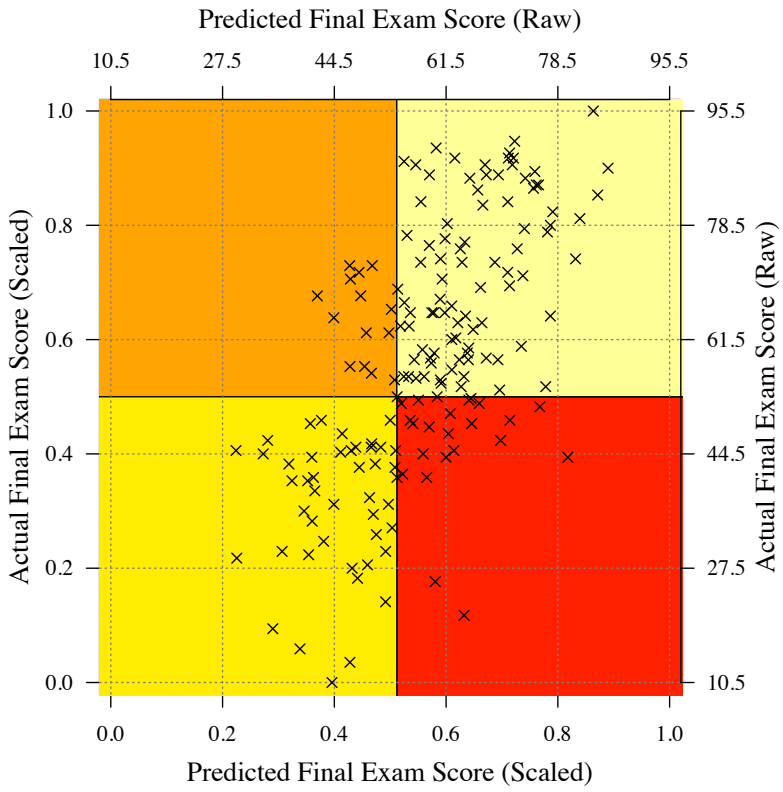


Figure 4.3: Model Accuracy for Training Data

Model Accuracy for Training Set

Although the ultimate goal of developing this model is to examine its accuracy on the test set, here we begin by examining the model's accuracy for the training set. Figure 4.3 plots the model's predicted final exam score per student (x-axis) against their actual final exam score (y-axis). There are two labels per axis. The "Raw" scores are the actual scores on the exam out of 105 points. The "Scaled" scores are scaled by the students' z-scores and are hence between 0 and 1. Recall that the reason for the scaled scores is that exams across multiple terms may have different difficulty.

The linear model in this figure, built using nine principal components, is fairly accurate ($rho : 0.628, R^2 : 0.395$), considering that R^2 of the correct predictions is 1. To examine the prediction accuracy, we focus on the four colored quadrants. Keeping with the color convention

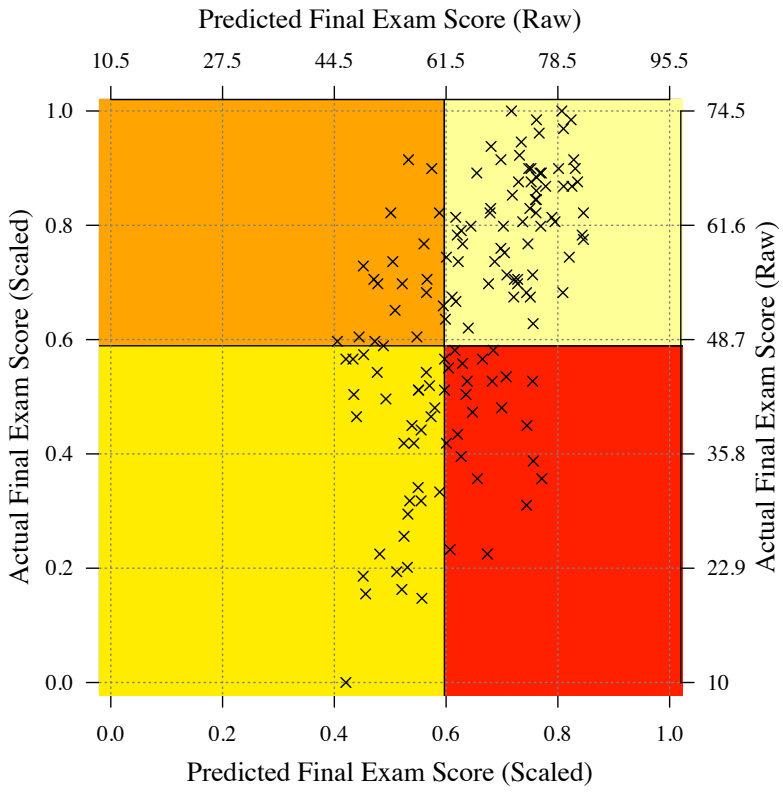


Figure 4.4: Model Accuracy for Test Data

from Figure 4.2, we can identify the regions of correct predictions (At-risk, Not At-risk, False At-risk, and False Not At-risk) for the threshold chosen above.

4.2.2 Applying the Model

Model Accuracy for Test Set

Figure 4.4 provides the plot of predicted versus actual final exam score for the test data. The model is again reasonably accurate ($rho : 0.574, R^2 : 0.329$) despite the differences between terms (ordering of topics, different students, different exams, etc.). We revisit these differences in Section 4.3.

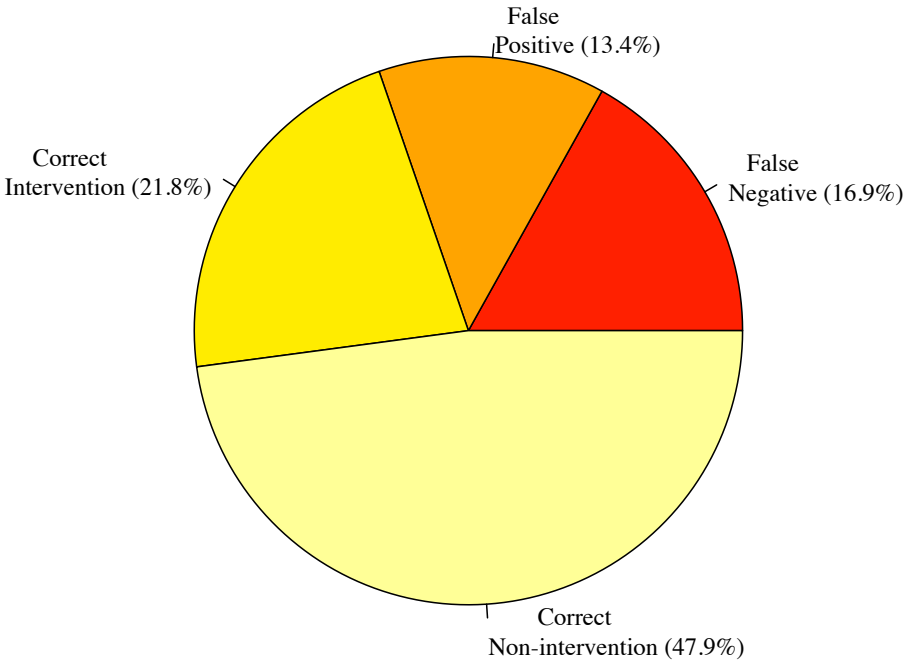


Figure 4.5: Classification Accuracy for Test Data

Student Intervention Accuracy

While the model provides a predicted final exam score, our intended use is to apply the 41% threshold to determine students who are likely to be at-risk. Figure 4.5 provides the accuracy of our student classifications. 70% of students are accurately predicted. 13% of predictions are False At-risk and 17% are False Not At-risk. Recall that one can increase the intervention threshold to decrease the number of False Not At-risk, but at the expense of increasing False At-risk.

As mentioned above, an instructor's choice of intervention threshold impacts False Not At-risk and False At-risk. Figure 4.6 demonstrates the impact of the intervention threshold on the test set. This figure is shown only to demonstrate the tradeoffs in the context of the test set results; recall that the threshold was determined using the validation set. Such a figure of the test set results would not be available until the end of the later term, too late for intervention to be useful.

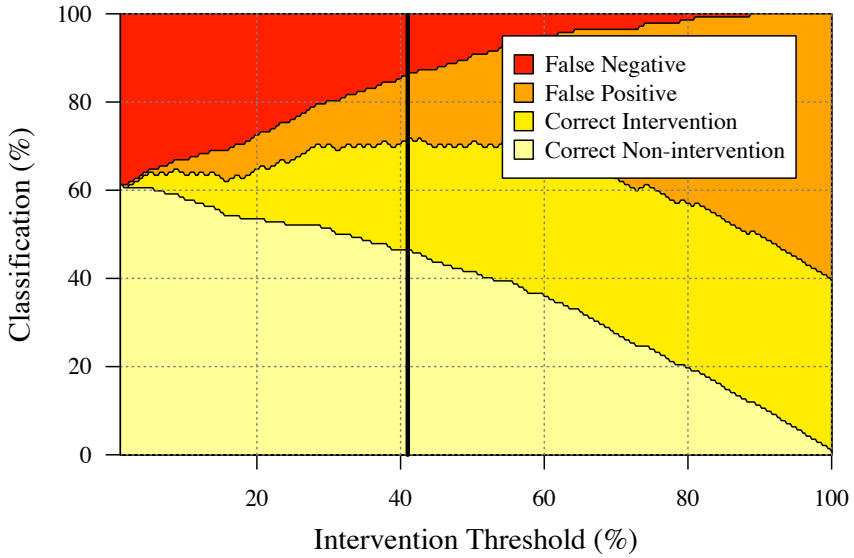


Figure 4.6: Threshold Impact for Test Data

4.2.3 Model Influences

Figure 4.7 provides the importance scores for the early-term clicker questions as determined by the model. In the top half of the figure, the questions are labeled by the relevant vote in the PI process: individual (solo), group, or isomorphic (iso). The key take-away here is that both isomorphic and individual votes play a larger role, particularly among the very top predictors, than group votes. This corresponds with prior findings that group votes can be noisy due to confounds between actual learning and copying perceived correct answers [58].

In the bottom half of the figure, the questions are organized by the time they occurred in the term.[1] Here we see that questions from week 2 are among the very top predictors. Week 2 content includes functions, particularly return types, and boolean expressions and conditionals. As expected, questions from week 1 and week 3 also appear among the top 15 predictors.

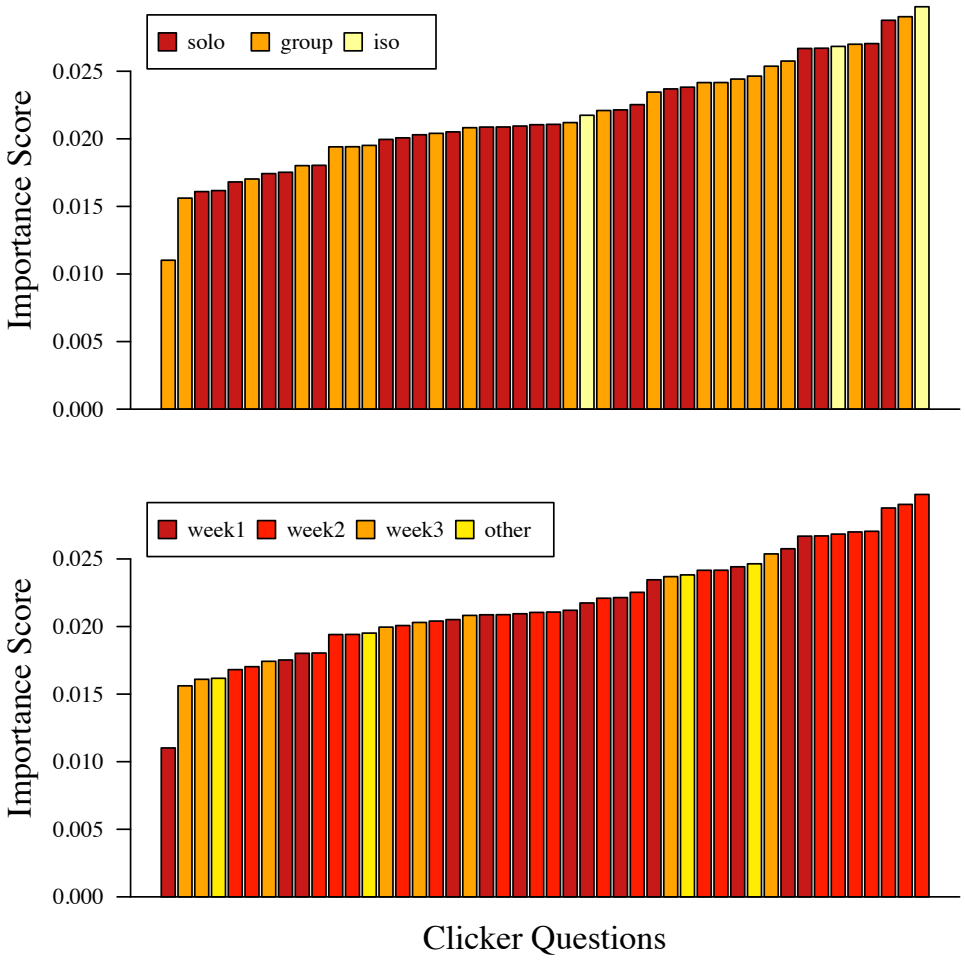


Figure 4.7: Question Importance for the Model

4.2.4 Impact of Student Attendance

The results above include all students in the test set, regardless of their class participation. Is it reasonable to expect the model to predict a student's final exam score when they have only attended a small number of classes? To study this question, we examined the model's accuracy on only students who answered 70% or more of the clicker questions. This attendance criteria excluded 31% and 13% of students in the training and test set respectively. For this large subset of students, the model predicting final exam scores for the test set remains similarly accurate

[1] "Other" occurs because some questions that appeared in the first 3 weeks of the test set were in later weeks in the training set due to minor reorganization of topics in the course.

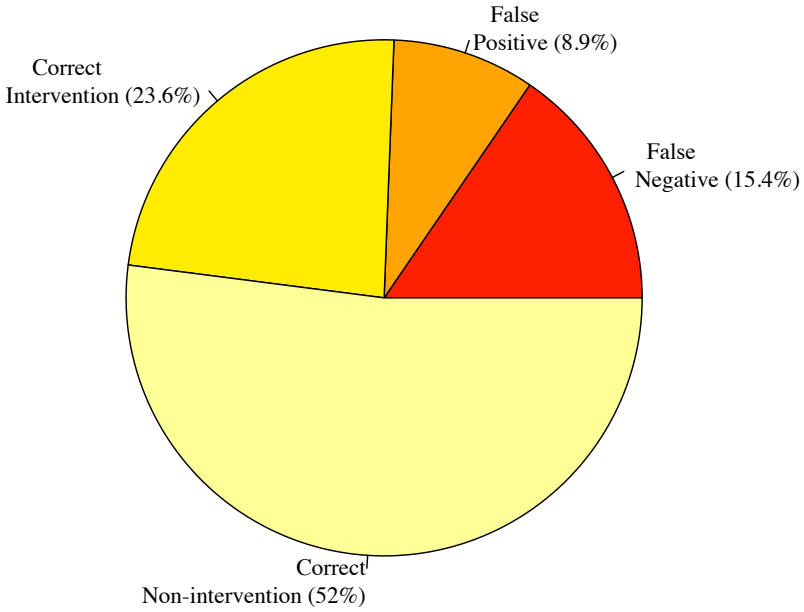


Figure 4.8: Model Classification Accuracy for Only Frequently-Responding Students in Test Data

($rho : 0.556, R^2 : 0.309$).

