Early Identification of Student Struggles at the Topic Level Using Context-Agnostic Features

Kai Arakawa
Western Washington University
hicksk5@wwu.edu

Qiang Hao
Western Washington University
qiangh@wwu.edu

Wesley Deneke
Western Washington University
denekew@wwu.edu

Indie Cowan
Western Washington University
cowani@wwu.edu

Steven Wolfman
The University of British Columbia
wolf@cs.ubc.ca

Abigayle Peterson
Western Washington University
peter394@wwu.edu

ABSTRACT

The identification of student struggles has drawn increasing interests from computing education and learning analytics communities in recent years, considering the high failure rate and fast enrollment growth of computer science courses. Prior studies on this topic employed a multitude of data sources and methodologies with varying degrees of success. Nearly all studies attempted to predict low overall course performance to identify struggling students, risking oversimplifying student learning and struggles. Additionally, many studies utilize data sources that are limited to their original contexts or local student demographics, making it difficult to replicate or put the findings into practice. To address these gaps, we studied the feasibility of identifying student struggles at the topic level using features that are agnostic to courses and contexts. Our results show that it is possible to identify student struggles at a more fine-grained level within days. Our findings contribute new insights into automatic identification of student struggles at the topic level on a large scale, which can be used to guide meaningful interventions on student learning.

CCS CONCEPTS

• **Social and professional topics → Computer science education**; **Student assessment**; *Adult education*; • **Applied computing → Education**;

KEYWORDS

prediction of student success, struggle identification, agnostic features, learning analytics

ACM Reference Format:
Kai Arakawa, Qiang Hao, Wesley Deneke, Indie Cowan, Steven Wolfman, and Abigayle Peterson. 2022. Early Identification of Student Struggles at the Topic Level Using Context-Agnostic Features. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2022), March 3–5, 2022, Providence, RI, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3478431.3499298

1 INTRODUCTION

Early identification of student struggles has drawn increasing attentions from research communities of both computing education and learning analytics. This is driven by the fast enrollment growth in computer science courses and their consistently high failure and dropout rate. A continued challenge in computing education is that the students who struggle the most are the least likely to seek help [1, 2]. If these at-risk students can be identified early, it can enable timely interventions, even when facing large student to teacher ratios [3, 4].

The computing education and learning analytics research communities have been studying various approaches to predict struggling students. Early studies focused on investigating important factors in predicting student performance, including prior course performance, programming experience, demographics, and student readiness [5–7]. These studies advanced the understanding of the relationship between such factors and student performance in programming courses, but yielded mixed results in practice [8, 9]. More recent works have focused on collecting student data generated throughout the course period and building models to predict student overall performance. The nature and source of such data vary greatly. Examples include student programming behavior, assignment performance, clicker data, and question-and-answer interactions on online forums [10–12]. These recent studies improved our understanding of the characteristics of low- or high-performing students, as well as the extent to which machine learning can predict struggling students.

Despite the contributions of prior studies, the identification of student struggles has been largely limited to the prediction of student overall course performance. Prior studies generally equated student struggles with low overall performance (e.g., < 50% overall score), and used it as the prediction or classification target. Prediction or classification models based on this definition provide little actionable insights on where and when students may struggle, which risks oversimplifying student learning. Additionally, many prior studies utilized features specific to the study contexts or student demographics and relied on complicated data collection procedures, which limit the generalizability of their findings and create challenges for successful future replications.

This study aims to address these issues by investigating early identification of student struggles at the topic level in programming courses. Being able to identify student struggles at the topic level has the potential to provide actionable insights into when and where students are encountering difficulties. This can allow instructors to provide higher quality and more targeted interventions. To increase the replicability of our approach, our study utilized software popular in programming courses such as version-control and automated testing tools, and built an automated pipeline to collect student performance and behavior data from such software. To strengthen the generalizability of our findings, we built our models using features that are agnostic to courses and contexts, based on data collected from different courses taught by different instructors. Our findings have shown that it is possible to identify student struggles at a fine-grained level. Our results contribute new insights into automatic identification of student struggles at the topic level on a large scale, which can be used to guide meaningful interventions and enhance student learning.

2 BACKGROUND

2.1 Early Studies

Early efforts to identify predictors of student learning struggles primarily studied static factors. Static factors refer to student attributes that are less susceptible to changes within the context of a study. For example, a student's college grade point average (GPA) can be considered a static factor because it is unlikely to be changed during a semester when students take a programming course.

