An Instructor Dashboard for Real-Time Analytics in Interactive Programming Assignments

Nicholas Diana
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
ndiana@cmu.edu

Michael Eagle
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
meagle@cs.cmu.edu

John Stamper
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
john@stamper.org

Shuchi Grover
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
shuchi.grover@sri.com

Marie Bienkowski
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
marie.bienkowski@sri.com

Satabdi Basu
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
satabdi.basu@sri.com

ABSTRACT

Many introductory programming environments generate a large amount of log data, but making insights from these data accessible to instructors remains a challenge. This research demonstrates that student outcomes can be accurately predicted from student program states at various time points throughout the course, and integrates the resulting predictive models into an instructor dashboard. The effectiveness of the dashboard is evaluated by measuring how well the dashboard analytics correctly suggest that the instructor help students classified as most in need. Finally, we describe a method of matching low-performing students with high-performing peer tutors, and show that the inclusion of peer tutors not only increases the amount of help given, but the consistency of help availability as well.

CCS Concepts

•**Applied computing** → **Education;** *Interactive learning environments;*

Keywords

Introductory Programming; Learning Analytics; Machine Learning; Dashboards; Peer Tutors

1. INTRODUCTION

Recent advances in learning management systems and their ability to collect and display information has been shown to aid student learning. The learning analytics embedded in dashboards can provide instructors with a wealth of information about their students, however much of the research in this area has been focused on online courses and "next class" dashboards rather than traditional, offline courses and real-time dashboards [16], [17], [10]. Furthermore, even in domains rich with data, such as introductory programming, there is often little to no infrastructure in place to make insights gleaned from these data available to instructors.

Unlike most other domains, computer science education is almost always (but not always) mediated by computers. A number of development environments used in computer science education collect log files of student actions. For example, systems such as BlueJ [7], CloudCoder [14] (used in more open ended programming environments), and Alice [18], which we use in this research, generate log data. Despite this relative abundance of data, to date none of these systems integrate an instructor dashboard to take advantage of log use.

One successful use of analytics applied to student logs is providing students direct feedback in the form of hints or help messages. This forms the basis of many adaptive systems such as intelligent tutors [9] that automatically create feedback [15]. Some research has explored automatically applying these techniques in program representations [8], [12] as well.

Unfortunately, we cannot always count on the students who need help to ask for it. Student performance goals, instructor attitudes, and classroom climate can result in different patterns of help-seeking behavior [5]. For example, students concerned with social status tend to exhibit help-avoidance [13].

A potential alternative to relying on students to ask for help themselves is to train a model to predict when a student needs help and present this information to an instructor. For this to be possible, first the data need to be formatted such that at any point throughout the course the student's progress can be represented. Second, it must be possible to then make accurate predictions of student outcomes from these data. Finally, these predictions can be combined with some assumptions about their use to evaluate how well they aid in choosing students who are most in need of help. The current paper uses a dataset collected in the aforementioned Alice introductory programming environment to explore the possibility of providing real-time insights derived from raw programming log data.

LAK '17, March 13-17, 2017, Vancouver, BC, Canada
© 2017 ACM. ISBN 978-1-4503-4870-6/17/03…$15.00
DOI: http://dx.doi.org/10.1145/3027385.3027441

2. RELATED WORK

The assessment used in the current study was originally created by Werner and colleagues [18] as a way to measure computational thinking skills in middle school students. The assessment, referred to as the ”Fairy Assessment” (because the characters used are fairies), consists of three tasks designed to test comprehension, design, and complex problem solving. The authors found that while scores were not correlated with gender, age, and attendance, they did correlate with parent education, parent computer use, interest in taking a computer science class, confidence with computers, and attitude toward computers [18]. The authors also found that content knowledge of the programming environment (Alice) measured at post-survey was positively correlated with scores on the Fairy Assessment, which they argue is evidence of construct validity.

Two key features of the Werner [18] dataset are the human-graded rubric scores generated for each student and the collection of log data. The researchers graded each of the three tasks along a series of task metrics. Those task metrics are totaled to produce the Task Total, and then the Task Totals are aggregated to give the Aggregated Total. The researchers also utilized a seldom used logging feature present in Alice to capture student actions at each step. The rubric scores served as the basis for their various correlational analyses, but analysis of the log data was largely left for future work.

We revisit the Fairy Assessment dataset to explore what insights can be gained from combining the low-level log data with the human graded rubric scores, and how those data-driven insights can be made accessible to instructors in real-time. We hypothesized that, by using a supervised machine learning algorithm, we will be able to accurately predict Task Totals and Aggregated Totals. We then integrated these predictive models into a real-time instructor dashboard. We evaluated our dashboard by simulating how a teacher might use it to identify students who need help, and measuring how accurately our model identifies those students. Finally, to increase the number of students who were able to receive help, we generated a network graph of the student data to test a method of peer tutor matching.