The classification of At-risk students is more accurate than when all students are included. Figure 4.8 provides the resulting classification accuracy. The model's accuracy for student classification has increased from 70% to 76%, demonstrating the importance of class attendance for model accuracy.

4.2.5 Applicability to Non-PI Classes

One core aim of this effort is to ensure that the modeling is made more accessible to other instructors who wish to benefit from predictions in their classes. Although PI has gained considerable traction in computing [60], a large number of instructors may not wish to adopt PI and/or clickers. In this section, we examine the possibility of using the PI clicker questions as either before-class quizzes or brief start/end-of-class quizzes. The questions are available at [21].

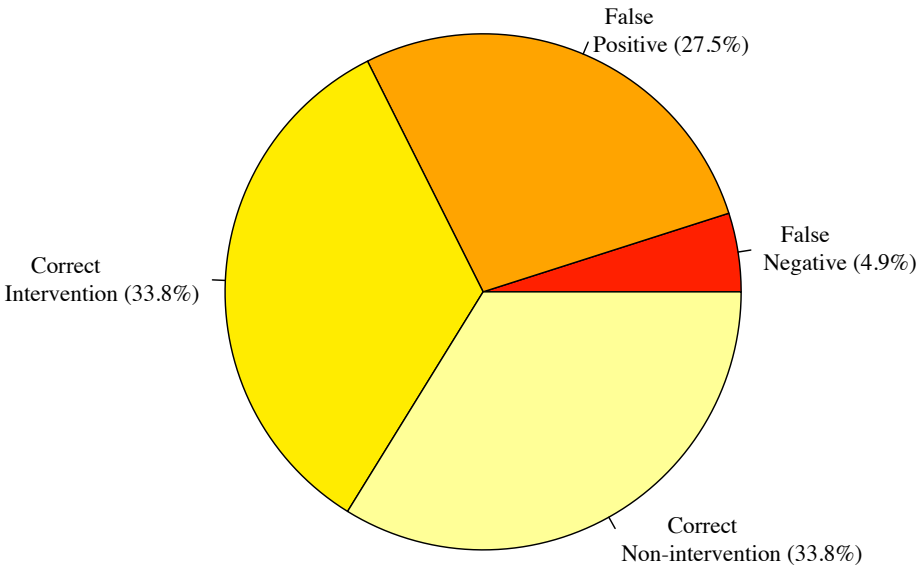


Figure 4.9: Classification Accuracy for Test Data When Limited to Only Using Individual Votes

To explore this idea, we built a new model using only individual clicker votes. These votes may be representative of student thinking before class as they occur before the group and classwide discussion. One confound, however, is that they may occur mid-way through a class and students may have learned from prior class content. Alternatively, one might ask these questions after the class, in which case the results may more closely resemble the isomorphic votes. As the isomorphic votes were highly predictive in our model, the results in this section may understate the expected modelling ability of questions used at the end of class.

Compared to the full model, this new model is only marginally worse at predicting student final exam scores in the test set ($rho : 0.546, R^2 : 0.299$) and at classifying poor performers. Figure 4.9 shows that the classification accuracy is 68% for all students, compared to 70% accuracy when we included the group and isomorphic votes. Although False Not At-risk have been reduced significantly, this is simply due to selecting a higher intervention threshold for maximum accuracy, which, in turn, increased the number of False At-risk. The overall accuracy

suggests that instructors who wish to adopt this approach with lightweight multiple-choice quizzes may be able to successfully predict at-risk students without the requirement of adopting PI.

4.3 Discussion

This work demonstrates the potential to use student clicker data to identify at-risk students across terms. The requirement on instructors is lightweight as they can either use the same clicker questions across terms or, potentially, quizzes including these multiple choice questions. After building the linear model, instructors can then make their own decisions about how to identify at-risk students based on the two threshold values: the percentile on the final exam that is considered at-risk and how to optimize their classification threshold.

4.3.1 Revisiting Results

Table 4.2: Topics among top 10 predictive questions, ordered by importance rank. Questions whose votes appeared multiple times include the importance rank and corresponding vote categories, respectively.

Importance Rank(s)	Lecture Week	PI Question Type†	Question Topics
1,5,7	2	iso, gr, ind	Code tracing through nested function calls where variables used are in or out of scope
2,3	2	ind,gr	Logic/ Boolean expression evaluation and boolean variable assignment
4	2	ind	The difference between a function printing a value versus returning a value
6	2	iso	Given a boolean expression with variables, determine what values those variables would need to have to evaluate as false
8	1	ind	Code tracing through a single function call where the function has multiple arguments
9	1	gr	Variable types (integer versus double)
10	3	gr	Code Tracing through nested conditionals

†iso: isomorphic, gr: group, ind: individual. Details on each type is provided in Chapter 3.2.

Cross-Term Differences

The ability to predict student outcomes across terms may seem straightforward given that the clicker questions changed little from the first term to the second. However, a number of differences are evident even though the same instructor taught both terms. The differences include: different students, different in-class student inquiries and resultant discussion, different assignments, different topic ordering (topics were rearranged slightly to address assignment changes), and most critically, different exams. An additional confound is the noise and missing data inherent in clicker responses graded on participation. In that sense, the performance of our modeling methodology is surprising when these differences and confounds are recognized.

Question Quality

The quality of the model is implicitly based on the quality of the clicker questions and their utility for identifying student misconceptions across a variety of topics. Two major factors contributed to the quality of the questions in this course. The first is that CS1 has been widely studied by the computer science education research community and the design of these questions was informed by that research. The second is that the instructor has taught the course for a number of years and has refined the questions over this time period. As a result, we provide these questions to those wishing to adopt this approach [21].

Helping At-risk Students

This paper is not the first to recognize the inherent benefits of identifying at-risk students early in the term [5, 65], but the approach outlined here vastly reduces barriers to adoption (e.g., large number of coding assignments in the first week). Once it is easy to identify these students, instructors will be tasked with acting on these results. Interventions to improve students' outcomes deserve further attention and are the focus of ongoing work.

Definition of At-risk Students and Intervention Threshold

The thresholds for at-risk students have a large impact on the classification accuracy and, potentially, instructor resources dedicated to intervention plans. We chose a bottom 40% threshold based on cutoffs chosen by the instructor, but other values could be chosen. We caution, however, that the modeling technique outlined here is more effective at classifying large groups than small groups. As such, this particular model may struggle to identify small subsets of students (e.g., the bottom 5%) as identifying such outliers is both difficult in general and ill-suited for this model.

As mentioned previously, an instructor's choice of intervention threshold impacts the number of students for whom intervention is offered. An instructor can optimize this threshold to maximize accuracy, minimize False Not At-risk, or perform an intervention for some particular number of students. Resource requirements may, at least partially, constrain the instructor. For example, should the instructor wish to e-mail students in jeopardy, they may optimize for a fairly low False Not At-risk rate (accepting more False At-risk) because the intervention is inexpensive. In contrast, if the instructor aimed to have a special in-person session for struggling students, they may optimize based on room sizes or the availability of instructional staff.

4.3.2 Comprehensive Picture of Early CS1

Prior work suggests fruitful links between early identification of students and informing what we know about CS concepts that are challenging to students [5, 65]. Porter et al. focus on individual clicker questions and their relationship to exam outcomes [65]. However, due to the type of modeling used, a number of highly correlated clicker questions on the same topic may all appear important. Ahadi et al. study predictors in the larger context of a machine-learning model, but the predictors are scores on code-writing assignments that do not isolate individual concepts [5].

Our method allows us to again benefit from the best features of each of these prior works.

By examining highly-predictive clicker questions in the context of a comprehensive model, we gain a more complete picture of the critical topics and concepts in the first three weeks of the course.

To do this, we examined the top ten questions in terms of importance from Figure 4.7. As each question may be answered multiple times through the individual, group, and optional isomorphic vote, a single question may appear multiple times among the top predictors. Although this may seem contradictory to the notion that highly-correlated results should be pruned from the model, recall that each of these votes represents a different point in student understanding: before discussion, after discussion, and after instructor explanation, respectively. Indeed, two questions appeared for multiple votes, resulting in eight unique questions.

The topics of these questions appear in Table 4.2. Critical topics from the beginning of CS1 appear in the top predictors, including variables, types, boolean expressions, function calls, parameters, scope, and conditionals. Loops are absent from this list because they do not appear in the course until Week 4. None of these topics, nor their ranking, appears particularly contradictory to what one might expect as top predictors from an introductory course [29, 69].

4.3.3 Threats to Validity

There are two categories of threats to validity. The first is with regard to the building of the model while the second focuses on the course itself.

Model Construction

- **Class size**: The construction of this model required a fairly large class as we partition the training data into a $2/3$ training set and $1/3$ validation set. Moreover, among the remaining training set, k-fold cross validation (k=10) is necessary to avoid overfitting. As such, the applicability of our technique to smaller classes is unknown.

- **Model Robustness and Overfitting**: In exploring successful model construction on the training set, a number of parameters were explored and the resultant models had similar degrees of correctness. Use of the test set was limited in order to avoid overfitting to the test set. However, in the research process, the test set was queried more than once. The robustness and similarity of results (e.g., classification accuracy between 68%-73%) across these queries suggest that overfitting did not occur, however whenever a test set is examined more than once, overfitting/overtuning becomes a concern.

Cross-Term Course Repetition

- **Questions**: As mentioned, the clicker questions in this study had a basis in the literature and have been used in multiple courses. As the model is based on the question results, modeling for different questions may yield different results.
- **Exam**: The questions on the final exams across the two terms are completely different because exams are made public at the instructor's institution. As previously mentioned, the differences between the exams (both in terms of question difficulty and conceptual coverage) may lead to lower model accuracy. By contrast, reusing significant portions of exams could yield increased modeling accuracy.
- **Instructor**: The same instructor taught both terms of the course. Whether the model applies to another instructor using the same questions is unknown. However, the later work provided in Chapter 5 and 6 applied a single technique to multiple courses taught by multiple instructors.

4.4 Acknowledgements

This chapter, in full, is a reprint of the materials as they appears in Proceedings of the Conference on International Computing Education Research. S. N. Liao, D. Zingaro, M. A.

Laurenzano, W. G. Griswold, and L. Porter, ACM, 2016. The dissertation/thesis author was the primary investigator and author of this paper.

Chapter 5

Robust Identification Technique in Multiple CS Courses

Chapter 4 proposes an early and lightweight identification method using student clicker data. However, the method is only tested on a single cohort of CS1 students from a single institution. Thus the work in this chapter has two primary research questions:

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

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

5.1 Method

To answer our research questions, we developed a modeling technique and applied it to data from courses from two institutions and five courses. 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.

Table 5.1: Dataset Details

Course	Institution	Instructor	Training Set (Size)	Test Set (Size)	# of Clicker Questions
CS1-Python	B	Y	Fall 2013 (192)	Fall 2014 (142)	54
CS-Java	A	X	Fall 2014 (373)	Fall 2015 (374)	17
CS-Java	A	X	Spring 2014 (169)	Spring 2015 (176)	34
Adv. DataStruct	A	X	Fall 2013 (197)	Winter 2015 (191)	24
Aritecture	A	Z	Fall 20 15 (339)	Fall 2016 (266)	52

Data Collection

Our data were obtained from three instructors across two public universities in North America. Detailed information for each course batch is provided in Table 5.1. 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.

Each course batch 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 used Peer Instruction in their courses regularly.

5.1.1 Data Preprocessing

The data used in this study consists of student clicker responses 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 is deidentified and 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 from our training and test set for that course any questions that were not present in both sets. The number of clicker questions after this selecting process is provided on the last column of Table 5.1. Average correctness of clicker questions varied from 0.01 to 0.99 but on average, it was within the 0.50-0.75 range.

- **Labeling question correctness:** For each clicker question response, 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 in Chapter 4, which used data imputation to statistically predict missing clicker responses. Data imputation relies on patterns in legitimate responses to guess missing data values. One concern here 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 and, at the very least, imparts whether the student attends class or not.

5.1.2 Model Generation

Figure 5.1 illustrates our machine learning process, where the model is built based on the training set and that same model is then used on the test set to predict students at-risk. To build the

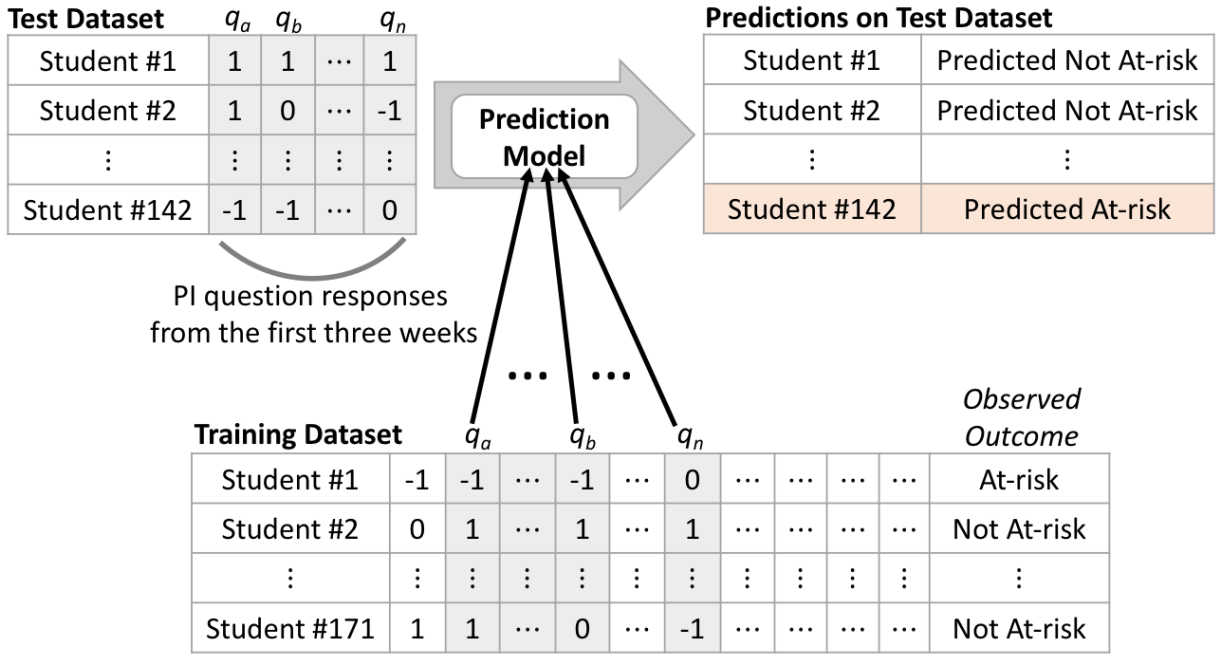


Figure 5.1: The Machine Learning Process Used in This Chapter

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 offered the most consistent results compared to other machine learning models (logistic regression, decision tree, and random forest) across our courses. We used a well-known package in R named kernlab [84] for implementation.

As part of the training process, we tuned three modeling parameters: C, sigma, and classweights [84]. These modeling parameters were tuned during the model validation stage for each course. In the validation stage we used two-thirds of the training set to build a number of models with different parameter values and chose the best one using the rest of the training set samples. We describe how we measured the quality of our model in Section 5.1.3.

After choosing the optimal modeling parameters, these parameters were used along with the entire training set to train our final prediction model. We used 10-fold cross-validation on the entire training set to generate the final model for each course.