A common approach is to collect biographical data in an attempt to predict student course performance. For example, Newsted used biographical information collected from a survey of 472 students across three semesters in an intro to FORTRAN programming course [13]. Their aim was to provide better counseling to prospective students and advise on teaching methodology. The models that were created in the study were able to account for 41% of the variation in the students' grades. Other studies employed biographical information to develop statistical models to aid in admissions decisions to mixed success [14, 15].

Kurtz utilized an aptitude test that placed students into one of three development levels to predict success in learning programming [16]. While the sample size was fairly limited ($n = 23$), the assigned development levels explained 66% of the variation in the letter grades earned. On account of Kurtz's encouraging results, a later study by Barker and Unger used an abridged version of Kurtz's assessment to segregate students into high and low performers [17]. The authors claimed that a predictor constructed from their abridged assessment and other static factors such as college GPA could be suitable for placement and advising. Werth explored this idea using Kurtz's intellectual development test along with biographical information, and two other tests [18]. Werth found that Kurtz's test, a cognitive style test, and two pieces of biographical information correlated with letter grade.

Overall, the exclusive use of static factors is generally insufficient to identify struggling students in a classroom setting. First, the data is difficult to collect, often requiring surveys and/or lengthy examinations. Furthermore, the models developed for one institution fail to generalize to others. This would require each university, at a minimum, to perform a series of studies to evaluate the efficacy of another institution's model using their own data – an expensive and time-consuming process. However, the use of static factors in admissions and placement is, while suspect, understandable given the general lack of alternative data sources.

2.2 Recent Studies

Studies in recent years have began utilizing dynamic factors to make predictions of student performance [8]. Dynamic factors are ones which change throughout a term; thus their requisite data is collected multiple times. Dynamic factors of particular interest for the prediction of student performance are ones that respond rapidly to a student's evolving success in the course.

Clicker quizzes are short quizzes administered during a lecture to gauge the students' comfort with the subject and contribute to an active learning environment [19, 20]. Porter et al. was able to use clicker quizzes to predict students who later struggled on the final exam early in the course [12]. Furthermore, they found that scores from the first three weeks out of twelve were more predictive than scores from any other set of weeks. Liao et al. expanded upon this work by creating a machine learned model using the clicker quiz data from one term and testing the model on data collected from a subsequent term [21]. The model that they developed was capable of producing high quality predictions of final exam scores after only the first three weeks of twelve. The researchers later explored other factors and found that, when available, grades in prerequisite courses outperformed clicker quiz scores as predictors of student success [3].

Other studies instead monitor how students interact with course management systems. Zafra et al. collected student interactions with discussion boards, quizzes, and assignments as well as the amount of time each student spent in the forum [22]. The data was used to train many classifiers, and the outputs of the individual classifiers were weighted to construct a more reliable multi-instance classifier. Fire et al. went another direction and monitored which students were working together, forming a graph [23]. The correlation between key graph theoretical metrics and final exam score was explored. Unfortunately, despite the novel approach, all models failed to explain more than 17.5% of the variation. Other studies track interactions with online learning materials offered as a part of the course. Leppänen et al. monitored how long each element, be it a paragraph or an image, of the online textbook was visible on each student's screen [11]. Similarly, Yang et al. collected clickstream data from students watching lecture videos in a massive online open course (MOOC) and used time series machine learning techniques to develop a predictor to much success [24].

Perhaps the most fine-grain data source of interest is keystroke data or compilation events collected by the IDE or a plugin running in the IDE while students are programming. In 2006, Jadud proposed a metric called the error quotient (EQ) which is a handcrafted metric dependent on the type, frequency, and location of errors [25]. While Jadud observed that EQ correlated with academic performance, Petersen et al. found that the EQ parameters are highly context dependent and furthermore, there are some contexts in which no parameters result in an effective predictor [26]. Despite this, variants of EQ have shown better success [27, 28]. Other studies using IDE data have turned to machine learning [29, 30]. These studies appear to produce more reliable results compared to the handcrafted methods. Unfortunately, the course setup required to collect such data is inherently constrained. Either plugins would have to be written for a variety of IDEs, or students would be restricted in their choice of IDE/text editor.