3. METHOD AND MATERIALS

Our experiment consisted generally of three stages. First, we converted the raw log data into a series of code-states. Next, we trained a series of predictive models to predict various student grades. Finally, we integrated these predictive models into an instructor dashboard, and estimated the usefulness of the dashboard using a classroom replay.

3.1 Data

The data were collected by Werner and her colleagues [18] as part of a two year project exploring the impact of game design and programming on the development of computer science skills. The students were asked to complete an assessment task called the “Fairy Assessment,” in which students are required to fix several errors in a malfunctioning program. A key feature of this dataset is the way in which it was graded. Each student’s program was hand-graded by two experimenters along a 24 point rubric. These grades serve as the ground truth that we can use to both train and evaluate our models. We used a subset of the original data (N=227), excluding students who worked on the assessment more than 5 minutes longer than the 30 minutes allotted or with missing, ambiguous, or incorrect grade or log data.

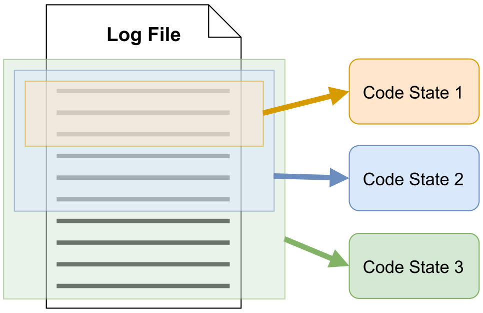


Figure 1: Visual representation of the conversion process from log files to cumulative code-states.

The raw log data generated by Alice are simply a sequential list of software actions in a text log file, and do not accurately capture the structure of the final program. To make the log data more amenable to analysis, we implemented a two-step data transformation. The first step is simply reformatting the mostly unreadable, raw list of log entries into a readable JSON format. This step was not simply for aesthetics; it allowed us to visually inspect the log data and make meaning from it, which helped us identify some important characteristics. Two key characteristics are the temporal and structural relationships between log entries. A single user action in Alice may result in multiple log file entries, and determining where one action ends and another begins is difficult for both humans and computers. Similarly, most log entries contain information about where this entry happens in Alice’s internal data structure, but the exact structural relationship is often difficult to determine due to the limited detail present in the logging system.

To empirically define these temporal and structural relationships more precisely, we created a small, locally-hosted Python server to continually monitor the log file of an active instance of Alice. Each time we performed some single action inside Alice, the server would detect a change in the log file, reformat the new data, and output the list of log entries associated with that single user action. That list could then be condensed into a single, meaningful entry. The result of this exploration is a principled method for transforming complex, sequential log data into a meaningful and succinct data structure that mirrors the internal data structure of Alice. We refer to the resulting data structure as a ”code-state.”

Representing the log data as code-states also allowed us to shift our focus from the student’s product (i.e., the final program) to the student’s process (i.e., each student action). Generating the student’s set of actions is done using the same data transformation; we simply limit the amount of data to transform. For example, to generate the student’s first code-state, we only transform the log entries that correspond to the student’s first action. Code-states are cumulative, so to generate the second code-state we transform the log entries that correspond to the student’s first and second actions, and so on. We generated a code-state for each action, for each student.

3.2 Building Predictive Models

The human-graded scores allowed us to train a supervised machine learning algorithm. First, we tokenized the final code-states of each student to generate a vocabulary of 707 tokens. We then counted the number of times each token occurs in each state, and used these token counts as features for our model. We used this vocabulary created from the final states to generate a matrix of token counts for all other codes-states. This ensured that the training data (i.e., final states) and the testing data (i.e., states prior to final states) used the same set of features.

Each reported value is the average of a 10 fold Shuffle-Split Cross-Validation. For each fold, we chose a random, classroom-sized sample of students (n=30) to use in the testing set. The remaining 197 students were assigned to the training set. We then fit a ridge regression model on the final states of every user in the training set. Because we were interested in how the model performs over time, we generated 30 time points (1 per minute) at which to test the predictive ability of the model. At each time point, we selected only the most recent code-state for each student in the testing set, and used the fitted model to predict Task Totals and Aggregated Totals for each student. We then compared these predicted scores to the known scores to produce the Root Mean Square Error (RMSE) for that time point. The python package scikit-learn was used for both cross-validation and ridge regression [11].