5.1.3 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 what percentage of students are misclassified by the model, it does not reveal the tradeoff between different types of errors that our model might make.

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 provided below.

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

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 for not needing assistance.

Receiver Operating Characteristic (ROC) curves [33] are a well-established metric for model evaluation in machine learning, which plot both sensitivity and specificity in a 2-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. An example ROC curve is provided in Figure 5.2. The x-axis plots how accurately a model identifies not at-risk students (specificity) and the y-axis plots how accurately a model

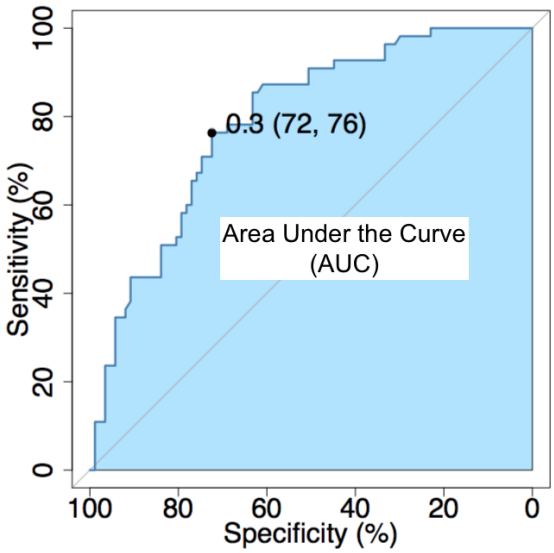
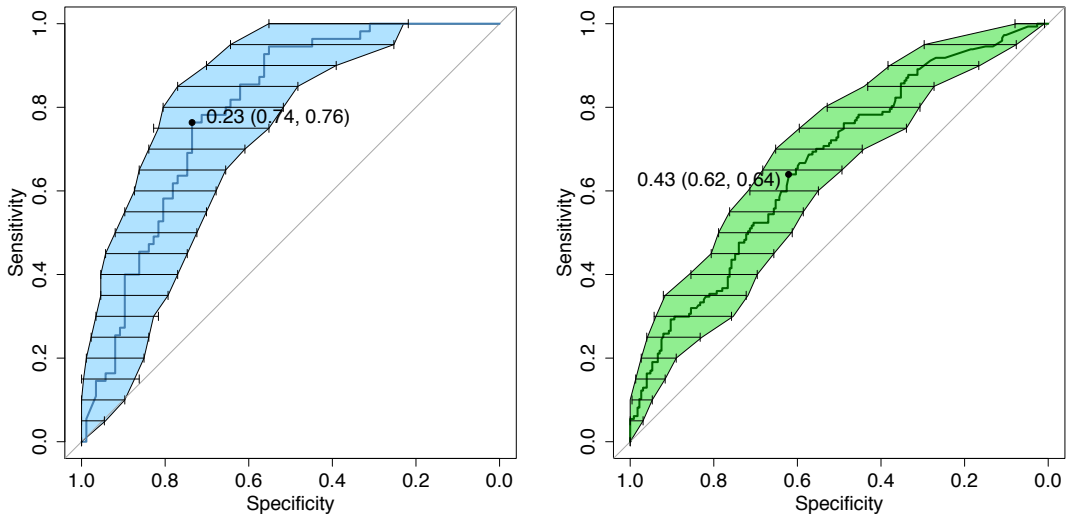


Figure 5.2: An Example ROC Curve

identifies at-risk students (sensitivity). The output of the logistic regression model is a number between 0 and 1 representing the probability that a particular student is at-risk. We can trade off the specificity and sensitivity of our model by varying the probability threshold we use to decide whether a student is classified as Not At-risk or At-risk. This tradeoff is what is shown in the ROC curve; each point on the curve corresponds to a different selected value for the probability threshold. For example, the point corresponding to a probability threshold of 0.3 is shown in Figure 5.2. This example point, for this model at this threshold, produces a specificity of 72% and a sensitivity of 76%.

When evaluating the overall modeling performance, the Area Under the Curve (AUC) on an ROC curve is commonly used as it captures, in effect, how close to an ideal curve is provided by the model. Figure 5.2 visualizes the AUC in blue. Overall, a high AUC (closer to 1.0) implies a better model as it is better capable of capturing sensitivity and specificity.



(a) CS1-Python (Institution B, AUC 0.79)

(b) CS1-Java (Institution A, AUC 0.67)

Figure 5.3: Cross-Institutional ROC Curves

5.2 Results

In this section, we evaluate our model's performance on the courses included in this study. We begin by examining how well the model performs 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.

5.2.1 Multi-Institutional Analysis (RQ1)

Addressing Research Question 1, we examine whether the modeling approach is successful across institutions. Figure 5.3 provides the ROC curves for the two CS1 courses taught at two institutions.

In this figure, 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 5.3 indicate the confidence

intervals of specificities 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 set for the best combination of sensitivity and specificity. If this probability threshold (first number) is selected, it produces the specificity (second number) and sensitivity (third number) for the test set.

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.

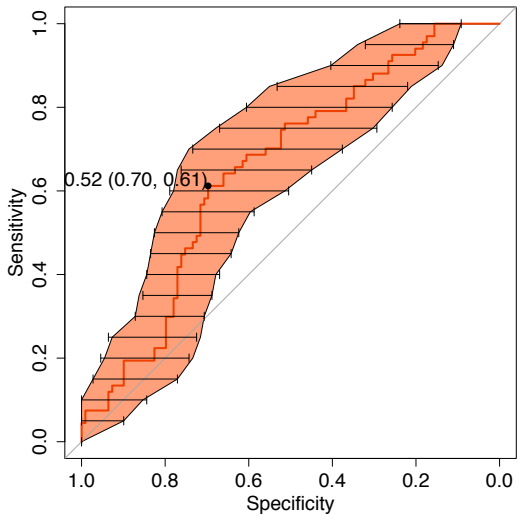
For each of these CS1 courses, we find that our modeling approach provides reasonable classification predictions. The AUC is well above 0.5 (0.79 and 0.67) and at the best probability threshold, the models offer specificities of 0.74 and 0.62 and sensitivities of 0.76 and 0.64. We discuss how these metrics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

Given the differences between institutions and courses, it is to be expected that the model would result in slightly different prediction accuracy. In Section 5.3 we discuss possible sources of this difference.

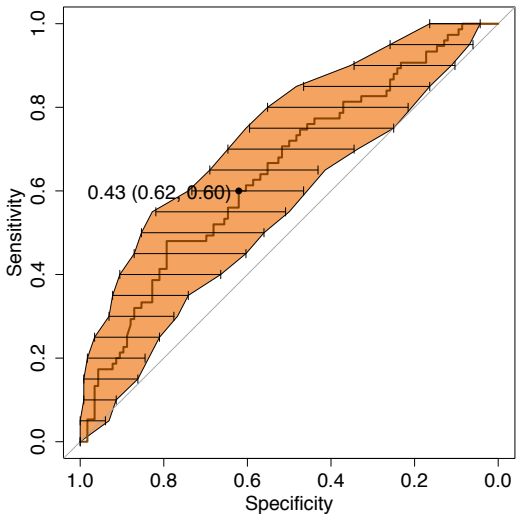
5.2.2 Analysis across Courses in the CS Curriculum (RQ2)

Focusing now on Research Question 2, we examined PI clicker data for four courses across the CS curriculum. Figure 5.3b along with Figure 5.4 are the ROC plots of four different courses taught at the same institution covering material ranging from introductory to advanced computing topics. The results are highly encouraging as the model is 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.

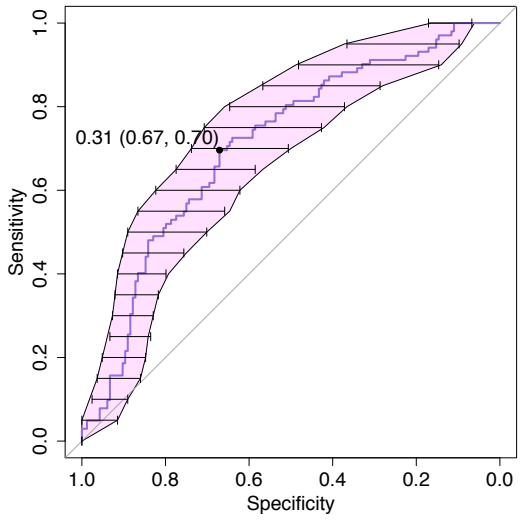
There is no truly comparable study in the literature against which to compare these results. However, our results are consistent with recent studies predicting student outcomes across terms using similar in-progress data in introductory computing courses [5, 13, 42]. Recall from



(a) CS2-Java (AUC 0.67)



(b) Adv. DataStruct (AUC 0.65)



(c) Architecture (AUC 0.71)

Figure 5.4: Cross-Curricular ROC Curves from Institution A

Section 2.1.4 that one study used data that was more heavyweight [13]. As such, our approach in this chapter and Chapter 4 is more lightweight and more broadly applicable than prior work while maintaining similar levels of prediction performance.

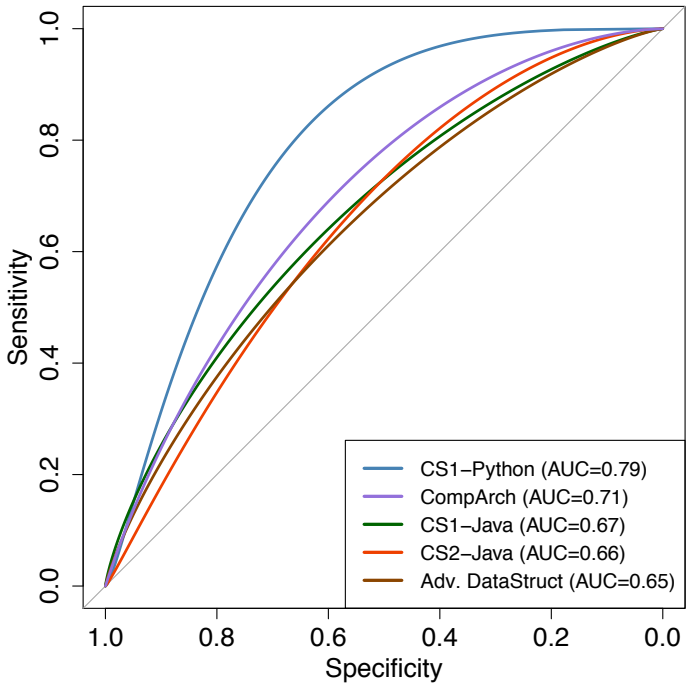


Figure 5.5: Smoothed ROC Curves

5.2.3 Overall Analysis

Figure 5.5 and Table 5.2 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 5.5, 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.

Table 5.2: AUC and its 95% Confidence Interval.

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

5.2.4 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 5.6 provides prediction quality using 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 5.6 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 data collected later in the term is temporally and topically 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 [66]. We suspect there is a similar effect in courses later in the curriculum, in that these courses tend to build on early material throughout the term, making early material critical for later success.

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 [66] as offering reasonable accuracy early in the term; our data supports this finding.

5.3 Discussion

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

5.3.1 Applicability of Prediction Methodology

Similarly to Chapter 4, the method in this chapter 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 [64], 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.

The prediction method in this chapter also 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 this method requires the use of clickers and the PI pedagogy: what is one to do if they do not use PI? We suggest that this is not a core threat. As mentioned in chapter3, PI has become more widely used in computing courses including programming, theory, and system courses [21, 39, 57]. This is advantageous here, as prior work suggests that PI questions can be productively used as quiz questions in an otherwise non-PI course [42]. 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. In addition, the method defines at-risk students based on their

final exam scores so it does not work in courses that do not have final exam. We could potentially use students final project score or final grade as a dependent variable of the prediction method, instead of final exam scores, but further study is required in the future.

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 paper 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 available 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.

Lastly, we highlight the consistency of our prediction performance across our two institutions and multiple CS courses. This lends support to the generalizability of our results: rather than possibly being confined to a particular course context, we have demonstrated modeling facility across courses, instructors and institutions. The PI context is particularly apropos here by virtue of its automatic collection of data that could be shared within and across institutions.

5.3.2 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, differences in students, the extent to which course misconceptions are known in the literature and used to develop the course, how many iterations of the course have occurred to polish questions, instructor experience generating good PI questions, and how closely the final

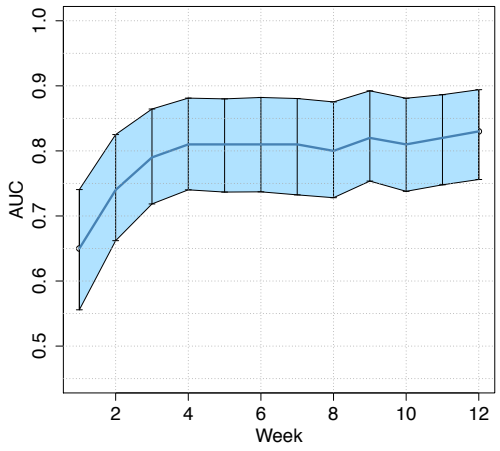
exam aligns with the PI questions. We hesitate to posit theories on which factors are more or less critical in the courses we have studied here, but we suspect many of these factors interact to influence the extent to which we can make accurate predictions.

5.3.3 Learning about Students from the Model

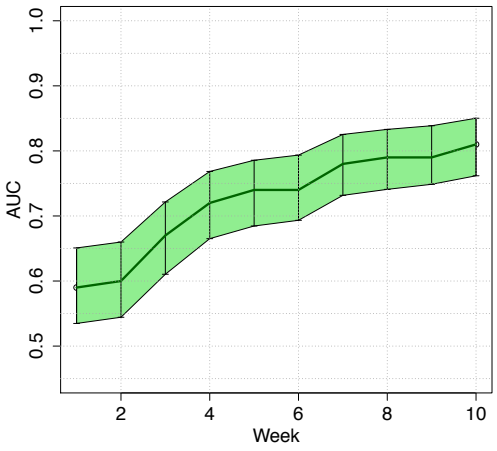
One of the promising elements of this work is that our models are successfully extracting some measure of student understanding from student responses to PI questions. Prior work has used Bayesian classification [66] and model variable importance (Chapter 4) to reverse engineer the questions that were 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 paper. 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 their questions from the model results.

5.4 Acknowledgements

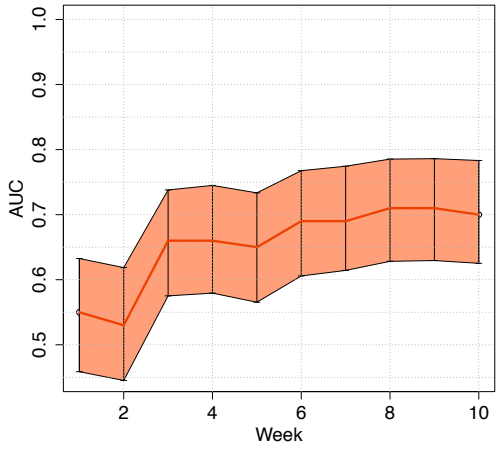
This chapter, in full, is a reprint of the materials as they appears in ACM Transactions in Computing Education. S. N. Liao, D. Zingaro, K. Thai, C. Alvarado, W. G. Griswold, and L. Porter, ACM, 2019. The dissertation/thesis author was the primary investigator and author of this paper.