Other efforts to classify and predict struggling students involve the use of version control (VC) systems. Such technologies are invaluable in industry, and used extensively in the classroom [31, 32]. In 2005, Mierle et al. utilized the now antiquated VC software CVS to identify features which may act as predictors of course performance [33]. Only three of the 166 features that were explored demonstrated a significant correlation with course performance. Later, in 2018, Guerrero-Higueras et al. further explored this idea, this time using Git [34, 35]. In contrast to all previously mentioned studies, this study was focused on predicting the students' performance on a particular assignment, not the course as a whole. While the authors' models appear to have performed well, it was not stated if the predictions were performed using data that was available before the assignment was due. If the models were in fact trained and evaluated with data collected right up until the assignment was due, the real world use cases are severely limited. The window in which such predictions would be both possible and of any use is restricted to after the assignment is due, but before the assignment is graded. Sprint and Conci also explored the relationship between students' interactions with Git and their performance in both assignments and the course [36]. They found two features that showed a significant correlation with either the course grade or the performance on a programming assignment. However, the authors found that the identified correlations struggled to generalize across all courses studied. Still, their findings suggest that more sophisticated nonlinear methods could be used to better predict student performance.

Similar to [34, 35] and [36], Teusner et al. predicted which students were struggling on specific assignments [37]. However, the prediction of the struggling students was not Teusner et al.'s end goal. Instead, they were investigating the effects of automated interventions and bonus exercises delivered to struggling students in a massive open online course (MOOC). This focus, necessitated the in situ prediction of struggling students. A student was classified as struggling if they spent more than a set amount of time on an exercise. The simplicity of this prediction strategy is problematic, as it fails to account for the multitude of reasons a student may take extra time to complete an exercise. The authors recognize this, stating that their predictor failed to discern the truly struggling students from the ones that were simply working slowly. Prediction efficacy aside, the time a student spends on an exercise is also difficult to collect. It not only assumes that an assignment is broken up into discrete exercises, but it also requires the knowledge of exactly when, and on which exercise, a student is working. While this is not a problem in the study's original context, a MOOC, it does prevent their methodology from being applied to traditional course modalities.

2.3 Gaps

Prior studies on this topic, with few exceptions, sought to identify student struggles through binary classification on if a student would fail a course or final exam in the end. This focus fails to account for the dynamic nature of student learning progress and student struggles, which risks oversimplifying student learning. For example, every student that ultimately passes the course will be labeled as not struggling, despite the fact that many of those students will still struggle on various topics throughout a term.

Courses	Students	Assignments	Student submissions
CS2 (face-to-face)	137	4	542
CS2 (online)	117	3	338
Data structures (face-to-face)	58	3	169

Table 1: Basic information of participating students

Furthermore, many prior studies relied on features which are unique to their classrooms, or features that are difficult to track or collect data on in many computer science courses. For example, studies which utilize clicker quizzes rely on remote control devices, which many classrooms may not be equipped with [12, 20, 21, 38]. As another example, studies which utilize clickstream or compiler event data require trackers embedded in IDEs or text editors, limiting students in the editors they can choose from [25, 27, 29, 30].

To address these gaps, we investigated the identification of student struggles at the topic level in programming courses using context-agnostic features. We utilized version control and automated testing as data sources – sources which we identified as the two most popular tools in computing education – to avoid the reliance on context-dependent features [39, 40]. Our findings contribute to the understanding of automatic identification of student struggles in programming courses on a large-scale.

3 RESEARCH DESIGN

This study is guided by one research question:

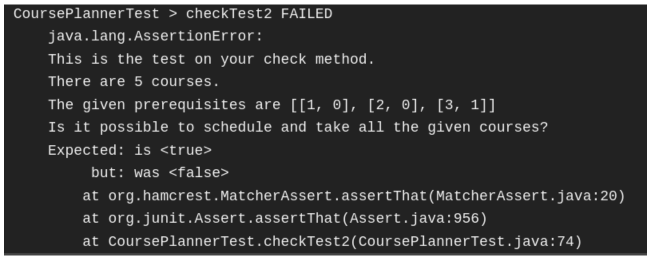
To what extent can we identify student struggles in programming courses at the topic level using context-agnostic features?

3.1 Experiment Design

We conducted this study in the setting of a large university in the northwestern United States. To strengthen the generalizability of our findings and mitigate potential noise and variance, we involved 312 students taking three different programming courses offered by different instructors and in different modalities (see Table 1).

To avoid oversimplification in identifying struggling students, we collected student data at the topic level instead of the course level. This is achieved by collecting student behavior and performance data on all assignments per course. It is typical for a programming course to assign multiple programming assignments, each addressing a particular and different topic that is covered by the course. The courses that we collected data from are no exception. For example, the Data Structures course we collected data from assigned three individual programming assignments that covered one topic per assignment, including *sorting*, *searching*, and *graphs*. As a result, student data collected for each programming assignment could be leveraged to identify struggling students at the topic level.