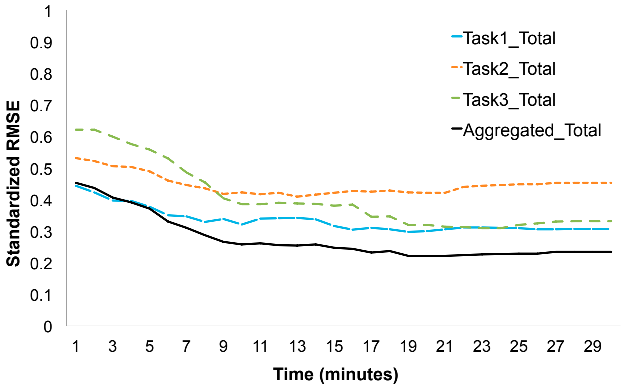


Figure 2: RMSEs of task and aggregated total predictions over time. Each model seems to stabilize at approximately 10 minutes into the course.

3.3 Instructor Dashboard

3.3.1 *Classroom Replay*

In order to evaluate the potential useful benefits of our predictive models in a typical lab-based classroom environment, we used the corpus of previously collected log data to create a classroom replay. Each student is assumed to start the assignment at the same time, and as the course progresses, their data are streamed into the live dashboard. The instructor can then monitor what the student's predicted task and aggregated totals are at any point in the course, and how they change over time.

3.3.2 *Dashboard Components*

Figure 3 highlights the important components of the dashboard. First, the Timeline (indicated by letter A. on the figure) displays how much time has past since the start of the class. Users can either drag the slider to a specific time-point or run the simulation automatically by choosing a playback speed. Figure 7 shows the dashboard progressed to 12 minutes into the class.

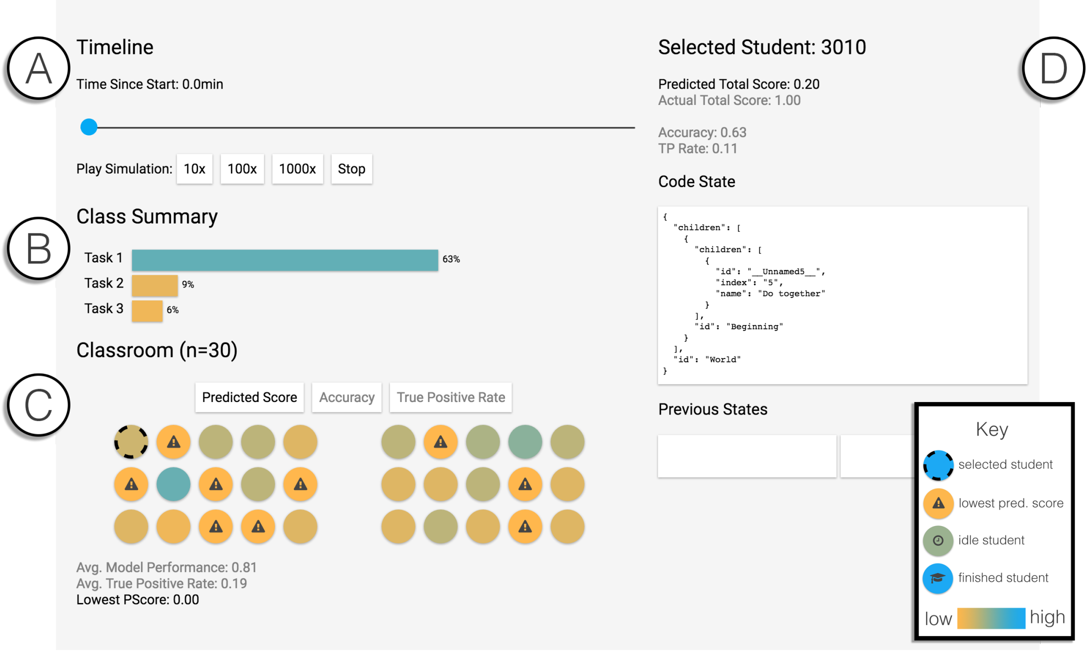
Below the Timeline is the Class Summary (letter B. on the figure). The Fairy Assessment consists of three distinct tasks. This component utilizes the predictions of task metrics to estimate the proportion of students who are currently or have already worked on each task. If the model generates a prediction of greater than 50% for a particular metric, then we guess that that student is or has worked on the task that corresponds to that metric.

Below the Class Summary is a visual representation of the classroom (indicated by letter C. on the figure). Here each circle represents a student in the class. In the screenshot shown, the color of each student corresponds to their predicted Aggregated Total, but the coloring can be changed to correspond to evaluation measures such as model accuracy and true positive rate by selecting one of the buttons listed above the students. These evaluation measures, as well as others in the dashboard, are displayed in gray text to indicate that these features are only available because the software has access to the true scores for comparison. An icon displayed within a student's circle indicates the student has been classified as belonging to one of three states. First, a caution sign icon indicates a student who has the lowest predicted score. Second, a clock icon indicates a student that has been idle for at least five minutes. Finally, a graduation cap icon indicates a student that has been idle for at least five minutes, but who also has a high predicted Aggregated Total (above a 93% or 28 out of 30 points). We classify these students as having finished the assessment. These icons can be seen in use in Figure 7.

