One minute is enough: Early Prediction of Student Success and Event-level Difficulty during a Novice Programming Task

Ye Mao
North Carolina State Univ.
Raleigh, NC, USA
ymao4@ncsu.edu

Rui Zhi
North Carolina State Univ.
Raleigh, NC, USA
rzhi@ncsu.edu

Farzaneh Khoshnevisan
North Carolina State Univ.
Raleigh, NC, USA
fkhoshn@ncsu.edu

Thomas W. Price
North Carolina State Univ.
Raleigh, NC, USA
twprice@ncsu.edu

Tiffany Barnes
North Carolina State Univ.
Raleigh, NC, USA
tmbarnes@ncsu.edu

Min Chi
North Carolina State Univ.
Raleigh, NC, USA
mchi@ncsu.edu

ABSTRACT

Early prediction of student difficulty during long-duration learning activities allows a tutoring system to intervene by providing needed support, such as a hint, or by alerting an instructor. To be effective, these predictions must come early and be highly accurate, but such predictions are difficult for open-ended programming problems. In this work, Recent Temporal Patterns (RTPs) are used in conjunction with Support Vector Machine and Logistic Regression to build robust yet interpretable models for early predictions. We performed two tasks: to predict student *success* and *difficulty* during one, open-ended novice programming task of drawing a square-shaped spiral. We compared RTP against several machine learning models ranging from the classic to the more recent deep learning models such as Long Short Term Memory to predict whether students would be able to complete the programming task. Our results show that RTP-based models outperformed all others, and could successfully classify students after just *one* minute of a 20-minute exercise (students can spend more than 1 hour on it). To determine when a system might intervene to prevent incompleteness or eventual dropout, we applied RTP at regular intervals to predict whether a student would make progress within the next five minutes, reflecting that they may be having difficulty. RTP successfully classified these students needing interventions over 85% of the time, with increased accuracy using data-driven program features. These results contribute significantly to the potential to build a fully data-driven tutoring system for novice programming.

1. INTRODUCTION

Modeling student cognitive processes is highly complex since it is influenced by many factors such as motivation, aptitude, and learning habits. This is especially challenging for computer-based programming environments, because of the open-ended nature of programming. In this study, we focus on two important types of student modeling tasks to improve student experience in computer-based programming environments: 1) whether students will eventually succeed, and 2) at any given time, whether students need intervention. The interventions are more effective as we can predict earlier. Indeed, student modeling has been studied extensively for well-defined domains like algebra or physics [18, 17, 22]. For an ill-defined domain like programming, there are no pre-defined steps that students must take to complete a given program. Thus, it is hard to map the observations from students step by step. This makes the step-aligned models, like Bayesian Knowledge Tracing (BKT) [10], inappropriate for the programming domain.

Analyzing programming data often requires a way to represent a student's current state on a given problem. In the domain of programming, a student's state is typically represented by their current code, called a code-state. However, this representation leads to very large and poorly connected state spaces [14, 31], which makes it difficult to compare students and apply data-driven methods. Thus, in this study, we transformed student click-like log files into fixed feature sets. As shown in our prior work [38], this feature-state representation dramatically reduces the size of an open-ended programming state space, while creating semantically meaningful states. In this study, we explore both expert feature (EF) and data-driven feature (DDF) sets, and further compare their robustness on the two prediction tasks.

In recent years, deep learning models, specifically Recurrent Neural Networks (RNNs) and RNN-based models such as Long Short Term Memory (LSTM) [13] and gated recurrent unit (GRU) [9], have been shown to achieve state-of-the-art results in many applications with multivariate time series data including educational data. Such models enjoy several nice properties such as strong prediction of performance through deep hierarchical feature construction as well as the ability to effectively capture long-term temporal dependencies in time series data. Despite their extensive applications and great success, the open-ended nature in programming

Ye Mao, Rui Zhi, Farzaneh Khoshnevisan, Thomas Price, Tiffany Barnes and Min Chi "One minute is enough: Early Prediction of Student Success and Event-level Difficulty during Novice Programming Tasks" In: *Proceedings of The 12th International Conference on Educational Data Mining (EDM 2019),* Collin F. Lynch, Agathe Merceron, Michel Desmarais, & Roger Nkambou (eds.) 2019, pp. 119 - 128

state-space and various time intervals can pose enormous challenges for deep learning. More importantly, these deep learning models are often treated as "black box" models because of the lack of interpretability – they are particularly difficult to understand because of their non-linear nature.

On the other hand, Temporal Pattern-based approaches are designed to extract *interpretable, meaningful* temporal patterns directly from irregularly-sampled multivariate time series data. Recently, significant efforts have been made to develop and apply various pattern-based approaches to healthcare and shown to be effective. [7, 21]. However, as far as we know, temporal Pattern-based approaches have not been extensively investigated in the field of educational data mining especially for programming. In this work, we will directly compare one Temporal Pattern-based approach, Recent Temporal Patterns (RTPs) [7] against a series of baseline models including three classic machine learning models: k-Nearest Neighbors (KNN), support vector machines (SVM), and Logistic Regression (LR), and a state-of-the-art deep learning model: LSTM. More specifically, we compare these approaches on two early prediction tasks. Overall, we show that RTPs outperform all baseline models including LSTM on both early prediction tasks and more importantly, RTPs can generate quite reliable predictions with only the first *one* minute data. Additionally, RTPs can discover interpretable, meaningful temporal patterns that would be informative for domain experts or educators.