Additionally, we focused on student data that can be easily collected in modern computer science learning and teaching activities, in order to increase the practicality of our findings and facilitate future replications. After examining over 500 empirical studies



```
CoursePlannerTest > checkTest2 FAILED
    java.lang.AssertionError:
    This is the test on your check method.
    There are 5 courses.
    The given prerequisites are [[1, 0], [2, 0], [3, 1]]
    Is it possible to schedule and take all the given courses?
    Expected: is <true>
         but: was <false>
        at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:20)
        at org.junit.Assert.assertThat(Assert.java:956)
        at CoursePlannerTest.checkTest2(CoursePlannerTest.java:74)
```

Figure 1: A sample of automated testing on Travis-CI.

published between 2010 and 2020 in the main publication venues of computing education research[1] in the past five years, we identified version-control systems and automated testing as the two most popular tools adopted in modern computer science classrooms measured by keyword frequency. Version-control systems and automated testing are widely supported by a range of efficient tools that can be easily adopted by computer science instructors. More importantly, version-control systems and automated testing, in tandem, can provide fine-grained data on student behavior and performance, such as when students worked on an assignment and whether students successfully achieved a series of milestones over a period of time. To ensure that the collected data is agnostic to the course context or students who took the course, we set up the experiments with no extraneous assumptions about the students or course context. We used no survey data (which may be limited by context or demographics) or specialized devices which are in any way unique to the classroom.

Our study adopted Git and GitHub as the version-control system, given their wide popularity in computer science courses and the ease at which data can be collected through GitHub APIs [39]. We utilized Travis-CI, a continuous integration service, to support automated testing and feedback considering its easy configuration, research-based efficacy in providing automated testing and feedback to students, and its capacity for real-time data collection [41]. Additionally, Travis-CI also supports data collection through its APIs. To ensure consistency across all classes, we worked with instructors to develop multiple unit tests for each assignment that provide sufficient test coverage. Each test case included feedback that reported the expected and actual outputs from the student's submission when a unit test failed (see Figure 1).

3.2 Data collection and processing

Data was collected from 1049 student submissions on 10 programming assignments from three different courses. For each assignment, a student maintained a separate GitHub repository. While working on an assignment, students periodically committed and pushed their progress to GitHub. Automated testing was performed on each push, and feedback was sent to students through emails, and could also be viewed through the Travis-CI web interface. We collected data on each push and computed the following features accordingly:

[1]ACM ICER, ACM SIGCSE, ACM ITiCSE, ACM Transactions on Computing Education, and Computer Science Education Journal

- *Normalized timestamp*: The timestamp of the push expressed as a ratio where 0 is when the assignment was posted and 1 is when the assignment is due.
- *Normalized time of day*: The time of day expressed as a ratio where 0 is 00:00:00 and 1 is approximately 23:59:59.
- *Additions and deletions*: The number of lines added and removed.
- *Test ratio*: The ratio between the number of passed test cases and the number of total test cases.
- *Error ratio*: The ratio between the number of previous pushes that failed to compile and the number of previous pushes.
- *First commit timestamp*: The normalized timestamp of the first commit.
- *Number of pushes with no progress*: The number of pushes the student has made with their test ratio remaining constant or decreasing.
- *Highest test ratio*: The maximum test ratio that had been achieved thus far by the student.

Whether a student struggles on an assignment (a topic) is the target of our analysis. For the ground truth labels, the natural definition of "struggling on an assignment" is if the student achieves less than a certain percentage on said assignment. However, this definition fails to account for the different standards that students set for themselves. As such, we defined a student as struggling if, by the time the assignment is due, they failed at least one test case – thus, only the students whose code passed every test case were labeled as not struggling. This is supported by Arakawa et al. which found a substantial overlap between our definition of struggling and students that were confirmed to be struggling via testimony [42]. On average, 20.9% of all students were labeled as struggling across all assignments. In contrast, less than 8% of students failed a course across the three courses.

4 RESULTS

4.1 Machine Learning Methodology

Due to our data being sequential with differing sequence lengths, most machine learning techniques are unsuitable. Recurrent neural networks are the natural choice since our data is well-ordered. Recurrent neural networks are capable of natively processing sequential data of an arbitrary length. In a typical neural network, each neuron takes the outputs of the previous layer as input and transforms them into a single output. A recurrent neuron, often called a node, does the same, but it also maintains a hidden state called the recurrent or the memory. Most recurrent neuron constructions differ in how they update their memory. The two types of recurrent neural networks that we used in this study are the vanilla recurrent neural network (RNN) and long short term memory network (LSTM). An LSTM node has more fine-grained control over its memory than an RNN and, as a result, has more trainable parameters than an RNN node.