Selecting a student will provide more detailed information about that student in the right hand panel (letter D. in the figure). This panel shows the predicted total score for the selected student and the actual total score for comparison. Also shown are student specific model evaluation measures and the selected student's current and former code-states.

3.3.3 *Evaluating the Instructor Dashboard*

We estimated the potential value of the instructor dashboard by replaying classroom data and providing some assumptions about how the instructor might use the dashboard. First, we assume that the instructor always wishes to help a student that needs help (i.e., a student who would do poorly without help). Second, we assume that the instructor helps each student for five minutes. This number is fairly arbitrary and merely dictates the number of students helped in the 30 minute class period. Third, once a student is helped, we exclude that student from the pool of possible students who could receive help. Finally, we assume that the students most in need of help are the students whose final grades (i.e., Aggregated Totals) are the lowest. Provided these assumptions, we can estimate how well our model can aid this instructor in identifying the students she wishes to help.



Timeline

Time Since Start: 0.0min

Class Summary

Task 1	63%
Task 2	9%
Task 3	6%

Classroom (n=30)

Avg. Model Performance: 0.81
Avg. True Positive Rate: 0.19
Lowest PScore: 0.00

Selected Student: 3010

Predicted Total Score: 0.20
Actual Total Score: 1.00

Accuracy: 0.63
TP Rate: 0.11

Code State

```
{
  "children": [
    {
      "children": [
        {
          "id": "__Unnamed5__",
          "index": "5",
          "name": "Do together"
        }
      ],
      "id": "Beginning"
    }
  ],
  "id": "World"
}
```

Previous States

Figure 3: Various components of the instructor dashboard. A. Timeline - Classroom replay controls, B. Class Summary - General estimates of student progress, C. Classroom - Visual representation of students, D. Selected Student - Student specific predictions and model evaluation measures as well as current and former code-states

We evaluate how well our model is selecting the correct students, the Help Index (HI), at time t as:

$$HI_t = \frac{X - |A_t - B_t|}{X} \tag{1}$$

Where X is the highest number of points possible, A_t is the lowest true score at time t, and B_t is the lowest predicted score at time t.

3.4 Peer Tutor Matching

While a measure like Help Index can aid in directing instructors to students who need assistance, it does little to address the primary resource limitation: instructor time. Even assuming we find a perfect model, if the instructor spends 5 minutes helping each student, only 6 students can possibly be helped in a 30 minute class period. Furthermore, the instructor's time does not scale with the size of the class, making this limitation especially troubling for large classes.

To increase the percentage of students who are able to receive help, we propose utilizing high performing students as peer tutors. A basic (and typical) approach to picking peer tutors consists of simply choosing a small group high performing students. Each one of these students is generally assigned to a low performing student randomly, with the two students sharing nothing except for the fact that they are both students. In a fairly open-ended environment like Alice, multiple solutions can be equally correct without sharing any similar features. Therefore, randomly matching a student with a tutor who has a different approach to the problem at best is inefficient and at worst may result in the tutor suggesting the student start over. Having access to student log data allows us to test a peer tutor matching method that is more precise than random assignment.

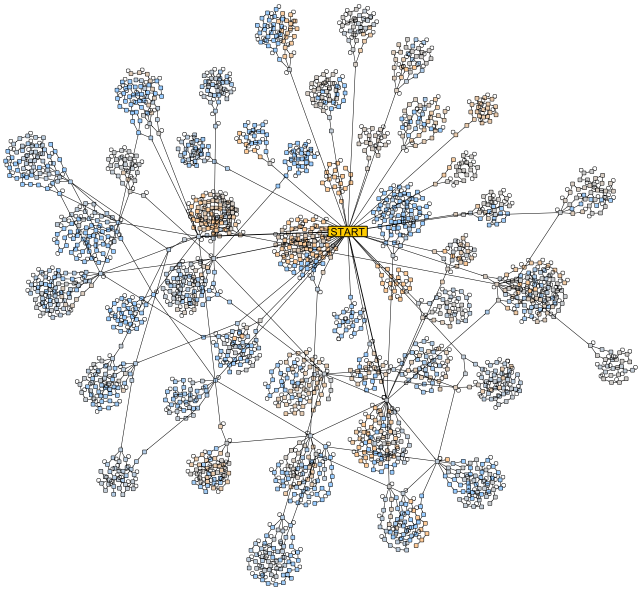


Figure 4: Interaction Network for the Fairy Assessment task.