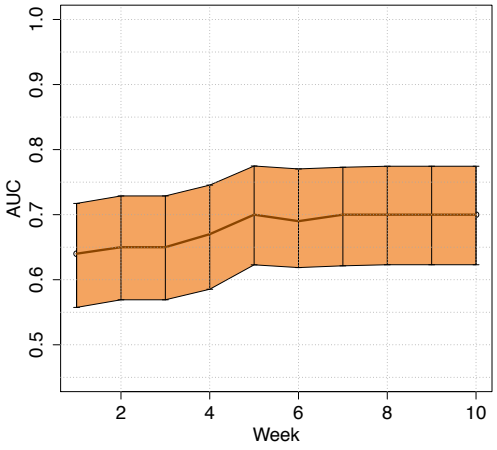
(a) CS1-Python



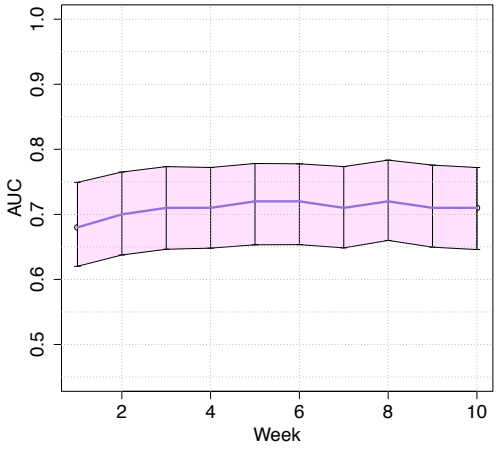
(b) CS1-Java



(c) CS2-Java



(d) Adv. DataStruct



(e) Architecture

Figure 5.6: AUC and its 95% confidence interval, adding more clicker data.

Chapter 6

Exploring the Value of Different Data Sources in Multiple CS Courses

Unlike Chapter 4 and 5, the work in this chapter utilizes other easy-to-collect different data sources. This chapter was guided by two central research questions:

RQ1: What is the value of each of our data sources for predicting student performance in computing?

RQ2: How do prediction accuracies improve as we use multiple data sources?

Table 6.1: Data Sources Used in the Present Study

ID	Description	Preprocessing Method	CS1-Python	CS1-Java	CS2-Java	Adv. DataStruct	Architecture
p	Final grades from prerequisite courses	Numerical grade			✓	✓	✓
c	In-class clicker correctness	Average correctness	✓	✓	✓	✓	✓
a	Take-home assignment grades	Average score		✓	✓	✓	✓
q	Online quiz grades	Average score	✓	✓	✓	✓	✓

Table 6.2: Dataset Details

ID	Prerequisites
CS1-Python	None
CS1-Java	None
CS2-Java	CS1
Adv. DataStruct	CS2, Programming Tools Lab, Discrete Mathematics, Computer Organization
Architecture	Computer Organization, Digital Logic, Digital Logic Lab

6.1 Methods

6.1.1 Our Datasets

This work used the same datasets as described in Table 3.1 and Table 5.1 but with more various data sources. Table 6.1 describes all the data sources including clicker data and others used for the present study. We selected relatively common course components within these five courses to predict student performance: course prerequisites, clicker responses, assignments, and online quizzes. Course prerequisites (see the Prerequisites column in Table 6.2) are not available for CS1, as it is the first CS course taken by students at these institutions. Clicker data was available for all five courses, as all instructors used Peer Instruction [20] in class. Peer Instruction involves the use of conceptually-rich multiple-choice questions to which students respond individually and then in groups. The questions are designed to target student difficulties and misconceptions, and are informed by teaching experience, research literature, and available question banks. Assignment grades were incorporated into four of the models; no assignment data was available for CS1-Python, because the first assignment in that course was due later in the term. Arch students completed individual assignments. In the other courses, pair programming was required (CS1-Java) or encouraged (CS1-Python, CS2-Java, Adv. DataStruct). Online quiz data was available in all courses through pre-class reading quizzes; CS2-Java also had review quizzes assessing what students learned in the past week.

6.1.2 Model Generation

The model generation process in this chapter can be divided into two major steps: preprocessing the data and training a model. We used the glm R function [2] to generate and evaluate prediction models.

- **Preprocessing.** We preprocessed each data source as indicated in the Preprocessing Method

column of Table 6.1. Each data source was distilled into a single value per student, except for prerequisites where we used one value per prerequisite course grade. The reasoning behind keeping the course grade for each prerequisite course is that one prerequisite course grade represents a student's overall academic standing in that course; while the other data sources contain student's understanding on a specific concept that are taught in a specific lecture. When we kept each assignment score, the modeling performance was slightly worse than aggregating to a single value per student.

We collected final letter grades from the prerequisite courses, and then converted these letter grades into numerical grades on a 0-4.33 scale, so A+ became 4.33, A became 4.0, A- became 3.67, and so on. For clicker data, we used records of student responses to each clicker question, following prior work [66] stating that student clicker correctness is correlated with final exam performance. The correlations between student correctness and the z-score of final exam scores from all 5 courses are in the range of 0.31 – 0.63. Similarly, for assignments and quizzes, we calculated average grades, after scaling the maximum possible score of each assessment item to be 1.0 to correct for different maximum possible scores. As a final step, we normalized each data feature to have a normal distribution.

- **Training.** We trained a Logistic regression model [91] to perform binary classification (low-performing vs. high-performing) for each course. We selected logistic regression after exploring a number of different models for accuracy, simplicity, and ability to work well with a small number of input features. The model was trained on data from an earlier term with the data sources as features and with the binary outcome labels of low-performing or high-performing based on final exam score.

6.1.3 Model Analysis

Similarly to Chapter 5, the work in this chapter also uses AUC of ROC curves.

6.1.4 Analysis of Multiple Sources

Ultimately, we use a combination of multiple data sources to train and test the model using AUC. We would like to know whether adding a data source leads to improvement in the model. However, one cannot directly determine whether a model is statistically better than another based on resultant AUC curves. As such, we begin by determining whether adding a specific data source is statistically significant, and to do so, we run a likelihood ratio test [51] (using the lrtest R package [1]). This test compares two models, before and after adding a specific data source, and returns a p-value to show whether adding that data source statistically changes the generated model. We define statistical significance at the $p < 0.05$ level. There is an important distinction between this analysis and the model analysis from Section 5.1.3 in that likelihood ratio tests are performed on the training set data only, whereas evaluation with an AUC occurs on a test set. This has the important consequence for our evaluation that a statistically significant improvement in the model's accuracy for the training set, evidenced by the likelihood ratio, may not necessarily translate to an improved result for the AUC on the test set (due to potential overfitting, changes across terms, etc.).

6.2 Results

We first examine the value of different data sources (RQ1) and then explore how model accuracy improves as we use multiple data sources (RQ2).

6.2.1 RQ1: Value of Different Data Sources

To determine which data sources are most valuable, first we examine the resultant AUC of each data source when used in isolation and then we determine whether adding each source statistically significantly improves the model using likelihood ratio analysis.

Value of Data Sources in Isolation

Figure 6.1 provides the resultant AUC for the test set when using each data source individually, collected from the five courses. Recall that CS1 courses do not have any course prerequisites at our institutions, so they do not include results from prerequisite data. Moreover, CS1-Python did not have any assignments due within the first three weeks of the term, so Figure 6.1a does not have results on assignments.

On average, prerequisite data (p in Figure 6.1) returns the best AUC (0.76) out of all data sources. This implies that for courses beyond CS1 (even as early as CS2), prerequisite grades are a powerful predictor of student outcomes. The next highest average AUC (0.70) is provided by clicker data (c), confirming prior work that this data source as a valuable predictor of student outcomes [42, 66]. Online quizzes and assignments provide similar AUC for CS1-Java and Architecture; assignments are better for CS2-Java; and quizzes are better for Adv. DataStruct. Although the Average AUC is 0.68 and 0.63 for online quizzes and assignments, respectively, the inconsistencies between courses means no clear trend between online quizzes and assignments has emerged. Overall, the trend is that prerequisites are more valuable than clickers and clickers are more valuable than assignments or quizzes. In Adv. DataStruct, we see that assignments have effectively no predictive power, as the associated AUC is worse than random guessing (i.e., AUC < 0.5). This is further discussed in Section 6.3.

Value of Data Sources Relative to Each Other

Next, we determine which data sources add statistical significance to the model accuracy. For example, although online quizzes in isolation have some predictive power, perhaps the variance explained by that data source is already better explained by the clicker data.

To perform this evaluation, we test how well the generated model performs as we add one additional data source at a time. Specifically, the initial model with one data source is compared against a null model with only the intercept term to determine the statistical significance of the

Table 6.3: Statistical Significance of Adding Data Sources

	CS1-Python	**CS1-Java**	**CS2-Java**	**Adv. DataStruct**	**Architecture**
'p' only	N/A	N/A	***	***	***
Add 'c' to 'p'	***	***	*		***
Add 'a' to 'pc'	N/A		*	*	***
Add 'q' to 'pca'		***			***

* $p < 0.05$, ** $p < 0.01$, ***$p < 0.001$

initial model. We then determine whether adding an additional data source improves the model for the training set.

Likelihood ratio analysis depends heavily on the order that one adds the data sources. For example, if one adds data sources ordered from least predictive to most predictive, then the least predictive are more apt to be deemed significant. As such, we used the common approach in likelihood ratio analysis of adding data sources from most to least predictive (based on our results from Section 6.2.1). We therefore start with prerequisite data, then sequentially add clicker, assignment, and online quiz data. (We additionally experimented with a variety of different orders, and the overall conclusions remained the same.)

Table 6.3 shows whether adding each data source makes any statistically significant changes in the generated model. The initial model based only on either prerequisite data (CS2-Java, Adv. DataStruct, Architecture) or clicker data (CS1-Python, CS1-Java) is always meaningful compared to the null model. Clicker data is statistically significant overall, not only when compared to the null model (CS1-Python, CS1-Java), but also when added to the model with prerequisite data only (CS2-Java, Adv. DataStruct, Architecture). Also, adding assignment data is statistically significant in all courses but CS1-Java. Lastly, adding online quiz data on top of prerequisite, clicker, and assignment data statistically significantly improves only the model for two courses (CS1-Java, Architecture).

Overall, this statistical analysis confirms findings from Section 6.2.1 that the value of our data sources, from most to least, is as follows: prerequisites, clickers, assignments, and online

quizzes.

6.2.2 RQ2: Combining Different Data Sources

Similar to Section 6.2.1, we examine how the model improves as we add additional data sources. Rather than focusing on the statistical significance of the model for the training set, however, here we look at the resultant AUC for the test set when building a model with a combination of data sources. Because the statistical analysis in Section 6.2.1 was on the training set only, and the analysis here is for the training and test sets combined, we again note that data sources found "significant" in Section 6.2.1 may not translate to improved accuracy here.

Figure 6.2 shows how AUC changes as we add more data to the model. Overall, AUC improves as we add more data, except in Adv. DataStruct and Architecture. In those two cases, the model with both prerequisites and clickers performs quite well, but the additional data does not improve (and sometimes hurts) the model accuracy. The fact that prerequisites and clickers are more meaningful than the other sources for Adv. DataStruct and Architecture was also noticeable when we evaluated each data source individually (see Figure 6.1d and Figure 6.1e). One might suppose that there is a ceiling to the possible prediction accuracy in our courses, given different student demographics, different topic orderings, and different exams each term. It is unclear whether such a ceiling effect is limiting further prediction accuracy for Adv. DataStruct.

For CS1-Python, CS1-Java, and CS2-Java, the model generally improves as more data sources are added (with the exception of online quizzes for CS2-Java). This finding may suggest that the multiple data sources collectively improves accuracy; but one might also argue that accuracy is generally quite high already with just two data sources, and that the additional sources only provide marginal improvements.

6.3 Discussion

This section discusses implications of our results and provides suggestions for instructors wishing to use various data sources to identify low-performing students. We also explain why we selected these data sources and possible threats to validity of our results.

6.3.1 Implications

Our work adds to our understanding of how to predict student outcomes by examining the value of different data sources. To our knowledge, this is the first work that uses various data sources, collected in multiple courses across the CS curriculum, from multiple institutions, to perform cross-term predictions.

- **Prerequisite Grades.** Of the four data sources, prerequisite course data is the most predictive. This predictive power relates to the number of prerequisite courses for each course. For instance, CS2-Java had only one prerequisite (CS1-Java), while Architecture and Adv. DataStruct had two and three, respectively; in turn, the AUC of the prerequisite-only model of CS2-Java is the lowest, followed by Architecture, then by Adv. DataStruct.

  The predictive power of prerequisite grades is unsurprising when one considers that a final grade in a course represents a student's overall academic performance throughout the term. Specifically, the final grade includes direct observations of a student's class performance, often including attendance, assignment grades, midterm exam grades, final exam grades, and so on. Moreover, it likely also captures some underlying factors that lead to student success on these assessments (motivation, study strategies, background knowledge, etc.). Lastly, a single term of data represents a full 10-12 weeks (at our institutions) of student performance. Recall that our other data sources were collected over a comparably brief three-week period. Given these differences in duration, it is somewhat astonishing that our other sources are as powerful as they are.

One may wonder why still some students would fall into at-risk even when they have got high grades from the prerequisite courses. We are hesitant to make any conclusion here, since it requires further study. Some possible reasons could be lacking time to study due to taking too many courses in that specific term, having no social network in class, etc. We suspect the reason is out of scope of this dissertation, because when looking at clicker, online quiz, and assignment data of at-risk students who received higher GPA from the prerequisites, we did not find any significant trend.

One important benefit of prerequisites being so predictive for upper-division courses is that this data is available at the beginning of a term, before collecting any data from the course itself. This suggests the possibility of offering support before the course officially begins. For example, refresher sessions could be offered, and students could be emailed to encourage participation.

- **Clicker Responses.** Clicker data is the second best predictor of student performance, supporting prior findings that use clicker data for prediction [42, 66]. Peer Instruction questions may be particularly predictive as they are designed to address core concepts and highlight common misconceptions. In addition, these questions in our courses are graded on participation rather than correctness [61]. This gives students the freedom to represent their understanding accurately, as there is no grade incentive to solicit assistance. For those who have adopted Peer Instruction in their courses, clicker data is automatically available. Scoring clicker questions presents a small overhead, but is likely performed as part of teaching the course. Those not using Peer Instruction, by contrast, could use multiple-choice questions as part of online quizzes instead [42].

- **Assignments and Online Quizzes.** Our AUC results imply that the additional value of assignments or online quizzes beyond prerequisite and clicker data is relatively small. Two exceptions are the addition of assignment and quiz data to CS1-Java, and the addition of

assignment data to CS2-Java. In each of these courses, neither the clicker-only model (for CS1-Java) nor the prerequisite-only model (for CS2-Java) was as predictive as for the other courses, so this may have allowed more room for improvement from additional data sources. Why prerequisite data and clicker data were less predictive for these two classes is enigmatic, and could stem from a variety of factors (key differences in exams across terms, multiple CS1 courses feeding a single CS2 course, quality of clicker questions, etc.). We leave this as a topic for future inquiry. In addition to the AUC results, the likelihood ratio test results showed that online quiz data does not statistically significantly improve the generated model for most courses. Thus, if an instructor already has prerequisite course grades and clicker data, it may not be worth the additional effort to add assignments and online quizzes to a prediction model.

The poor performance of assignments as a predictor of success in Adv. DataStruct was initially surprising. However, upon reflection, we observed that there were only two Adv. DataStruct assignments given in the first three weeks of the term: one was a pre-class survey and the other was a programming assignment that is easier than other assignments given later in the course. The relative low difficulty of the programming assignment (and subsequently high grades on average) may explain the lackluster performance of assignments as a predictor of overall course performance.