In our preliminary investigations, we experimented with a single node RNN and LSTM, a wide RNN and LSTM each with one layer of three nodes, and a deep RNN and LSTM with multiple layers of multiple nodes. While the single node and wide architectures performed well, the deep architectures tended to overfit, even after applying regularization. The few combinations of hyperparameters

Model type	AUROC	Variance	# of parameters
RNN	0.821	0.011	13
LSTM	0.910	0.0021	46
Wide RNN	0.910	0.0034	43
Wide LSTM	**0.922**	0.0019	160
Baseline model	0.744	N/A	

Table 2: Cumulative performance of the various models

that did not result in gross overfitting did not demonstrate appreciable results when compared to the simpler models. Thus, we did not explore the deep architectures further.

To serve as a point of comparison, we constructed a baseline model that does not require pretraining. Given a sequence of commits belonging to a student, the baseline model finds the line of best fit to approximate the test ratio as a function of the normalized timestamp. This approximated function is used to predict at what time the student will achieve a test ratio of 1 corresponding with all tests passing. Thus, the student can be classified as struggling if they are predicted to finish after a predetermined threshold. This threshold can be adjusted as necessary to affect the sensitivity and specificity of the baseline model.

4.2 Model Evaluation

To better evaluate our methodology and mitigate potential effects of heterogeneity in our dataset, we utilized 10-fold stratified cross-validation. Additionally, because our dataset contains a class imbalance, with only 20.9% of students being labeled as struggling, we used the area under the receiver operating characteristic (AUROC) as the metric by which we evaluated models [43].

We evaluated each model's cumulative performance by making a series of predictions for every commit sequence in the validation dataset; one prediction for every commit. These predictions along with the ground-truth labels were used to calculate the ROC. This evaluates the performance of the model at making both early and late predictions. The cumulative performance of each model is reported in Table 2. Wide LSTM, as our best-performing model, achieved a cumulative AUROC of 0.922 (see Figure 2).

The cumulative performance of a model is useful, but it does not reveal how well a model performs in terms of making accurate early predictions. To achieve this, we presented the model performance comparison in terms of how early a model can accurately identify student struggles in Table 3. The best-performing model, wide LSTM, required the data of 2.43 days to achieve a mean AUROC $\geq$ 0.7 on an assignment that lasted for 14 days. In contrast, the RNN, required the data of 7.72 days while the baseline model only requires the data of 6.3 days to achieve the same efficiency.

To gauge the real-time efficiency of our best-performing model, we filtered the validation dataset to use data prior to various cutoff times (i.e. data within the 50% of the assignment duration). Based on that, we calculated the real-time performance of wide LSTM using the filtered dataset. This result is reported in Figure 3. For a more fine-grained analysis of the performance of the models, we plotted all cutoff time-performance pairs, forming a real time

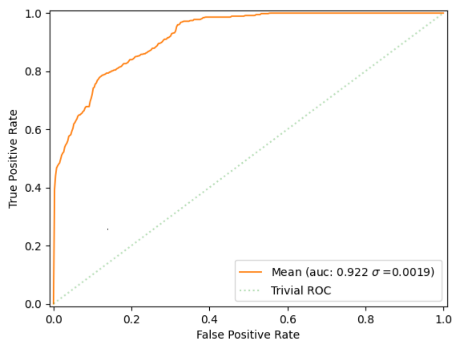


Figure 2: Receiver operating characteristic of the wide LSTM

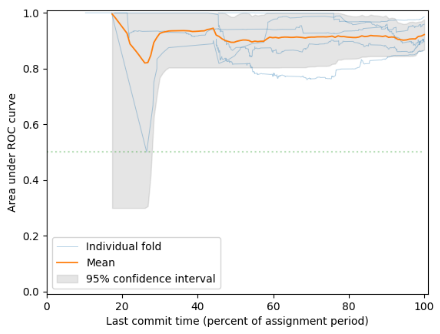


Figure 3: Real time performance curve of the wide LSTM

performance curve. It is worth noting that wide LSTM achieved stable performance after 25% into the assignment duration time.

5 DISCUSSION

All machine learning models substantially outperformed the baseline model in terms of cumulative performance. While the baseline model achieved a cumulative AUROC of 0.744, the best machine learning model, wide LSTM, achieved a cumulative AUROC of 0.922, and the worst, RNN, achieved a cumulative AUROC of 0.821. Despite achieving superior cumulative performance, not every machine learning model performed equally well in terms of how early an accurate struggle identification can be achieved. The best-performing model was the wide LSTM, which only required the data of 2.43 days to achieve a mean AUROC $\geq$ 0.7 on an assignment that lasted for 14 days. In contrast, the RNN, required the data of 7.72 days while the baseline model only requires the data of 6.3 days. These findings support the possibility of building an automated machine learning system that identifies student struggles at the topic level and provides actionable insights for instructors to intervene student learning.