We used a network representation of student work to measure student approach similarity. Interaction networks represent student interactions with the Alice environment as a complex network; vertices represent snapshots of the environment and edges represent the transitions that occur when students edit the Alice code. Eagle et al., expanded on the theoretical framework of interaction networks, exploring their structure and the processes that generate them [2]. Hint Factory from Stamper et al., uses an interaction network created from previous student data to train a Markov Decision Process (MDP) of student problem-solving approaches to serve as a domain model for automatic hint generation [15]. Hint factory has been applied across domains [3, 4, 6], and been shown to increase student retention in tutors [15].

The network was constructed using igraph [1], a free graphing library for network analysis. Each node of the graph represents a code state. Each edge represents a transition from one code-state to another. The network was populated by looping over each user's code states, linking them together sequentially with state transitions. If a code state identically matched another code state already represented as a node in the graph, that code state was not added, and a state transition would be drawn from the already present node to the user's next code state. A visualization of this network is shown in Figure 4.

At each time point t, we select the students in the class whose predicted final score is in the bottom 25% of all students. This represents the pool of low-performing students who we operationally define as needing assistance. From this pool, we remove students who either have already been helped or are currently being helped. Then we try to assign the remaining students tutors. This is done by selecting students from the class whose predicted final score is in the top 25% (though these thresholds are arbitrary and can be adjusted). These high-performing students make up our pool of potential tutors. For each unhelped low-performing student, we use the network graph to search for a node that is the most recent common ancestor to both the low-performing student and one of our high-performing potential tutors. These nodes not only represent a common-ground that both students have passed through, but also a potentially crucial decision-point in the task. In other words, from this shared point, one student goes on to do well, while the other goes on to do poorly. By matching low-performing students to tutors using these common ancestor nodes, we are 1) giving those students an opportunity to take a different path, and 2) reducing the probability that the tutor will simply ask the student to start over – saving not only time, but the value of the work the student has already done.

4. RESULTS

4.1 Student Performance Predictions

We were able to accurately predict the scores for all three tasks in the Fairy Assessment. Task 1 produced the best model (RMSE=0.384), followed by Task 2 (RMSE=0.500), and Task 3 (RMSE=0.556). Our model predicting the aggregated total score performed the best overall (RMSE=0.367).

To examine how the model changes over time, we generated a new model for every minute of the 30 minute course. Results from this analysis can be seen in Figure 2. As expected we see the models generally do worse at the beginning of the class period when data is scarce. However, the models seem to stabilize at around the 10 minute mark. Interestingly, we see a second pronounced dip in the Task 3 model around 16-18 minutes into the course. This may be due to several factors (e.g., diminished student activity), but may indicate the point at which most students begin working on Task 3. It is important to remember that these tasks are cumulative, so we might expect to see these temporal markers. The aggregated total model follows a similar, though less pronounced, pattern.

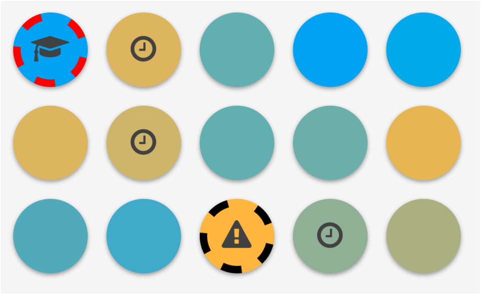


Figure 5: A selected low-performing student (black dotted outline) and a suggested peer tutor (red dotted outline).