Our main contributions are summarized as follows: 1) To the best of our knowledge, this is the first attempt to apply RTP mining to extract student programming temporal patterns and compare it with several baseline models, including deep learning. 2) We run extensive experiments evaluating RTP and various baseline models on both the task of early prediction for student success and that of early prediction for student difficulty, while most prior research mainly focused on one or the other but not both. 3) We explored their robustness and the effectiveness of using EF vs. DDF for the programming system on both early prediction tasks. 4) We identify interpretable, meaningful temporal patterns that can be informative for domain experts.

2. BACKGROUND, CONTEXT, & DATA

2.1 Modeling Student Learning

Student modeling has been extensively explored as a series of approaches have been proposed [10, 34, 8, 24] to better understand and model student learning process. Among them, Bayesian Knowledge Tracing (BKT) [10] is one of the most widely investigated student modeling approaches. It models a student's performance in solving problems related to a given concept using a binary variable (i.e., correct, incorrect) and continually updates its estimation of the student's learning state for that concept. BKT and BKT-based models have been shown to be effective in many student modeling tasks, such as predicting students' overall competence [22], predicting the students' next-step responses [37, 6, 23, 18], and the prediction of post-test scores [16, 19]. However, in this study, BKT-based models cannot be directly applied to our open-ended programming tasks, because of the adversity of mapping students' time-various actions step by step.

In recent years, extensive research has been conducted on deep learning models, especially Recurrent Neural Network (RNN) or RNN-based models such as LSTM in the field of educational data mining [25, 33, 15, 35, 36, 17]. In our prior work, we have shown that LSTM has superior performance on the early prediction of student learning gain compared with classic BKT-based models [19]. For the task of predicting students' responses to exercises, LSTM was shown to outperform conventional BKT [25] and Performance Factors Analysis [24]. However, RNN and LSTM did not always have better performance when the simple, conventional models incorporated other parameters [15, 35].

While most of the previous studies on student modeling focus on predicting students' success and failure in the next-step attempt, some research has used student-tutor interaction data to predict student post-test scores [11, 30]. In this work, we explore the early prediction of both student success and difficulty. As far as we know, none of the previous studies have explored both prediction tasks for computer-based programming systems.

2.2 Programming and Help-seeking in iSnap

In this work, we analyze data from *iSnap*, a block-based novice programming environment [26]. iSnap is an extension of Snap*!* [12], which allows students to easily create interactive, 2D applications, such as apps and games. iSnap provides students with on-demand, next-step programming hints, which are generated automatically from student data with the SourceCheck algorithm [28]. In addition, iSnap collects detailed interaction data as students work, including complete snapshots of all student code, allowing us to perform detailed time series analysis. iSnap's data-driven hints also offer a useful motivation for this work. Prior analysis of student help-seeking behavior in iSnap suggests that few students ask for help when they need it, especially lower-performing students [29]. This is consistent with help avoidance reported in other tutoring systems [2, 4]. Prior works suggest that a number of factors lead to this help avoidance in iSnap, including lack of trust in the system, a desire for independence, and a lack of awareness of their own need for help [27]. Effective help-seeking is a metacognitive skill that many students need additional training to develop [3]. A system that could detect or predict student difficulty times could offer interventions such as automated help or instructor alerts. However, prior work suggests that such proactive intervention is most effective when systems assess the student's likelihood to succeed [20], rather than responding to all errors with help messages [1]. In this work, we present a data-driven approach capable of making early and accurate predictions of student success, which can enable a variety of interventions, such as iSnap hints.

2.3 Dataset

Our datasets were collected from students using iSnap in an "introduction to computers" course for non-majors, held at a research university, from the Spring 2016 (S16), Fall 2016 (F16), Spring 2017 (S17), and Fall 2017 (F17) semesters. We excluded students who requested hints since hints may alter students' problem-solving patterns, and our remaining data contained code traces from 65, 38, 29, and 39 students from S16, F16, S17, and F17 respectively. Each *code trace* consisted of a sequence of timestamped *snapshots* of student code. We chose one "Squiral" assignment to explore