Our study was conducted on a more fine-grained level compared with prior studies — we investigated the early identification of struggles on the topic level, whereas most prior studies focused on predicting student overall performance [8]. Although the results

Model type	Mean AUC ≥ 0.7	Mean AUC ≥ 0.8	AUC lower bound ≥ 0.7	AUC lower bound ≥ 0.8
RNN	7.72 days	14.0 days	–	–
LSTM	3.97 days	6.28 days	8.16 days	11.02 days
Wide RNN	3.86 days	4.30 days	8.16 days	13.45 days
Wide LSTM	**2.43 days**	**2.43 days**	**4.08 days**	**4.52 days**
Baseline model	6.3 days	–	N/A	N/A

Table 3: The earliest a model can make a prediction in a 14-day assignment with various AUC constraints

are not directly comparable, our models performed on par with, or better than, many models constructed to predict overall student success or failures if the same metrics are being used. Machine learning models constructed to predict student overall performance generally have an accuracy rate between 70% and 90%. For instance, one study conducted by Castro-Wunsch et al. achieved an accuracy of 85.29% (after bias correction) when predicting which students would fail a course [30]. When the models are restricted to data from the first 40% of the term, the accuracy drops to 72.90%. While their results are not directly comparable to ours, the wide LSTM achieved an AUROC of 0.7 (analogous to 70% accuracy) within the first 30% of an assignment duration.

More importantly, training efficient models that identify struggling students may not require features specific to courses, context, or student demographics. Many prior studies relied on features that are specific to their contexts or student bodies, utilized surveys and quizzes to understand student prior programming experience, or retrieved student information from the institutions [8, 12, 25, 26, 44]. These methods may contribute to training a very performant machine learning model, but also pose serious challenges to replications as well as automating the identification of struggling students, considering the amount of manual labor required for these methods. In contrast, our findings revealed that efficient models can be developed and trained without such data. The data that was collected in this study came from version-control (GitHub) and automated testing (Travis-CI) software that are widely popular in computer science courses [39, 40]. Both data sources support automatic data collection through their APIs. The design as such can lower the barriers and costs associated with future replication studies on this topic, and highlights the viability of building a fully automated system that identifies student struggles.

Additionally, our findings demonstrated that it is possible to build efficient machine learning models that identify student struggles at the topic level based on the data collected in just a few days, regardless of the topic, course, or students. In addition to the implication of fully automated systems that specialize in early struggle identification, this finding also has implications for educators in terms of how to help students proactively in a large-scale computer science course. The conventional approach to help students in need is to hire capable teaching assistants and set up as many office hours as possible. However, there is abundant research showing that students who need help the most are reluctant to seek help [1, 2]. If instructors can be equipped with a system that automatically identifies struggling students, they might be able to mobilize teaching assistants to intervene on student learning struggles with more guided information on when and where the struggles might happen. This will have substantial positive impact on failure and dropout rate of programming courses [37].

6 LIMITATIONS

The most substantial limitation of our study is the size and composition of our dataset. The size of our data is relatively small, given just over one thousand submissions and five thousand commits. All data was collected from a single institution. The extent to which our results can be effectively replicated in other contexts requires further investigation. A significantly larger dataset from multiple institutions can potentially enable more experiments, resulting in models with better performance. If possible, future studies should consider collecting data at the topic level from different higher education institutions.

Additionally, the replication of the data pipeline of our approach requires the use of version-control tools. Despite the wide popularity of version-control tools (e.g., Git) in many programming courses, not every programming course has adopted such tools. When the learning of programming has to happen at the same time as the learning of how to version-control code in the same course, it may lead to cognitive overload for some students. If a student misunderstands version-control, they might commit and push the code too frequently or not at all. In such a scenario, the efficiency of the model to identify student struggles could be negatively affected. Future studies may study the extent to which the noise in data can impact the data collection and model construction.

7 CONCLUSIONS

This study investigated the feasibility of identifying student struggles at the topic level using context-agnostic features that can be collected automatically in large-scale programming courses. Our results have shown that it is possible to identify struggling students at a more fine-grained level within a short period of time. Our results contribute new insights into automatic identification of student struggles at the topic level on a large scale, which can be used to guide meaningful interventions on student learning, as well as building software that connects to popular tools in programming courses and automating student struggle identification in computer science courses.