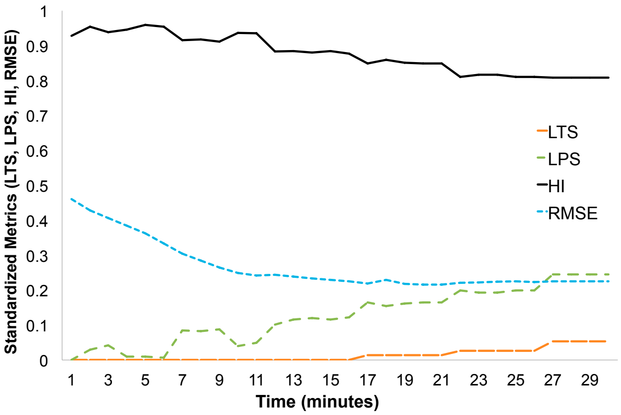
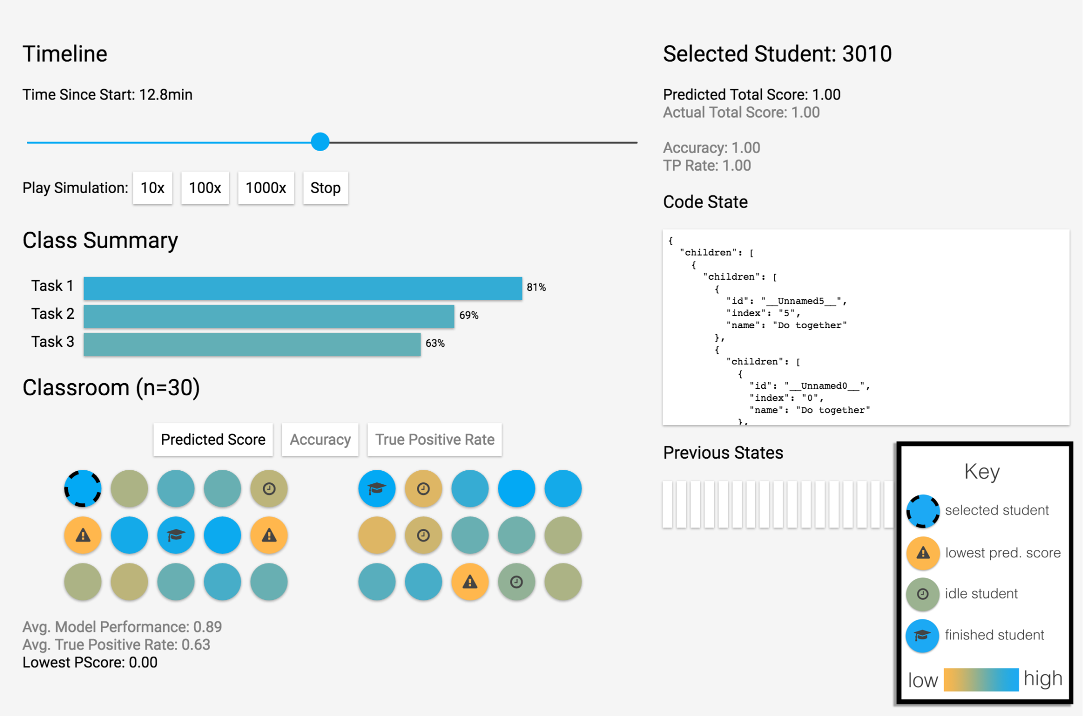


Figure 6: Help Index, RMSE, Lowest Predicted Score (LPS), and Lowest True Score (LTS) over time. Note that the Help Index (HI) does well overall, but decreases slightly over time despite a decreasing RMSE as well. This may be explained by an increasing divergence between the Lowest True Score and the Lowest Predicted Score.

4.2 Predicting Help Index

Figure 6 shows Help Index over time. On average, the model is fairly accurate at choosing the student with the lowest total score (average HI = 0.875). However, the HI also trends down over time. This may be due to a number of reasons. One possible explanation is that as low-performing students are helped (and consequentially excluded from the pool of students who can receive help), the lowest true score inches upwards. The model may be better at distinguishing no points at all (a 0%) from a small number of points, than it is at distinguishing a small number of points from a slightly higher small number of points. Another possible


Timeline

Time Since Start: 12.8min

Class Summary

Task 1	81%
Task 2	69%
Task 3	63%

Classroom (n=30)

Avg. Model Performance: 0.89
Avg. True Positive Rate: 0.63
Lowest PScore: 0.00

Selected Student: 3010

Predicted Total Score: 1.00
Actual Total Score: 1.00

Accuracy: 1.00
TP Rate: 1.00

Code State

```
{
  "children": [
    {
      "children": [
        {
          "id": "__Unnamed5__",
          "index": "5",
          "name": "Do together"
        },
        {
          "children": [
            {
              "id": "__Unnamed0__",
              "index": "0",
              "name": "Do together"
            },
```

Previous States

Figure 7: The instructor dashboard progressed to approximately halfway through the course. Note that the Class Summary now shows that the majority of students are working on the third task. The Classroom view shows less students in need of help, and more students that are idle or finished. Finally, the Selected Student pane now shows many more previous code-states than in Figure 3.

explanation is that, over time, the model has a more difficult time guessing the lowest scoring student as the code-states become more and more complex. Evidence of this can be seen in Figure 6 where the lowest predicted score (LPS in the figure) seems to trend upwards sooner than the lowest true score (LTS in the figure).

4.3 Peer Tutor Impact

In addition to evaluating how well our model can identify low-performing students, we were also interested in increasing the number of low-performing students that could be helped at any given timepoint. To this end, we implemented a peer tutor matching system that uses a network graph of all student code-states to match low-performing students with high-performing students who share a common ancestor code-state. If multiple potential tutors are found, we chose the tutor whose common ancestor is the shortest distance away from the student's current code-state. The average distance from a low-performing student's current code-state to the shared common ancestor code-state was 30.73 steps (SD=13.81).

Figure 8 shows the percentage of students classified as low-performing, high-performing, or tutors over time. The percentage of students identified as low-performing is very high at the beginning of the class period. This is most likely due to the scarcity of data at that time. As the students' code-states become more complex (and more distinguishable), we see a sharp drop in low-performing students and a steady increase in high-performing students. Interestingly, though the number of high-performing students continues to rise over the interval between 5 and 23 minutes, the number of those students who are selected to be tutors does not follow the same trajectory.