- **Combining Data Sources.** Overall, our results demonstrate that prediction accuracy often improves when adding additional data sources. When choosing to collect data, instructors may wish to prioritize the more predictive data sources of prerequisites and clicker responses. Instructors may consider augmenting the model with other data sources, with the proviso that accuracy gains may be limited.

6.3.2 Ease of Collecting Data Sources

We selected the data sources used in this study as they are generally easy to obtain. Three of the sources (clickers, assignments, and quizzes) are often already part of course assessments. The remaining source (prerequisites) is already available in the school database (although we recognize that access to this data may be limited depending on institution). One might be able to obtain grades in prerequisites through a start-of-term survey of students, though there may be concerns about reminding students of previously poor performance at the outset of a new course. These sources differ from some of those used in prior studies, where data was gathered using an extra instrument, such as a survey, pre- or post-test, or programming IDE [5, 95]. In contrast, we anticipate no significant burden for instructors to use the data sources in our study other than retrieving the required data and, in the case of clicker questions, marking the correct answer if not marked.

Prior work has demonstrated a relationship between prior programming knowledge and success in CS1 [31, 94]. We considered using a measure of prior knowledge as an alternative to prerequisite grades but did not do so because the data is a burden for instructors to collect, and because collecting accurate data requires asking students to solve programming-related questions that might be intimidating to those lacking prior knowledge. An alternative to explore in the future may be APCS grades or high school math grades, but in our experience instructors rarely have access to such data.

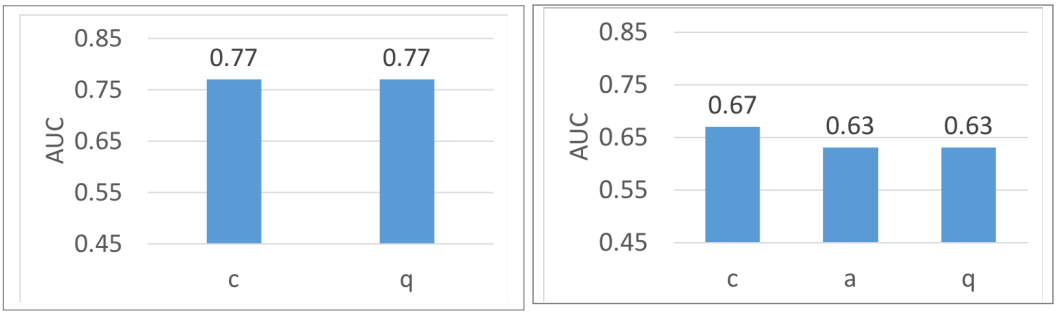
6.3.3 Threats to Validity

Although we included multiple courses, both lower- and upper-division, the data from the upper-division courses (Adv. DataStruct and Architecture) were collected only at a single institution. That said, these upper-division courses were taught by different instructors, so our results are not biased to a single instructor. In addition, we used logistic regression as it provided

strong prediction accuracy for a binary outcome given only a few features per student. To arrive at that modeling technique, we tried a variety of models commonly used in machine learning (e.g., SVM and random forest) and selected logistic regression based on its consistently strong results. As our findings are based on using logistic regression, our results may be biased by our choice of modeling technique.

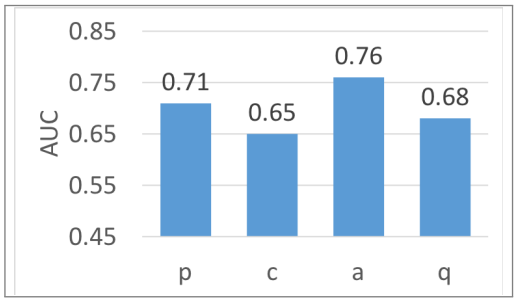
6.4 Acknowledgements

This chapter, in full, is a reprint of the materials as they appears in Proceedings of the 50th ACM Technical Symposium on Computer Science Education. S. N. Liao, D. Zingaro, C. Alvarado, W. G. Griswold, and L. Porter, ACM, 2019. The dissertation/thesis author was the primary investigator and author of this paper.

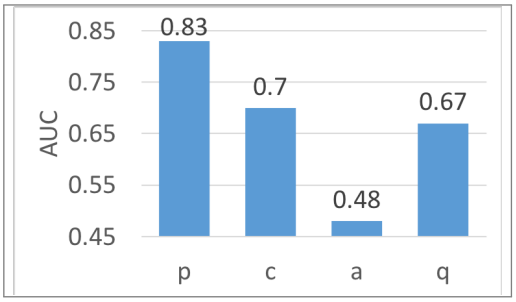


(a) CS1-Python

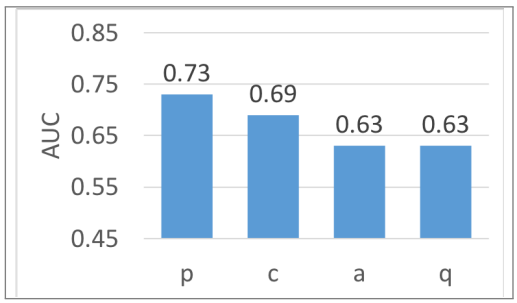
(b) CS1-Java



(c) CS2-Java

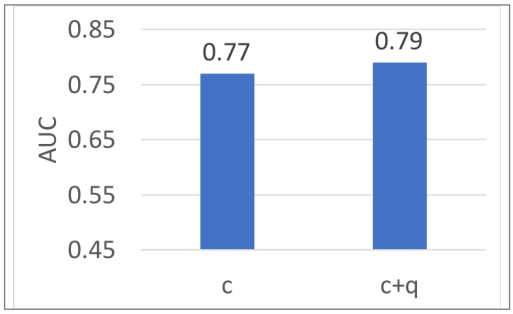


(d) Adv. DataStruct

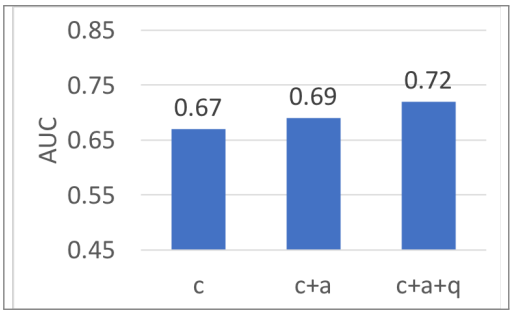


(e) Architecture

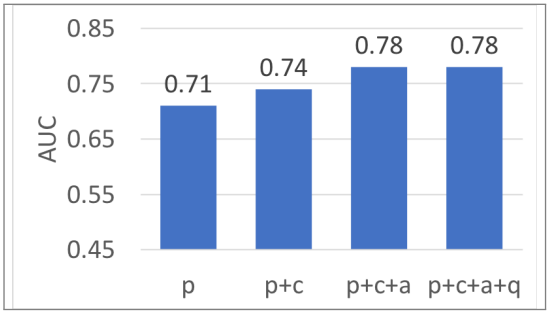
Figure 6.1: AUC When Modeling with Individual Data Sources. IDs are specified on Table 6.1.



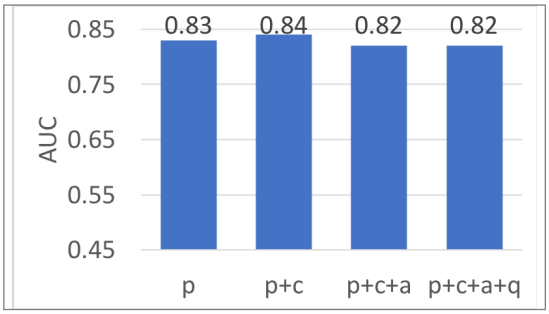
(a) CS1-Python



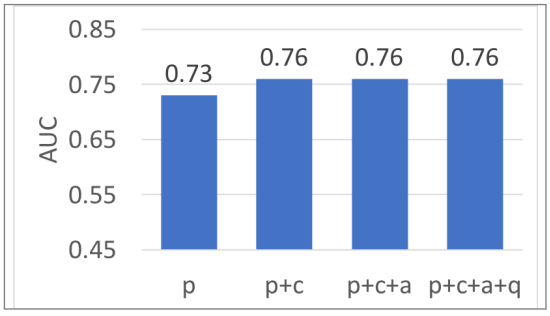
(b) CS1-Java



(c) CS2-Java



(d) Adv. DataStruct



(e) Architecture

Figure 6.2: AUC When Modeling with Combination of Data Sources

Chapter 7

Behaviors of At-Risk Students in CS1

Our goal was to discover behaviors that might be associated with higher or lower student performance in CS1 courses. We chose an open-coding technique, which iteratively groups and regroups related fragments of interview transcripts in order to identify themes and uncover patterns in the data [28, 47]. Also, we collected survey responses to examine whether the observations from the interviews extrapolates to a larger group of students. Below, we describe our data collection procedure, interview structure and analysis process, survey structure and its analysis.

7.1 Methods

7.1.1 Course Context

We selected an introductory CS course ("CS1") at a large North American research-focused university as the context for our study. Students from two terms (fall and winter) participated in the study. This 10-week CS1 course is designed for CS majors, intended CS majors, and non-majors alike. It uses a media computation curriculum to teach students basic programming in Java [30]. Table 7.1 lists the main components of the course; we refer to these components throughout the paper.

Table 7.1: Components of the Studied Course

Component	Description
Lecture	Class with clickers and Peer Instruction. [61]
Podcast	Recording of class, posted online.
Discussion	Extra weekly lecture run by graduate student TAs. Attendance optional.
Office Hours	Held by undergraduate tutors, graduate TAs, or the instructor.
Piazza	Online question board and discussion forum.
Reading Assignment	Weekly textbook-reading assignment for lecture preparation.
Lab	Weekly programming practice session led by tutors.
Lab Quiz	Online quiz given at the end of each lab.
Review Quiz	Weekly online quiz for the week's material.
PSA	Weekly programming assignment.
Star Points	Optional extra credit assignments for each PSA.
Practice Midterm	Midterm exam questions from a previous term, provided by the instructor.
Midterm	50 minute In-class exams; held in week 4 and 7.
Final	Summative Assessment in Week 11.

7.1.2 Interview Participants

Students were recruited to participate in interviews, as approved by our institution's Human Subjects Review Board. Students were compensated for their time with gift cards. During the first term, we interviewed 8 students. We then used the data from these interviews to refine the interview process, and interviewed 19 students in the second term.

First Term

The first round of our study began in the middle of the term. The students had by that time completed their first midterm and received grades and feedback on it. The course instructor provided the midterm exam rankings of the 441 consenting students to a research colleague, who then helped us invite students (124) to participate. These 124 students were selected such that at least some portion were in the bottom 40% of midterm grades. Eight of these students agreed to participate in the interviews. To avoid unintentionally influencing the interviews, the interviewers

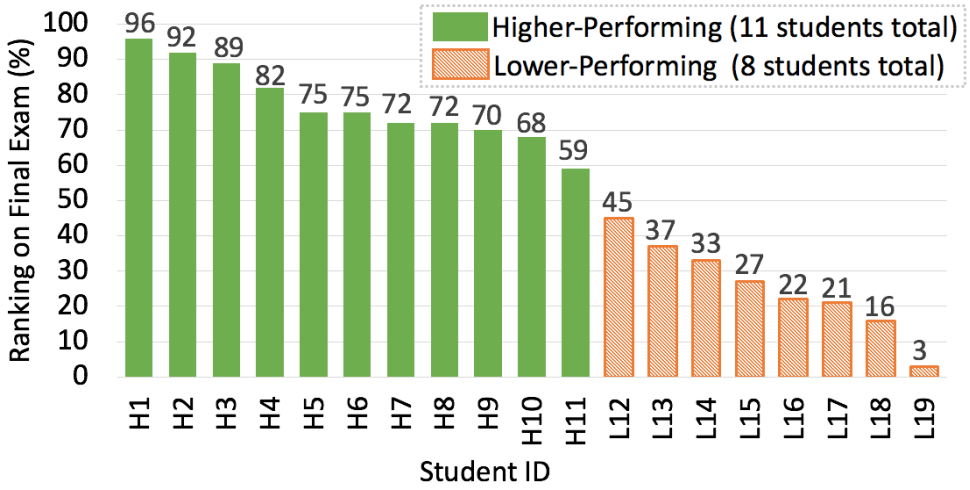


Figure 7.1: Interviewee Performance on Final Exam

were unaware of which students did poorly on the midterm.

Second Term

In the second round of the study, we began recruiting for the interviews before students took the first midterm, so we needed a different way of selecting a representative sample group. When recruiting students to participate in the study, we asked those interested to apply via an online survey. Part of that survey asked students to report the grade they were aiming to get in the course. 45 out of approximately 400 students responded to the survey. Because most students reported aiming for an "A" in the course, we first selected all students that aimed for a grade less than "A" (5 out of 45). One of those students did not respond to our request to schedule an interview. Students were then selected randomly until our participant pool reached 20 confirmed students (though one student dropped the class prior to the interview). Again, the interviewers did not inquire about actual student grades directly (though students sometimes volunteered the information) until after interviews were completed.

Our question is different from more usual way of asking - the grade they expect. We note that student responses to our question could depend on other factors such as motivation or self-efficacy. Although the grade a student is aiming for does not necessarily reflect their

performance, we felt that it would help us diversify the achievement levels of students in our pool. Indeed, as shown in Figure 7.1, the approach seemed successful.

7.1.3 Categorizing Performance

We used final exam grades to divide students from the second term into two categories after the interviews: Higher-Performing (HP) and Lower-Performing (LP). Students who received final exam scores in the top 50% of the class were considered high performers, while those in the bottom 50% of the class were considered lower performers. This threshold reflects the way that Ahadi et al. defined at-risk and not-at-risk students [5]. We recognize that performance in a course is on a continuum; though we have used a binary classification in our analysis, we also include the rank of each student (among our interviewees) to help provide readers with a more nuanced perspective of performance.

7.1.4 Interviews

First Term

During the first term, we conducted interviews shortly after the first midterm and follow-up interviews after the second midterm. Starting one week before the second midterm, students were asked to keep a timetable. In this table they recorded the date, start time, finish time, what they studied, and the difficulty of the studied material (1-5, where 1 is very easy and 5 is very hard). Additionally, students were asked to take a picture of their work environment every time they filled in the table. During interviews, we used the timetable and pictures to help the students recall what exactly they were studying at a specific point in time, similar to Fincher et al.'s My Programming Week [27]. Finally, students were asked to bring all their study materials (e.g., laptop, notes, textbook) to the interview. Two authors interviewed the students; these authors were not associated with the course in any way.

Second Term

Three interviews were conducted for each student: after the first midterm, after the second midterm, and shortly before final exams. The main focus of our analysis is the first interview, as this interview captures student behaviors early in the term. To prepare for the first interview, students were asked to keep a timetable during the week prior to the first midterm. Students were again asked to take pictures of their study area and bring all their study materials to the interview. The interviews were semi-structured, with some prompts originating from our interview experience in the first term. Examples of these prepared questions include *"How is your term going so far?", "How did you prepare for the midterm?", "Do you usually begin your programming assignments on this day?", "How did you fix this bug here in your code?",* and *"Can you explain this method?"*. The same two authors from the first term interviewed the students and again were not associated with the course.