REFERENCES

[1] David H Smith IV, Qiang Hao, Vanessa Dennen, Michail Tsikerdekis, Bradly Barnes, Lilu Martin, and Nathan Tresham. Towards understanding online question & answer interactions and their effects on student performance in large-scale stem classes. *International Journal of Educational Technology in Higher Education*, 17(1):1–15, 2020.

[2] Qiang Hao, Brad Barnes, Robert Maribe Branch, and Ewan Wright. Predicting computer science students' online help-seeking tendencies. *Knowledge Management & E-Learning: An International Journal*, 9(1):19–32, 2017.

[3] Soohyun Nam Liao, Daniel Zingaro, Christine Alvarado, William G Griswold, and Leo Porter. Exploring the value of different data sources for predicting student performance in multiple cs courses. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 112–118, 2019.

[4] Soohyun Nam Liao, Sander Valstar, Kevin Thai, Christine Alvarado, Daniel Zingaro, William G Griswold, and Leo Porter. Behaviors of higher and lower performing students in cs1. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, pages 196–202, 2019.

[5] Sally Fincher, Anthony Robins, Bob Baker, Ilona Box, Quintin Cutts, Michael de Raadt, Patricia Haden, John Hamer, Margaret Hamilton, Raymond Lister, et al. Predictors of success in a first programming course. In *Proceedings of the 8th Australasian Computing Education Conference (ACE 2006)*, volume 52, pages 189–196. Australian Computer Society Inc., 2006.

[6] Jens Bennedsen and Michael E Caspersen. An investigation of potential success factors for an introductory model-driven programming course. In *Proceedings of the first international workshop on Computing education research*, pages 155–163, 2005.

[7] Brenda Cantwell Wilson and Sharon Shrock. Contributing to success in an introductory computer science course: a study of twelve factors. *Acm sigcse bulletin*, 33(1):184–188, 2001.

[8] Christopher Watson, Frederick WB Li, and Jamie L Godwin. No tests required: comparing traditional and dynamic predictors of programming success. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 469–474, 2014.

[9] Holger Danielsiek and Jan Vahrenhold. Stay on these roads: Potential factors indicating students' performance in a cs2 course. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 12–17, 2016.

[10] Petri Ihantola, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen H. Edwards, Essi Isohanni, Ari Korhonen, Andrew Petersen, Kelly Rivers, Miguel Ángel Rubio, Judy Sheard, Bronius Skupas, Jaime Spacco, Claudia Szabo, and Daniel Toll. Educational data mining and learning analytics in programming: Literature review and case studies. In *Proceedings of the 2015 ITiCSE on Working Group Reports*, ITICSE-WGR '15, page 41–63. Association for Computing Machinery, New York, NY, USA, 2015. ISBN 9781450341462. doi: 10.1145/2858796.2858798. URL https://doi.org/10.1145/2858796.2858798.

[11] Leo Leppänen, Juho Leinonen, Petri Ihantola, and Arto Hellas. Predicting academic success based on learning material usage. In *Proceedings of the 18th Annual Conference on Information Technology Education*, pages 13–18, 2017.

[12] Leo Porter, Daniel Zingaro, and Raymond 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.

[13] Peter R Newsted. Grade and ability predictions in an introductory programming course. *ACM SIGCSE Bulletin*, 7(2):87–91, 1975.

[14] Dennis H Sorge and Lois K Wark. Factors for success as a computer science major. *AEDS Journal*, 17(4):36–45, 1984.

[15] Patricia F Campbell and George P McCabe. Predicting the success of freshmen in a computer science major. *Communications of the ACM*, 27(11):1108–1113, 1984.

[16] Barry L Kurtz. Investigating the relationship between the development of abstract reasoning and performance in an introductory programming class. In *Proceedings of the eleventh SIGCSE technical symposium on Computer science education*, pages 110–117, 1980.

[17] Ricky J Barker and Elizabeth A Unger. A predictor for success in an introductory programming class based upon abstract reasoning development. *ACM SIGCSE Bulletin*, 15(1):154–158, 1983.

[18] Laurie Honour Werth. Predicting student performance in a beginning computer science class. *ACM SIGCSE Bulletin*, 18(1):138–143, 1986.

[19] Leo Porter, Dennis Bouvier, Quintin Cutts, Scott Grissom, Cynthia Lee, Robert McCartney, Daniel Zingaro, and Beth 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.

[20] Kathy Kenwright. Clickers in the classroom. *TechTrends*, 53(1):74–77, 2009.

[21] Soohyun Nam Liao, Daniel Zingaro, Michael A Laurenzano, William G Griswold, and Leo Porter. Lightweight, early identification of at-risk cs1 students. In *Proceedings of the 2016 acm conference on international computing education research*, pages 123–131, 2016.