Figure 9 shows the percentage of low-performing students helped over time by the instructor, the peer tutors, and overall. We see that, while peer tutors contribute to the number of students helped, the instructor contributes more. By the end of the class period, the instructor had helped 20.81% more students than the peer tutors.

While peer tutors may not be as effective as the instructor at helping a large percentage of low-performing students, they may offer another benefit: availability. Our imposed ”5 minutes of help” assumption can be seen clearly (as expected) in Figure 9's blue, *Helped by Instructor* line, but also is evident, to a lesser extent, in the other two lines as well. However, evidence of our ”5 minute” assumption is least prominent in the *Helped by Tutors* line, suggesting that different students are becoming available as tutors as previously selected tutors are still working with their students. The impact of this improved availability of help can be most clearly seen in the steady increase of the *Total % Helped* line. Without peer tutors, we would see stretches of time where only one student is helped, leaving other low-performing students waiting. Peer tutors provide a way to supplement the more efficient, less constant instructor help with a more steady stream of availability.

5. DISCUSSION

The results of the classroom replay evaluation are promising. Our grade prediction model starts off fairly accurate and increases in accuracy until leveling off after about 10

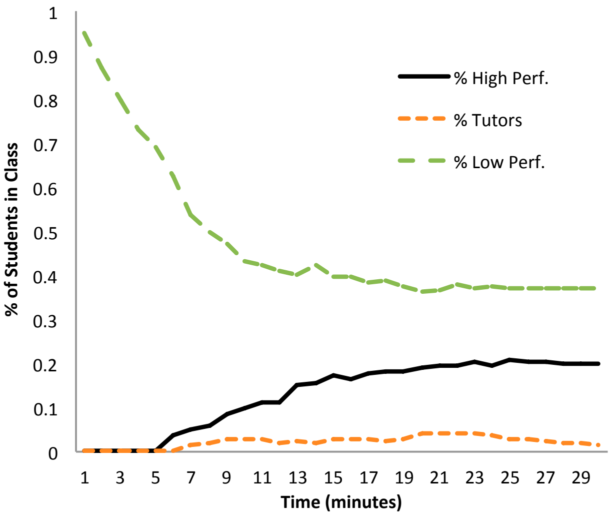


Figure 8: The percentage of the class classified as low-performing students, high-performing students, or tutors. Note: tutors are a subset of high-performing students.

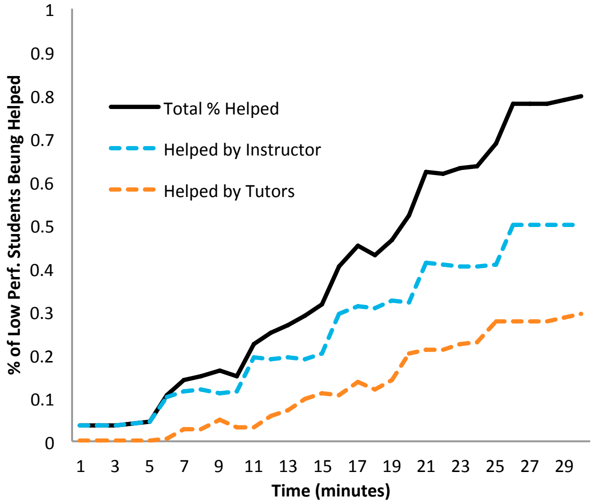


Figure 9: Percentage of low-performing students who have been helped (or are receiving help) over time. The dashed blue line represents the percentage of low-performing students helped by the instructor. The dashed orange line represents the percentage of low-performing students helped by peer tutors. The solid black line is the total percentage of low-performing students helped.

minutes. Our predictive model was also successful at accurately identifying students who are predicted to have low scores. The Help Index metric shows that the dashboard can consistently identify students who are the most in need of assistance. Finally, we were able to increase the percentage of students receiving help and the consistency with which students received help by matching low-performing students with high-performing peer tutors who have similar program states.

The "intervention strategy" we used in this evaluation, while simple, succeeds in demonstrating that we can identify students who are most in danger of failing the assignment, and that we can identify these students relatively early. In a real classroom, instructors using the dashboard will likely have interruptions from help-seeking students, and other real-world events that could result in selecting a different student for one-on-one intervention. Additionally, an expert instructor may not need the grade prediction portion of the dashboard, however it might still prove useful for any teaching assistants available. In addition to the grade predictions, the dashboard also provides a high level view of the current progress of the entire classroom including which tasks students are currently working on and how many students are sitting idle. These insights would not be possible otherwise.