7.1.5 Interview Analysis

Analysis was performed by two of the authors. We first transcribed the interviews. Next, we chose two student interviews from the first term, segmenting their transcripts into topical components. We printed these snippets and worked together to assign codes to them. After this session we began coding the rest of the interviews. We iteratively improved our coding rubric and validated each iteration by having the two authors independently code at least 10% of our data (two interviews) and subsequently measuring the inter-coder reliability. After 3 total iterations of independently coding interviews and further refining the coding rubric, we had reduced our initial rubric from 111 codes to 41 codes and achieved an inter-coder reliability of 87% agreement.

7.1.6 Student Surveys

We designed the survey to confirm the findings from the interview with a larger group of population. The survey consists of 31 multiple-choice questions and 28 of them required 5-point Likert scale responses. The rest three had categorical responses. The full list of the questions is provided in Appendix of this chapter. The question statements were generated using the word choices of the interviewees so that survey respondents could find the statements familiar.

The survey was given out to CS1-Java course at a North-American R1 institution before the first midterm exam. 260 students took the first midterm exam and 244 students out of them completed the survey (93.8% response rate). Students were asked to submit the survey before the exam so that their responses were not influenced by the exam.

We ran multiple linear regression analysis in R using lm( ) to investigate how each survey question contributes to the midterm exam score. Based on the p-values of each question's coefficients, we kept more contributing questions and re-calculated their coefficients. We used p-value of F-test to measure the goodness of the fit.

7.2 Results

In this section, we describe our observations of how students prepare for the first midterm and work on programming assignments, including their particular study practices, help-seeking behaviors, willingness to correct identified gaps and confusion, and whether they continue to seek out answers to questions even after deadlines. We refer to students as "H" (Higher-Performing) or "L" (Lower-Performing) along with their final exam rank relative to other interviewees (e.g., 11 out of 19).

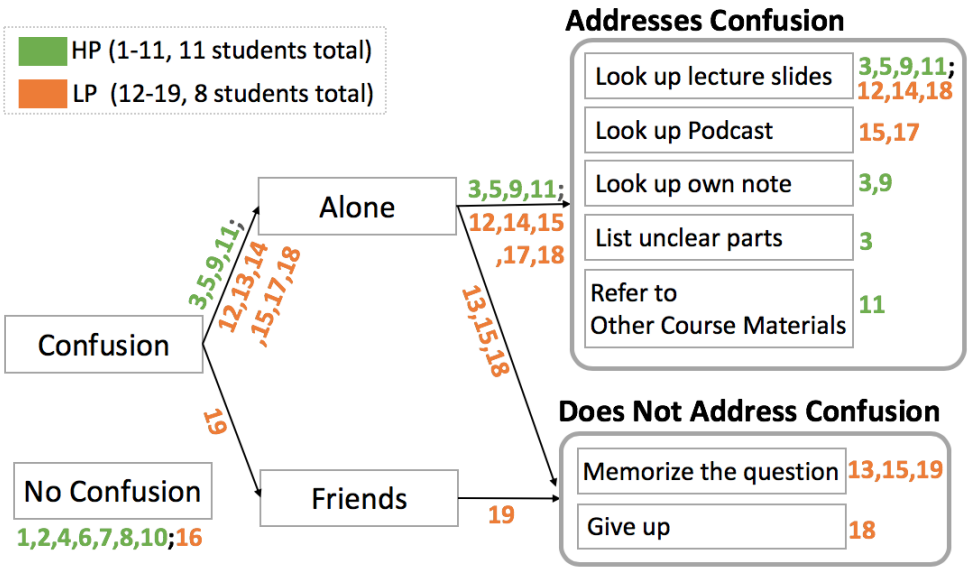


Figure 7.2: How Students Address Confusions Encountered on Practice Midterm (numbers represent student identifiers)

7.2.1 Midterm Preparation

All but four students started midterm preparation by working on the practice midterm. Two of those four students did not recall how they started their preparation. The third of the four, H11, used the practice midterm after reviewing other materials to assess whether they were prepared for the actual midterm. However, H11 identified some difficulties from the practice midterm, so they followed the same study pattern as the rest of the participants after taking the practice test: *"...I just got mistakes on, referencing [from the practice midterm], which showed I needed to study more on referencing" [H11].*

Figure 7.2 shows our codes for how students addressed concerns on the practice midterm while studying for the actual midterm. Seven HPs and one LP (L16) identified no significant difficulty on the practice midterm. Although they did not necessarily get every question entirely correct, they said they made only simple mistakes. For example: *"But they were just like ... dumb in my mind. I think one of them was uh, it asks 'What file is the, turtle class in?' And then I was like, 'Oh, isn't that dumb? It's in the turtle class.' But no, it's in the turtle.java, so I was like*

...'Oh, of course' " [H6].

Of the remaining 11 students, seven students reported resolving all of the difficulties that they identified (H3,5,9,11; L12,14,17), while the other four (L13,15,18,19) did not. All HPs who identified challenges succeeded in overcoming those challenges, while less than half of the LPs succeeded.

The following provides details on the four categories we identified in which HP and LP behavior differed.

Memorize or Give Up

The 11 students who identified confusion using the practice midterm worked to resolve it in different ways. As seen in Figure 7.2, the LPs who did not address their confusion either memorized the solution to relevant practice midterm questions (L13, 15, 19) or gave up, hoping it would not be on the exam (L18): *"I just redid the practice exam properly to try and see how much of it I actually did remember how to do. That's kinda just standard procedure with me when I'm given a practice test" [L13]. "...the eight different kinds of primitives was actually on the midterm but...I didn't memorize them because I thought we didn't have to...[when studying them]" [L18].* Moreover, L15 mentioned making note cards for memorization: *"'Okay, I understand this problem,' and then just save that note card for in the future, for hypothetical case if I forgot how to do a certain problem..." [L15].*

L19 also exhibited a behavior of focusing excessively on getting the right answer, as opposed to gaining understanding (L19 was in the bottom 3% on the final exam): *"I would just sit there for a while trying to figure it out. And whenever I thought I finished, I'll get my friend to check it, and then he would tell me yes or no. And if he said no, then I would go erase it and do it again." [L19].*

Resources Used

As shown in the Addresses Confusion box in Figure 7.2, students used different resources when studying for the midterm and attempting to address their confusion. It was common for students to refer to lecture materials, with most examining the lecture slides. Only two students (L15,17), both LPs, watched the video podcasts of the lectures. One of those (L17) went so far as to watch multiple lectures back-to-back: *"I just sat on my bed, and put on the TV and then I sped them up a little and just listened to them...I skipped the first beginning ones because they were pretty straightforward, and then I picked some of the middle ones...then went up to whichever one before the midterm" [L17].*

Beyond lecture material, HPs availed themselves of additional resources such as their own notes, the textbook, quiz questions, and the Internet: *"...I look at how some functions the textbook asks us to do, or...in the slides, [but] some idea that I want to understand is not covered in the slides, I'm trying to refer to my notes to get the extra information" [H9].* H3 also took the time to list the things they were not sure about and reviewed it right before the midterm exam: *"If I missed something, like I didn't know this, sometimes I write it down...It was all just like, random notes...[and I read it in] like the morning before the midterm" [H3].* H11 generated and practiced their own questions to strengthen their understanding: *"So after finishing the test in, like, 30 minutes, I think I did, or 20 minutes. I just went back and studied references again...I went onto the Internet, and I learned it a little bit...And then I used the textbook. [Also] I just keep on trying to make new questions and see if it was right or not" [H11].*

Friends Involved

While all HPs and most LPs worked on the practice midterm on their own, two LPs (L14, 19) worked together with a friend (Friends box in Figure 7.2). However, both reported that their friend did not end up being helpful. L14 reviewed alone after the meeting with the friend to resolve confusion, and L19 did not resolve their confusion. L19 acknowledged that tutors would

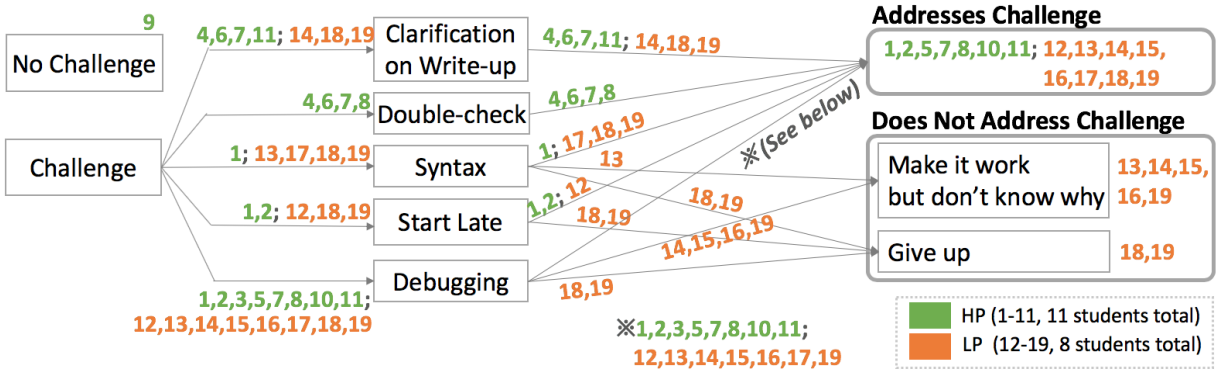


Figure 7.3: How Students Respond to Challenges during Programming Assignments (numbers represent student identifiers)

explain an answer in more detail, but ended up not going to tutor hours for the practice midterm because they were working on it at the last minute: *"one of my suite mates is also a CSE major. And I will just ask him. I mean, but the thing is that...he will just tell me how to do it. He doesn't teach me, just says...'This is how you do it.' I'm like, 'Okay.' But I don't know what this means. And that's why I'll go to the tutoring ... to be like, 'Uh, what does this mean? I don't get it"' [L19].*

After-exam Behavior

H1, whose final exam score was in the 96th percentile, worked to resolve a confusion after the midterm exam. No other students described such behavior: *"...I believe it's a problem about reference. I asked different tutors and they give me different answers. I just feel confused so I double-check it with my professor yesterday, even after the midterm" [H1].*

7.2.2 Programming Assignments

Figure 7.3 illustrates student behaviors when overcoming challenges during programming assignments. Every student except for H9 reported encountering challenges while working on programming assignments. The following subsections discuss each of the constitutive boxes of

the figure, specifically the challenge they faced and whether they were successful in addressing the challenge.

Planning

Two HPs and three LPs started programming assignments quite late, a day or less before the deadlines. However, the HPs who reported starting late were the two strongest-performing HPs (H1,2), did not show any concern about starting late, and were able to complete the assignments successfully. On the other hand, two of the three LPs were the bottom-ranked LPs and bemoaned the lack of help available at the last minute. One of those two, L19, started assignments on the due date or the day before the due date. As such, the student had no choice but to work on the assignment until late at night: *"when I'm programming and there's an error, um, usually...it's late at night, so I have no one really to contact. So I just sit there for a long period sometimes trying to figure it out" [L19]. "Yes because I couldn't get my program to work properly, so...and there was no tutors, any help, because it was like almost last minute..." [L18].*

These two LPs also mentioned that they thought of posting questions on the Piazza discussion board, but did not feel that they had enough time to wait for a tutor's response. Therefore, they felt they had to either go back to campus to ask questions to a random classmate or just give up on the problem: *"... once I asked the question but didn't respond till the next day. But...I can't wait that long. So I just went to the lab...asked someone, sitting there, 'Do you know how to do this?"' [L19]. "considering like the time that I was working on it, I didn't ask on Piazza because I didn't expect a response by then... I worked on it too late so that was kind of the setback on being able to finish this" [L18].*

However, contrary to the rest of the LP students, L12 did not seem to worry about starting programming assignments late. Later, we noticed that L12 had a classmate who worked on the programming assignments with them, and L12 thought the classmate was much smarter than them. Based on L12's comments, it seemed likely that they were overdependent on their programming

partner: *"... a lot of the times, I start on Saturday [the day before the due date] 'cause if I don't understand it the first day, I'll take a step back and come back the due date. And try and just do it because the assignments usually don't take that long."* After a long pause they continued, *"He's a lot smarter than me, 'Cause he has prior experience, but... I didn't really know what he was saying. So then, he wasn't that far into it, so we just started working together. He explained to me this part, and then, we just kind of did the rest of it together..." [L12].*

Debugging Challenges

All except H4, 6, and 9 brought up debugging experiences (i.e., fixing errors in the code) that were challenging to them. The majority of the LPs (L13,14, 15,16,18,19) were unable to resolve all of their debugging challenges. Once they were able to get their code to work, sometimes with help from tutors, they reported moving on without understanding why it worked. In some cases, the code was deemed to work because it appeared to have the desired effect or it passed test cases provided by the instructor: *"I don't know why it's going negative but my tests are passing. The whole time I was just trying to make my test pass. And when I finally got them passed, I was like, 'I shouldn't change anything.' 'Cause I feel like it's all gonna go down...the drain if I do. So I just left it as is...I turned it in" [L19].* When L14 was asked whether the friend's advice helped, they answered *"Yes. I think it did...We were able to somehow change it to make it actually display these values" [L14].*

Moreover, when we asked the LPs to explain a part of the code that they debugged with help from tutors or friends, some of them were not able to explain it at all. When L15 was asked about a return statement in their code, they replied *"Why I need that line? Um... It's similar to using the System.out.println()... is like the way I remember it... Okay, this was kind of confusing..." [L15].*

Syntax Challenges

H1 and four LPs (L13,17,18,19) revealed challenges on the correct usage of the Java language and methods. H1 had some initial difficulty with boolean operators, because the instructor had not covered that material yet. However, H1 figured it out by reading a web page: *"...at that time, we didn't learn this operator like —— ...So I just look it up online, so I feel like, we need to use some concepts we never learned before...That's a challenge, I think" [H1].*

On the other hand, most LP students struggled with what they had already been taught in class. They either went to tutor hours to ask for help, or did not try to understand the problematic material: *"It was mostly questions that I was confused about. Like...arrays at the time. I knew what an array was... but I didn't understand how to code it" [L19]. "I wasn't sure why and how it [array indexing] is formatted that way...I was just a little lazy looking through the book so I didn't really understand it...it was probably part of the starter code, or I just looked at the book and just copied off it" [L13].*

Double-check Answers

Only HP students (H4,6,7,8) confirmed their answers with others—knowledgeable classmates or tutors—before they submitted their code: *"I talk with them...they're pretty knowledgeable in CS, too...It's more of just like, 'Hey, what did you do? Okay, I did this, too'... Like we both did it, we're just going over like, how we did it. Or if we're even doing it right...to make sure" [H6]. "I asked the TA, 'Oh, is this good?' But she told me, 'You gotta change it,'...so I had to re-change it...when the deadline comes up, I go to the lab. So, you know, just to make sure" [H7].*

Post-deadline Behavior

H2 did the Star Point assignment on their own: *"For PSA 2, um, it [the Star Point assignment] was a research into other object-oriented programming. That one, um, I didn't have enough time to actually start it, so I didn't do that one, either. Though, I did, like, afterwards*

[after the PSA deadline], when I had more time...I wouldn't be able to, like, uh, submit anything like that, but it was interesting to read on like, Python with how they do object-orientating" [H2]. On the other hand, L16 knew that they lost points from a submitted programming assignment and remembered the tutor's comments, but they did not understand why and did not try to address it: *"they said 'Even though you called the Math.abs(), you should put it in the while condition.' ... Eh, um, can I just say 'I don't know'?" [L16].*

7.2.3 Survey Analysis

Multiple linear regression analysis shows that the coefficients of only 6 survey questions are statistically significant. They are q2, q8, q12, q26, q27, and q31. Thus, we generated the final linear regression model with only those 6 questions. The p-value from F-test were below 0.05, showing that the goodness of fit is sufficient. The resulting coefficients from the regression analysis are on Table 7.2. The coefficients of q2, q8, q12, and q26 are negative numbers. This indicates that the more frequently students do those behaviors, the lower midterm exam scores they would get. On the other hand, the behaviors described in q27 and q31 contributes in raising the midterm exam score. In addition, the magnitude of q31 were the highest, which means it impacts the midterm exam score more than the other questions.