[22] Amelia Zafra and SebastiáN Ventura. Multi-instance genetic programming for predicting student performance in web based educational environments. *Applied Soft Computing*, 12(8):2693–2706, 2012.

[23] Michael Fire, Gilad Katz, Yuval Elovici, Bracha Shapira, and Lior Rokach. Predicting student exam's scores by analyzing social network data. In *International Conference on Active Media Technology*, pages 584–595. Springer, 2012.

[24] Tsung-Yen Yang, Christopher G Brinton, Carlee Joe-Wong, and Mung Chiang. Behavior-based grade prediction for moocs via time series neural networks. *IEEE Journal of Selected Topics in Signal Processing*, 11(5):716–728, 2017.

[25] Matthew C Jadud. Methods and tools for exploring novice compilation behaviour. In *Proceedings of the second international workshop on Computing education research*, pages 73–84, 2006.

[26] Andrew Petersen, Jaime Spacco, and Arto Vihavainen. An exploration of error quotient in multiple contexts. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, pages 77–86, 2015.

[27] Christopher Watson, Frederick WB Li, and Jamie L Godwin. Predicting performance in an introductory programming course by logging and analyzing student programming behavior. In *2013 IEEE 13th international conference on advanced learning technologies*, pages 319–323. IEEE, 2013.

[28] Brett A Becker. A new metric to quantify repeated compiler errors for novice programmers. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, pages 296–301, 2016.

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

[30] Karo Castro-Wunsch, Alireza Ahadi, and Andrew Petersen. Evaluating neural networks as a method for identifying students in need of assistance. In *Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education*, pages 111–116, 2017.

[31] Daniel Rocco and Will Lloyd. Distributed version control in the classroom. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 637–642, 2011.

[32] Michael Cochez, Ville Isomöttönen, Ville Tirronen, and Jonne Itkonen. The use of distributed version control systems in advanced programming courses. In *ICTERI*, pages 221–235, 2013.

[33] Keir Mierle, Kevin Laven, Sam Roweis, and Greg Wilson. Mining student cvs repositories for performance indicators. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–5, 2005.

[34] Ángel Manuel Guerrero-Higueras, Vicente Matellán-Olivera, Gonzalo Esteban Costales, Camino Fernández-Llamas, FJ Rodriguez-Sedano, and MÁ Conde. Model for evaluating student performance through their interaction with version control systems. *Proceedings of the Learning Analytics Summer Institute Spain*, 2018.

[35] Ángel Manuel Guerrero-Higueras, Noemi DeCastro-García, Vicente Matellán, and Miguel Á Conde. Predictive models of academic success: a case study with version control systems. In *Proceedings of the Sixth International Conference on Technological Ecosystems for Enhancing Multiculturality*, pages 306–312, 2018.

[36] Gina Sprint and Jason Conci. Mining github classroom commit behavior in elective and introductory computer science courses. *The Journal of Computing Sciences in Colleges*, page 76, 2019.

[37] Ralf Teusner, Thomas Hille, and Thomas Staubitz. Effects of automated interventions in programming assignments: evidence from a field experiment. In *Proceedings of the Fifth Annual ACM Conference on Learning at Scale*, pages 1–10, 2018.

[38] Qiang Hao, Bradley Barnes, Ewan Wright, and Eunjung Kim. Effects of active learning environments and instructional methods in computer science education. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 934–939, 2018.

[39] 2020 github classroom report. https://education.github.com/classroom-report/2020, 2020. Online; accessed June 2021.

[40] Kirsti M Ala-Mutka. A survey of automated assessment approaches for programming assignments. *Computer science education*, 15(2):83–102, 2005.

[41] Qiang Hao, David H Smith IV, Lu Ding, Amy Ko, Camille Ottaway, Jack Wilson, Kai H Arakawa, Alistair Turcan, Timothy Poehlman, and Tyler Greer. Towards understanding the effective design of automated formative feedback for programming assignments. *Computer Science Education*, pages 1–23, 2021.

[42] Kai Arakawa, Qiang Hao, Tyler Greer, Lu Ding, Christopher D Hundhausen, and Abigayle Peterson. In situ identification of student self-regulated learning struggles in programming assignments. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, pages 467–473, 2021.

[43] James A Hanley. Receiver operating characteristic (roc) curves. *Wiley StatsRef: Statistics Reference Online*, 2014.

[44] Stanley Wileman, John Konvalina, and Larry J Stephens. Factors influencing success in beginning computer science courses. *The Journal of Educational Research*, 74(4):223–226, 1981.