The current Alice environment does not support this type of real-time logging, however the work we have presented here provides a good preliminary look into the potential benefits of implementing such as system. It is important to explore interventions, such as this dashboard, thoroughly before placing them into a classroom environment, and the classroom replay presented here is one way of doing that. The results of our study provide evidence that the implementation of real-time logging could have an impact in a real classroom.

6. CONCLUSIONS

In this paper, we demonstrate that task and aggregated totals from an introductory programming assessment can be predicted by training a supervised machine learning algorithm on human-graded rubric scores. These predictions were integrated into an instructor dashboard. Finally, the ability of this dashboard to successfully identify the students who might most benefit from help was evaluated by simulating an instructor's interaction with the dashboard. These results suggest that, given an appropriate representation of the student's program state coupled with a rich set of training data, a machine learning model can accurately predict student scores. These predictions have a multitude of applications. This paper explored identifying low-scoring students, but these predictions may also be useful in evaluating peer-grading or identifying students who have completed the assessment early.

7. FUTURE WORK

One potential way to increase the number of students helped is by clustering similar low-performing students together. Future work will focus on identifying clusters of students who may benefit from the same intervention.

Another potential way to increase the number of students helped is to provide intelligent non-human help. We hope to utilize the accuracy of our predictive models to implement automatically generated feedback for the students.

8. ACKNOWLEDGMENTS

This research was supported by the National Science Foundation (NSF grant award number 1522990).

9. REFERENCES

[1] G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*, 1695(5):1–9, 2006.

[2] M. Eagle, D. Hicks, B. Peddycord, III, and T. Barnes. Exploring networks of problem-solving interactions. In *Proceedings of the Fifth International Conference on Learning Analytics And Knowledge*, LAK '15, 21–30, New York, NY, USA, 2015. ACM.

[3] M. Eagle, M. W. Johnson, T. Barnes, and A. K. Boyce. Exploring player behavior with visual analytics. In *FDG*, 2013.

[4] D. Fossati, B. Di Eugenio, S. Ohlsson, C. W. Brown, L. Chen, and D. G. Cosejo. I learn from you, you learn from me: How to make ilist learn from students. In *AIED*, 491–498, 2009.

[5] J. M. Furner and A. Gonzalez-DeHass. How do students' mastery and performance goals relate to math anxiety. *Eurasia Journal of Mathematics, Science & Technology Education*, 7(4):227–242, 2011.

[6] A. Hicks, B. Peddycord III, and T. Barnes. Building games to learn from their players: Generating hints in a serious game. In *Intelligent Tutoring Systems*, 312–317. Springer, 2014.

[7] M. C. Jadud. A first look at novice compilation behaviour using bluej. *Computer Science Education*, 15(1):25–40, 2005.

[8] W. Jin, T. Barnes, J. Stamper, M. J. Eagle, M. W. Johnson, and L. Lehmann. Program representation for automatic hint generation for a data-driven novice programming tutor. In *Intelligent Tutoring Systems*, 304–309. Springer, 2012.

[9] K. R. Koedinger, J. R. Anderson, W. H. Hadley, M. A. Mark, et al. Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education (IJAIED)*, 8:30–43, 1997.

[10] M. Lovett, O. Meyer, and C. Thille. Jime-the open learning initiative: Measuring the effectiveness of the oli statistics course in accelerating student learning. *Journal of Interactive Media in Education*, 2008(1):1–18, 2008.

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

[12] K. Rivers and K. R. Koedinger. Automatic generation of programming feedback: A data-driven approach. In *The First Workshop on AI-supported Education for Computer Science (AIEDCS 2013)*, pages 50–59, 2013.

[13] A. M. Ryan, L. Hicks, and C. Midgley. Social goals, academic goals, and avoiding seeking help in the classroom. *The Journal of Early Adolescence*, 17(2):152–171, 1997.

[14] J. Spacco, D. Fossati, J. Stamper, and K. Rivers. Towards improving programming habits to create better computer science course outcomes. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, 243–248. ACM, 2013.

[15] J. Stamper, M. Eagle, T. Barnes, and M. Croy. Experimental evaluation of automatic hint generation for a logic tutor. *International Journal of Artificial Intelligence in Education (IJAIED)*, 22(1):3–18, 2013.

[16] K. Verbert, E. Duval, J. Klerkx, S. Govaerts, and J. L. Santos. Learning analytics dashboard applications. *American Behavioral Scientist*, 57(10):1500–1509, 2013.

[17] K. Verbert, S. Govaerts, E. Duval, J. L. Santos, F. Van Assche, G. Parra, and J. Klerkx. Learning dashboards: an overview and future research opportunities. *Personal and Ubiquitous Computing*, 18(6):1499–1514, 2014.

[18] L. Werner, J. Denner, and S. Campe. The Fairy Performance Assessment : Measuring Computational Thinking in Middle School. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education - SIGCSE '12*, 215–220, 2012.