7.3 Discussion

7.3.1 Implications from the Interviews

Table 7.3 serves to summarize our findings from Section 7.2, focusing on the trends that emerged from our analysis. For each trend in the table, we refer back to the appropriate subsection in Section 7.2 for more details. Some of the observed trends may relate to more general (non-CS specific) study skills (e.g., HP students tend to seek out extra resources to help clarify material and

Table 7.2: Coefficients of Resulting Linear Regression Model (Coef.: coefficients, *: $p < 0.05$)

ID	Question Statement	Coef.
q2	When I am unclear about the course material, I ask my classmates or friends for help. (ex: basement lab in the CSE building, my apartment, library)	-0.077*
q8	When I am unclear about the course material, I use my notes.	-0.100*
q12	To prepare for the midterm, I go to midterm review sessions or watch the podcasts of those sessions.	-0.125*
q26	When encountering a piece of code that I need to understand, I simply run the code on a computer.	-0.153*
q27	When I encounter a piece of code that I need to understand, I predict the output first, and then run the code to check my understanding.	0.201*
q31	When I solve a programming assignment, I make sure I understand all the code I turn in (including any code given to me by TAs/tutors when I asked for help).	0.294*

resolve challenges). Similarly, some of these trends are tied to general student metacognition [74] (e.g., LP students may perform untargeted reviews of course materials before exams because they are unclear about where their conceptual weaknesses lie). Lastly, there are behaviors that clearly relate to student self-regulation [43] (e.g., HP continuing to work on assignments even after deadlines to ensure they fully understand course concepts, and the role of procrastination on each group). These trends may be encouraging to the computing education community insofar as we may be able to leverage literature outside of computer science to improve the ways that students study.

Some concerning behaviors do appear to be more computing-specific: assembling a programming assignment solution from bits contributed by friends and instructional staff, stopping work when the right answer is in hand before understanding why it is correct, or emphasizing memorization of code before an exam. These behaviors may be due to students' unfamiliarity with the field of computer science. They may simply be unaware that some behaviors, such as memorization of prior code solutions, may not be effective study strategies. Here, the computing education community can seek ways to better inform students of productive and unproductive behaviors specific to computer science.

Table 7.3: Summary of Differences Encountered Between Student Groups

	Higher-Performing	**Lower-Performing**
Exam study strategy	• execute targeted review of areas of perceived weakness (4.1.2) • create new questions to solve (4.1.2) • review course notes	• execute untargeted review (reviewing everything) (4.1.2) • memorize existing code (4.1.1)
Getting help	• solicit help, successfully addressing their challenges (4.2.3, 4.2.4)	• solicit help, only sometimes addressing challenges (ask friends for help, 4.1.3)
Addressing confusion	• refer back to their own course notes for answers (4.1.2) • seek out extra resources (4.1.2) • double-check solutions with others (4.2.4) • keep notes on areas of confusion (4.1.2)	• give up if cannot find answer (4.1.1, 4.2.1) • get correct answers from others, but do not seek to understand why the answers are correct (4.1.1) • through debugging, can get the code to work but may not understand why (4.2.2, 4.2.3)
Post-deadline behaviors	• will continue working on assignments (4.1.4, 4.2.5)	–
Procrastination	• when they do, it does not impact them significantly as they can still complete assignments (4.2.1)	• when they do, they are unable to find help when they need it to overcome problems (4.2.1)

We also note that although some of these behaviors may result from lack of prior experience, we believe that many of the student struggles are tied to CS-specific and general study strategies. As such, we are encouraged that possible interventions might be successful without necessarily relying on background knowledge.

7.3.2 Extrapolating Interview Findings

q2, q26, q27, and q31 showed consistent results compared to what we discovered from student interviews. Asking friends for help was more common among Lower-performers according to the student interviews (q2). q26, q27, and q31 are asking whether students make sure they truly understand given assignments. As we observed during the interviews, higher-performers tend to ensure they understand the given code instead of simply running the code to check the

output of the code (q26). Thus, the question describing the opposite behavior, q27, showed the positive coefficient.

q31 impacts the most on the midterm exam score in terms of the magnitude of the coefficient. Although we intended to measure students metacognitive skills, those who were running short on time to complete the assignments might have given low Likert scale scores. This means, the responses to this question might include time management factor as well.

q8 and q12 contradicted what we observed during the interviews. The survey responses indicates that using notes negatively impacts a midterm exam score (q8), while the interviewees who are higher-performers tend to utilize their own notes more than those who are lower-performers. This shows that using notes itself may not be a good study strategy but how to utilize notes matter more. The way students use their notes needs further qualitative study. For q12, we believe the mismatch between the survey and interview results are from our word choices on the question statement. The question statement did not clarify a type of review activity between a targeted or untargeted review. Considering that from the interviews higher- and lower- performers tend to perform targeted and untargeted review respectively, we believe q12 did not function as we originally intended.

7.3.3 Next Steps and Possible Interventions

An ongoing next step to this work is to determine how common these behaviors are among students. This would allow us to craft interventions that target the most common (or most problematic) behaviors.

In the meantime, based on the results from Section 7.2, we argue for two features of interventions designed to help LPs that we believe may be most effective when applied in concert. First, help students to focus on achieving conceptual understanding rather than understanding particular solutions. While HPs tend to invest effort in clarifying and confirming their understanding (H3,9,11 in Section 7.2.1), LPs did not do so (L13,15,18,19 in Section 7.2.1),

simply memorizing solutions without understanding the underlying concepts (Section 7.2.1 and the last two quotes of Section 7.2.2). Thus, the instructional team should keep in mind that there may be a mismatch between perceived and actual student understanding. We therefore argue for the importance of confirming conceptual understanding with students, rather than relying on student self-report.

Second, and perhaps more obviously, help students to be aware of deadlines and plan accordingly. Prior literature suggests that procrastination leads to lower grades on programming assignments [45]. We similarly find that effects of poor study practices appear to be amplified when students procrastinate. For instance, L19 could not ask for help to understand the concepts behind the practice midterm because they started working on it at the last minute. Thus, the instructional team might consider an intermediate deadline to encourage students to start earlier, or use email reminders to encourage early participation [45]. Similarly, exam review sessions might be scheduled well before the exam, rather than the day before. Ultimately, we believe these recommendations will be most effective when combined, as starting earlier will likely be ineffective unless one also abandons poor study strategies like code memorization.

7.3.4 Threats to Validity

Our interview participants were all from a single CS1 course, and our sample size is small, so findings may not generalize to students in this or other CS1 courses. For instance, if another CS1 course does not offer practice midterm questions, the observations we found might not be applicable. In addition, we are relying here on self-reported data, though we did ask students to log their behavior and take pictures of their work environment to increase reliability. Furthermore, we moved on to the next topic if we noticed, or students themselves mentioned, that they could not accurately reply to our prompt. Lastly, even though we sought not to impact student behaviors, students might have behaved differently during the term as a result of any reflection hatched by participation in our study.

7.4 Appendix

The full list of the survey questions are as below. Everything but three questions had 5-point Likert scale answer choices: Never, Rarely, Sometimes, Usually, and Always. The answer choices of other three questions is provided for corresponding questions.

• **q1**: When I am unclear about the course material...I ask the instructional staff for help.

• **q2**: When I am unclear about the course material...I ask my classmates or friends for help. (ex: basement lab in the CSE building, my apartment, library)

• **q3**: When I am unclear about the course material...I ask Piazza for help.

• **q4**: When I am unclear about the course material ...

• **q5**: When I seek help on the course material from classmates or friends instead of instructional staff, it is mostly because...

• **q6**: When I post a question on Piazza to seek help, it is mostly because...

• **q7**: When I am unclear about the course material...I refer to the textbook, slides, or podcasts.

• **q8**: When I am unclear about the course material...I use my notes from class.

• **q9**: When I am unclear about the course material...I use external sources (ex: search online, another book, course material from another course).

• **q10**: To prepare for the midterm...I try to memorize facts from the course material.

• **q11**: To prepare for the midterm...I try to memorize the code I have seen before in this class.

• **q12**: To prepare for the midterm...I go to midterm review sessions or watch the podcasts of those sessions.

• **q13**: To prepare for the midterm...I review my notes from class.

• **q14**: To prepare for the midterm...I review the textbook, slides, or podcasts.

• **q15**: To prepare for the midterm...I review my prior programming assignments.

• **q16**: To prepare for the midterm...I refer to external sources (ex: search online, another book, course material from another course).

• **q17**: To prepare for the midterm...I practice questions from sample exams or quizzes.

• **q18**: To prepare for the midterm...I practice questions from external resources outside the class.

• **q19**: To prepare for the midterm...if I don't know how to solve a practice problem, I try to recall a similar question I worked on before to see how I solved it.

• **q20**: To prepare for the midterm...I try to figure out what types of exam questions there will be by using sample exams.

• **q21**: To prepare for the midterm...I try to make up different kinds of exam questions.

• **q22**: To prepare for the midterm...I practice writing actual code on paper to get used to writing code on paper during the actual midterm.

• **q23**: To prepare for the midterm...I write some new code to implement new features of prior assignments.

• **q24**: To prepare for the midterm...I write code to solve problems I come up with.

• **q25**: When I encounter a piece of code that I need to understand...I read through the code and try to predict the output of the code.

• **q26**: When I encounter a piece of code that I need to understand...I simply run the code on a computer.

• **q27**: When I encounter a piece of code that I need to understand...I predict the output first, and then run the code to check my understanding.

• **q28**: When I encounter a piece of code that I need to understand...I compare it to a similar piece of code that I already understand.

• **q29**: Additional Strategies:I help my classmates or friends understand concepts because it helps me understand those concepts better as well.

• **q30**: Additional Strategies:I form informal study groups with classmates.

• **q31**: Additional Strategies:When I solve a programming assignment, I make sure I understand all the code I turn in (including any code given to me by TAs/tutors when I asked for help).

7.5 Acknowledgements

This chapter, in part, has been submitted for publication of the material as it may appear in Proceedings of the 24th ACM Annual ACM Conference on Innovation and Technology in Computer Science Education. S. N. Liao, S. Valstar, K. Thai, C. Alvarado, D. Zingaro, W. G. Griswold, and L. Porter, ACM, 2019. The dissertation/thesis author was the primary investigator and author of this paper.

Chapter 8

Conclusion

This dissertation investigates how to identify at-risk students early in the term using a variety of data and analyzes behavioral differences between higher- and lower- performing students through a series of interviews.

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-degree.

The current increase in student enrollment makes if more difficult for instructors 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. This dissertation develops a modeling methodology that uses such data, specifically student clicker responses naturally collected in the PI classroom setting, to make predictions of student performance in a variety of computer science courses.

The initial work (Chapter 4) proposes a lightweight modeling technique based on prior-term data to identify low performers in a CS1-Python course. It also suggests that should an

instructor not use PI, simply asking the same multiple-choice questions before or after class could produce similarly accurate results. At the heart of this prediction methodology is the instructor's threshold for classifying a student as "at-risk." As such, they can choose to trade off overestimating the number of at-risk students to ensure they reach everyone in need or underestimating the number of at-risk students to avoid spending course resources on those who may not need help. Our approach results in a model that accurately predicts 70% of students in the test set as either needing or not needing assistance. We also show that we are able to more accurately predict students who attend class more frequently and that the important topics identified by the model can inform our view of the early weeks of CS1.

The follow-up work in Chapter 5 proposes 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.

Chapter 6 analyzes the predictability of different data sources and how prediction performance improves when combining multiple data sources. We collected students' prerequisite course grades, clicker correctness, assignment grades, and online quiz grades, and applied logistic regression to produce cross-term predictions (as one might do to authentically predict in practice). Overall, our results demonstrate that prerequisite course grades (when available) predict student performance most strongly, followed by clicker data, assignments, and online quizzes. Combining prerequisite and clicker data often improves the model's prediction accuracy; however, adding assignments and quizzes to that already accurate model often does not increase the accuracy further.

This work adds to our knowledge of predicting outcomes in computing by comparing different data sources collected from the same population to determine which are most meaningful. The data sources selected for the work are those generally available to instructors, reducing the

barrier for instructors to adopt these techniques. We note also that prerequisite and clicker data are available early: prerequisite data before the start of class, and clicker data from the first lectures (though several weeks of clicker data are likely required for accurate modeling). The findings are generally consistent across five different computing courses (including both lower- and upper-division courses) and are collected from two institutions. Overall, our findings suggest that instructors wishing to identify low-performing students should leverage prerequisite course grades and clicker questions to make that determination.

Based on our results from Chapter 4 – 6, 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 paper 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 `peerinstruction.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. This is an avenue for future work.

- **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.

Chapter 7 presents an analysis of CS1 interviews in an effort to understand the range of behaviors demonstrated by students. We distinguish between the behaviors of higher- and lower-performing students to investigate whether there are any notable trends in behaviors within or between groups. Our interviews revealed differing behavioral characteristics of higher- and lower-performing students that may be valuable in crafting potential interventions. A survey was given out to a larger group of students and 6 out of 31 survey questions turned out to be statistically significantly contribute to students' midterm exam scores. Those questions showed that some observations from the interviews extrapolated to a larger group.

Refining and deploying the intervention strategies suggested in this paper in a real classroom setting is an important next step for this work. The CS education research community has invested considerable effort in high-accuracy identification of at-risk students (Chapter 2.1.4). While that work remains important, we argue for a corresponding focus on using those identifications to intervene and help these students. We hope that our qualitative investigation here, along with our initial suggestions stemming from that work, will further this important research agenda.

8.1 Acknowledgements

This chapter, in part, is a reprint of the materials as they appears in four publications: 1) Proceedings of the Conference on International Computing Education Research. S. N. Liao, D. Zingaro, M. A. Laurenzano, W. G. Griswold, and L. Porter, ACM, 2016; 2) Proceedings of the 50th ACM Technical Symposium on Computer Science Education. S. N. Liao, D. Zingaro, C. Alvarado, W. G. Griswold, and L. Porter, ACM, 2019; 3) ACM Transactions in Computing

Education. S. N. Liao, D. Zingaro, K. Thai, C. Alvarado, W. G. Griswold, and L. Porter, ACM, 2019; and 4) Proceedings of the 24th ACM Annual ACM Conference on Innovation and Technology in Computer Science Education. S. N. Liao, S. Valstar, K. Thai, C. Alvarado, D. Zingaro, W. G. Griswold, and L. Porter, ACM, 2019. The dissertation/thesis author was the primary investigator and author of these papers.

Bibliography

[1] https://www.rdocumentation.org/packages/lmtest/versions/0.9-36/topics/lrtest.

[2] https://www.rdocumentation.org/packages/stats/versions/3.5.1/topics/glm.

[3] Y. M. Abdulrazzaq and K. I. Qayed. Could final year school grades suffice as a predictor for future performance? *Medical Teacher*, 15(2-3):243–251, 2009.

[4] A. Ahadi, A. Hellas, and R. Lister. A contingency table derived method for analyzing course data. *Transactions on Computing Education*, 17(3):13:1–13:19, 2017.

[5] A. Ahadi, R. Lister, H. Haapala, and A. Vihavainen. Exploring machine learning methods to automatically identify students in need of assistance. In *Proceedings of the International Conference on Computing Education Research*, pages 121–130, 2015.

[6] A. Anthony and M. Raney. Bayesian network analysis of computer science grade distributions. In *Proceedings of the 43rd Technical Symposium on Computer Science Education*, pages 649–654, 2012.

[7] A. Bandura. Self-efficacy: toward a unifying theory of behavioral change. *Psychological review*, 84(2):191–215, 1977.

[8] S. Bergin and R. Reilly. Predicting introductory programming performance: A multi-institutional multivariate study. *Computer Science Education*, 16(4):303–323, 2006.

[9] A. S. Carter and C. D. Hundhausen. With a little help from my friends: An empirical study of the interplay of students' social activities, programming activities, and course success. In *Proceedings of the 12th International Conference on Computing Education Research*, pages 201–209, 2016.

[10] A. S. Carter, C. D. Hundhausen, and O. Adesope. Blending measures of programming and social behavior into predictive models of student achievement in early computing courses. *Transactions on Computing Education*, 17(3):12:1–12:20, 2017.

[11] A. S. Carter, C. D. Hundhausen, and O. O. Adesope. 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*, pages 141–150, 2015.

[12] J. M. Case. A different route to reducing university drop-out rates. The Conversation, 2015.

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

[14] N. Cengiz and A. Uka. Prediction of Student Success Using Enrollment Data. *Proceedings of the Workshops held at Educational Data Mining: Workshop Approaching Twenty Years of Knowledge Tracing*, 2014.

[15] S. P. M. Choi, S. Lam, K. C. Li, and B. T. M. Wong. Learning analytics at low cost: At-risk student prediction with clicker data and systematic proactive interventions. *Journal of Educational Technology & Society*, 21(2):273–290, 2018.

[16] A. T. Corbett and J. R. Anderson. Knowledge tracing - modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, 4(4):253–278, 1994.

[17] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[18] CRA Enrollment Committee Institution Subgroup. Generation CS: Computer science undergraduate enrollments surge since 2006. Computing Research Association, 2017.

[19] J. W. Creswell and C. N. Poth. *Qualitative Inquiry and Research Design, Fourth Edition*. Sage Publications, Inc, 2017.

[20] C. H. Crouch and E. Mazur. Peer instruction: Ten years of experience and results. *American Journal of Physics*, 69(9):970–77, 2001.

[21] B. S. Cynthia Lee, Leo Porter and D. Zingaro. Peer instruction for computer science. 2012.

[22] H. Danielsiek and J. Vahrenhold. Stay on these roads: Potential factors indicating students' performance in a CS2 course. In *Proceedings of the 47th Technical Symposium on Computer Science Education*, pages 12–17, 2016.

[23] M. de Raadt, M. Hamilton, R. Lister, J. Tutty, B. Baker, I. Box, Q. Cutts, S. Fincher, J. Hamer, P. Haden, M. Petre, A. Robins, Simon, K. Sutton, and D. Tolhurst. Approaches to learning in computer programming students and their effect on success. *Research and Development in Higher Education: Higher education in a changing world*, 28:407–414, 2005.

[24] L. Deslauriers, S. E. Harris, E. Lane, and C. E. Wieman. Transforming the lowest-performing students: an intervention that worked. *Journal of College Science Teaching*, 41:80–88, 2012.

[25] S. H. Edwards, J. Snyder, M. A. Pérez-Quiñones, A. Allevato, D. Kim, and B. Tretola. Comparing effective and ineffective behaviors of student programmers. In *Proceedings of the fifth international Conference on Computing education research*, pages 3–14, 2009.

[26] E. M. Elias and C. A. Lindsay. The Role of Intellective Variables in Achievement and Attrition of Associate Degree Students at the York Campus for the Years 1959 to 1963, 1968.

[27] S. Fincher, J. Tenenberg, and A. Robins. Research design: necessary bricolage. In *Proceedings of the seventh international conference on Computing education research*, pages 27–32, 2011.

[28] B. G. Glaser and A. L. Strauss. The discovery of grounded theory: Strategies for qualitative research. *Transaction Publishers*, 1967.

[29] K. Goldman, P. Gross, C. Heeren, G. L. Herman, L. Kaczmarczyk, M. C. Loui, and C. Zilles. Setting the scope of concept inventories for introductory computing subjects. *Transactions on Computing Education*, 10(2):1–29, 2010.

[30] M. Guzdial. A media computation course for non-majors. *SIGCSE Bulletin*, 35(3):104–108, 2003.

[31] D. Hagan and S. Markham. Does it help to have some programming experience before beginning a computing degree program? In *ACM SIGCSE Bulletin*, volume 32, pages 25–28, 2000.

[32] B. Hanks and M. Brandt. Successful and unsuccessful problem solving approaches of novice programmers. In *Proceedings of the 40th Technical Symposium on Computer Science Education*, pages 24–28, 2009.

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

[34] N. Ibrahim, S. A. Freeman, and M. C. Shelley. Identifying Predictors of Academic Success for Part-time Students at Polytechnic Institutes in Malaysia. *International Journal of Adult Vocational Education and Technology*, 2(4):1–16, 2011.

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

[36] D. James and C. Chilvers. Academic and non-academic predictors of success on the Nottingham undergraduate medical course 1970-1995. *Medical Education*, 35(11):1056–1064, 2001.

[37] P. Jensen and R. Moore. What do help sessions accomplish in introductory science courses? *Journal of College Science Teaching*, 38(5):60, 2009.

[38] M. Kuhn. Building predictive models in R using the caret package. *Journal of Statistical Software*, 28(1), 2008.

[39] C. B. Lee, S. Garcia, and L. Porter. Can peer instruction be effective in upper-division computer science courses? *Transactions on Computing Education*, 13(3):12:1–12:22, 2013.

[40] C. M. Lewis, K. Yasuhara, and R. E. Anderson. Deciding to major in computer science: A grounded theory of students' self-assessment of ability. In *Proceedings of the Seventh International Conference on Computing Education Research*, pages 3–10, 2011.

[41] S. N. Liao, W. G. Griswold, and L. Porter. Impact of class size on student evaluations for traditional and peer instruction classrooms. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '17, pages 375–380. ACM, 2017.

[42] S. N. Liao, D. Zingaro, M. A. Laurenzano, W. G. Griswold, and L. Porter. Lightweight, early identification of at-risk cs1 students. In *Proceedings of the International Conference on Computing Education Research*, pages 123–131, 2016.

[43] D. Loksa and A. J. Ko. The role of self-regulation in programming problem solving process and success. In *Proceedings of the 12th Conference on International Computing Education Research*, pages 83–91, 2016.

[44] A. Lucas. Using Peer Instruction and I-Clickers to Enhance Student Participation in Calculus. *Problems, Resources, and Issues in Mathematics Undergraduate Studies*, 19(3):219–231, 2009.

[45] J. Martin, S. H. Edwards, and C. A. Shaffer. The effects of procrastination interventions on programming project success. In *Proceedings of the Eleventh International Conference on Computing Education Research*, pages 3–11, 2015.

[46] R. McCartney, A. Eckerdal, J. E. Mostrom, K. Sanders, and C. Zander. Successful students' strategies for getting unstuck. In *Proceedings of the 12th Conference on Innovation and Technology in Computer Science Education*, pages 156–160, 2007.

[47] A. J. Mills, G. Durepos, and E. Wiebe. Coding: Open coding. In *Encyclopedia of case study research*, 2010.

[48] R. Moore. Who does extra-credit work in introductory science courses? *Journal of College Science Teaching*, pages 12–15, 2005.

[49] L. Murphy, G. Lewandowski, R. McCauley, B. Simon, L. Thomas, and C. Zander. Debugging: The good, the bad, and the quirky – a qualitative analysis of novices' strategies. In *Proceedings of the 39th Technical Symposium on Computer Science Education*, pages 163–167, 2008.

[50] National Center for Education Statistics. 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, 2016.

[51] M. Natrella. NIST/SEMATECH e-handbook of statistical methods, 2010.

[52] V. Osborn and P. Turner. Identifying At-Risk Students in LIS Distributed Learning Courses. *Journal of education for library and information science*, 43(3):205, 2002.

[53] A. Petersen, M. Craig, J. Campbell, and A. Tafliovich. Revisiting why students drop CS1. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, pages 71–80, 2016.

[54] C. G. Petersen and T. G. Howe. Predicting Academic Success in Introduction to Computers. *Association for Educational Data Systems*, 12(4):182–191, 1979.

[55] S. Pilzer. Peer Instruction in Physics and Mathematics. *Problems, Resources, and Issues in Mathematics Undergraduate Studies*, 11(2):185–192, 2001.

[56] L. Porter, C. Bailey Lee, and B. Simon. Halving fail rates using peer instruction: A study of four computer science courses. In *Proceeding of the Technical Symposium on Computer Science Education*, pages 177–182, 2013.

[57] L. Porter, C. Bailey-Lee, B. Simon, Q. Cutts, and D. Zingaro. 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*, pages 138–142, 2011.

[58] L. Porter, C. Bailey-Lee, B. Simon, and D. Zingaro. Peer instruction: Do students really learn from peer discussion in computing? In *Proceedings of the Seventh international Conference on Computing Education Research*, pages 45–52, 2011.

[59] L. Porter, D. Bouvier, Q. Cutts, S. Grissom, C. Lee, R. McCartney, D. Zingaro, and B. Simon. A multi-institutional study of peer instruction in introductory computing. In *Proceedings of the Technical Symposium on Computing Science Education*, pages 358–363, 2016.

[60] L. Porter, D. Bouvier, Q. Cutts, S. Grissom, C. Lee, R. McCartney, D. Zingaro, and B. Simon. A multi-institutional study of peer instruction in introductory computing. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 358–363, 2016.

[61] L. Porter, D. Bouvier, Q. Cutts, S. Grissom, C. Lee, R. McCartney, D. Zingaro, and B. Simon. A multi-institutional study of peer instruction in introductory computing. *ACM Inroads*, 7(2), 2016.

[62] L. Porter, S. Garcia, J. Glick, A. Matusiewicz, and C. Taylor. 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*, pages 129–134, 2013.

[63] L. Porter and B. Simon. 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*, pages 165–170, 2013.

[64] L. Porter and D. Zingaro. Importance of early performance in cs1: Two conflicting assessment stories. In *Proceedings of the Technical Symposium on Computer Science Education*, pages 295–300, 2014.

[65] L. Porter, D. Zingaro, and R. Lister. Predicting student success using fine grain clicker data. In *Proceedings of the tenth annual conference on International computing education research*, pages 51–58, 2014.

[66] L. Porter, D. Zingaro, and R. Lister. Predicting student success using fine grain clicker data. In *Proceedings of the International Conference on Computing Education Research*, pages 51–58, 2014.

[67] D. M. Powers. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.

[68] C. Ramos and E. Yudko. "Hits" (not "Discussion Posts") predict student success in online courses - A double cross-validation study. *Computers & Education*, 50(4):1174–1182, 2008.

[69] A. Robins, P. Haden, and S. Garner. Problem distributions in a CS1 course. In *Proceedings of the 8th Australian conference on Computing education*, pages 165–173, 2006.

[70] J. E. Roueche. Research Studies of the Junior College Dropout. *American Association of Junior Colleges*, pages 1–5, 1967.

[71] P. M. Sadler and R. H. Tai. Advanced placement exam scores as a predictor of performance in introductory college biology, chemistry and physics courses. *Science Educator*, 16(2):1–19, 2007.

[72] W. E. C. Sadler and F. L. K. Levent. Factors Affecting Retention Behavior: A Model To Predict At-Risk Students. *the Association for Institutional Research Annual Forum*, 1997.

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

[74] G. Schraw, K. J. Crippen, and K. Hartley. Promoting self-regulation in science education: Metacognition as part of a broader perspective on learning. *Research in Science Education*, 36(1):111–139, Mar 2006.

[75] S. Shaban and M. McLean. Predicting performance at medical school: can we identify at-risk students? *Advances in Medical Education and Practice*, Volume 2:139–148, 2011.

[76] K. Shakerdge. High failure rates spur universities to overhaul math class. The Hechinger Report, 2016.

[77] S. Sharif, L. Gifford, G. A. Morris, and J. Barber. Can We Predict Student Success (and Reduce Student Failure)? *Pharmacy Education*, 3:1–10, 2003.

[78] D. F. Shell, L.-K. Soh, A. E. Flanigan, and M. S. Peteranetz. Students' initial course motivation and their achievement and retention in college CS1 courses. In *Proceedings of the 47th Technical Symposium on Computer Science Education*, pages 639–644, 2016.

[79] B. Simon, S. Esper, L. Porter, and Q. Cutts. Student experience in a student-centered peer instruction classroom. In *Proceedings of the Ninth International Conference on Computing Education Research*, pages 129–136, 2013.

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

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

[82] L. D. Singell and G. R. Waddell. Modeling Retention at a Large Public University: Can At-Risk Students Be Identified Early Enough to Treat? *Research in Higher Education*, 51(6):546–572, 2010.

[83] M. K. Smith, W. B. Wood, W. K. Adams, C. E. Wieman, J. K. Knight, N. Guild, and T. T. Su. Why peer discussion improves student performance on in-class concept questions. *Science*, 323(5910):122–124, 2009.

[84] A. Smola, K. Hornik, A. Zeileis, and A. Karatzoglou. Kernel-based machine learning lab. 2003.

[85] Y.-S. Su, A. Gelman, J. Hill, and M. Yajima. Multiple imputation with diagnostics (mi) in R: Opening windows into the black box. *Journal of Statistical Software*, 45(1):1–31, 2011.

[86] L. Tickle. How universities are using data to stop students dropping out. The Guardian, 2015.

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

[88] R. Vivian, K. Falkner, and N. Falkner. Computer science students' causal attributions for successful and unsuccessful outcomes in programming assignments. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*, pages 125–134, 2013.

[89] C. Watson, F. W. B. Li, and J. L. Godwin. Predicting Performance in an Introductory Programming Course by Logging and Analyzing Student Programming Behavior. In *International Conference on Advanced Learning Technologies*, pages 319–323, 2013.

[90] B. C. Wilson and S. Shrock. Contributing to success in an introductory computer science course - a study of twelve factors. In *Proceedings of the Technical Symposium on Computer Science Education*, pages 184–188, 2001.

[91] J. R. Wilson and K. A. Lorenz. Short history of the logistic regression model. In *Modeling Binary Correlated Responses using SAS, SPSS and R*, pages 17–23, 2015.

[92] A. Wolff, Z. Zdráhal, D. Herrmannova, and P. Knoth. Predicting Student Performance from Combined Data Sources. *Educational Data Mining*, 524:175–202, 2014.

[93] P. Zhang, L. Ding, and E. Mazur. Peer Instruction in introductory physics: A method to bring about positive changes in students' attitudes and beliefs. *Physical Review Physics Education Research*, 113(1):10, Jan. 2017.

[94] D. Zingaro. Peer instruction contributes to self-efficacy in CS1. In *Proceedings of the 45th technical symposium on Computer science education*, pages 373–378, 2014.

[95] D. Zingaro, M. Craig, L. Porter, B. A. Becker, Y. Cao, P. Conrad, D. Cukierman, A. Hellas, D. Loksa, and N. Thota. Achievement goals in CS1: Replication and extension. In *Proceedings of the 49th Technical Symposium on Computer Science Education*, pages 687–692, 2018.