in-depth, where students must write a procedure to draw a spiral with square corners. Common solutions contain 7-10 lines of code and use procedures, loops, variables, and arithmetic operators. In our previous work, we presented a *feature-state* representation that defines a student's state by the presence or absence of specific features of a correct solution [38]. Each feature describes a distinct property of correct solution code and may specify a necessary code structure or required program output. We defined *feature-state* as the presence or absence of each feature in a student's code (e.g. the feature-state "1101000" indicates Features 1, 2 and 4 are complete, while Features 3,5,6, and 7 are missing). In the same work, we implemented an algorithm to identify data-driven code features directly from student solutions, extracting 11 *data-driven features (DDF)*. We also had experts systematically define 7 *expert features (EF)* for the Spiral assignment. Using the expert feature definitions, we manually tagged each student's snapshot and had 6,339 tagged snapshots from 38 traces in F16. Our work showed that the data-driven features and expert features had moderate agreement on whether a student is in the same feature state or different states for F16. Additionally, we implemented an automated expert feature detector to detect the 7 expert features automatically. We used the detector to tag S16, S17, and F17 student snapshots with expert feature-states. On the F16 dataset, the expert feature detector had an agreement of 0.861 with the manually-tagged expert features. In all, we have 31,064 tagged snapshots from 171 traces from S16, F16, S17, and F17 semesters. Finally, we used a smoothing function to the tagged traces in which periods of rapid feature-state changes, defined as transitioning back and forth between two features within a 5 snapshot window, to set the values of a subsequent period after the variance. This process aims to reduce noise generated by actions in Snap*!* that do not represent meaningful feature-state changes (e.g. a student drags the code to a different position in the environment).

3. TWO EARLY PREDICTION TASKS

In this work, we explore two different early prediction tasks: the trajectory-level early prediction for student success and the event-level early prediction for student difficulty.

3.1 Trajectory-Level: Student Success

We classify the students who finished the programming assignment in one hour or less and got full credit as *successful*, and those who either failed to complete the assignment or submitted it within one hour as *unsuccessful*. Thus, each *trajectory* is assigned one ground truth label based on whether the student finished the assignment successfully or unsuccessfully. As a result, we refer to this task as *trajectory-level* early prediction task for student success. Based on this definition, 59 of 171 students are in the *successful* group, and the remaining 112 are in the *unsuccessful* group.

To predict student success, we are given the first *up to n* minutes of a student's sequence data and our goal is to predict whether the student will successfully complete the programming assignment at any given point in the remaining of the sequence. To conduct this task, we left-aligned all the students' trajectories by their starting times and our *observation window* (the part of data used to train and test different machine learning models) includes the sequences from the very beginning to the first n minutes. If a student's trajectory is less than n minutes, our observation window will include his/her entire sequence *except* the last one.

3.2 Event-Level: Student Difficulty

We define that a student is experiencing some sort of difficulty if at *any given moment*, a student fails to make any progress on his/her incomplete or imperfect answer in *the next five minutes*. Failure to make any progress in five minutes (on a supposed-to-be 20-minute programming task) reflects that the student may be experiencing some difficulty. Identifying the moment that a student is experiencing difficulty would allow us to determine when a system could intervene to address difficulty or prevent eventual dropout.

Because we predict whether a student is experiencing difficulty *moment by moment*, we refer to this early prediction task as *event-level* early prediction. More specifically, for a given moment, n minutes after starting time, we classify the students who failed to make any progress in the next five minutes as the *intervention* group, and those who made some progress as the *non-intervention* group. Note that we are not considering students who have already completed the training in n minutes and they are not assigned to either group. In short, for the event-level early prediction for student difficulty, our observation window contains the first *up to n* minutes of a student's sequence data and our goal is to predict whether the student will experience any difficulty and need intervention in the *next five minutes*.

4. RTP MINING

Our dataset can be represented as a set of trajectories, $X = \{x_1, x_2, ..., x_N\}$ where $N = 171$ is the total number of trajectories, one per student. It is composed of multivariate irregular time series data in that a student i's trajectory x_i consists of a sequence of events: $x_i = \{x_i^1, ..., x_i^{T_i}\}$, where x_i^t represents the student's code-records at time step t. We have $x_i^t \in \mathbb{R}^D$, where D is the number of predefined attributes/features and each attribute is a binary variable indicating whether a feature is present or not: $\mathbb{R} \in \{0, 1\}$. T_i is the length of the trajectory x_i; for different students, T_i varies. Each x_i is associated with a trajectory-level label (e.g. student success) or a series of moment by moment event-level labels (eg. student difficulty), denoted as $Y = \{y_1, y_2, ..., y_{T_i}\}$, where $y_i \in \{0, 1\}$. It is important to note that in the trajectory-level student success prediction, we treat students who are unsuccessful as the task of interest; thus, they are assigned to be 1 because it is more important for our model to classify and recognize the unsuccessful students as soon as possible. Similarly, for the event-level student difficulty task, we treat students who need intervention as the task of interest, and thus are assigned as 1.

Generally speaking, our RTP mining is conducted by following four steps: 1) Convert binary time-series variables into time interval sequences using Knowledge-based Temporal Abstraction, as described in the next section. 2)Extract frequent recent temporal patterns from different classes of data (i.e. 0 and 1). 3) Transform each x_i into a fixed-size binary feature vector v_i, where the size of vector corresponds to the number of frequent RTPs from Step 2. 4) Build the model on the binary matrix generated in Step 3 to predict the outcomes.

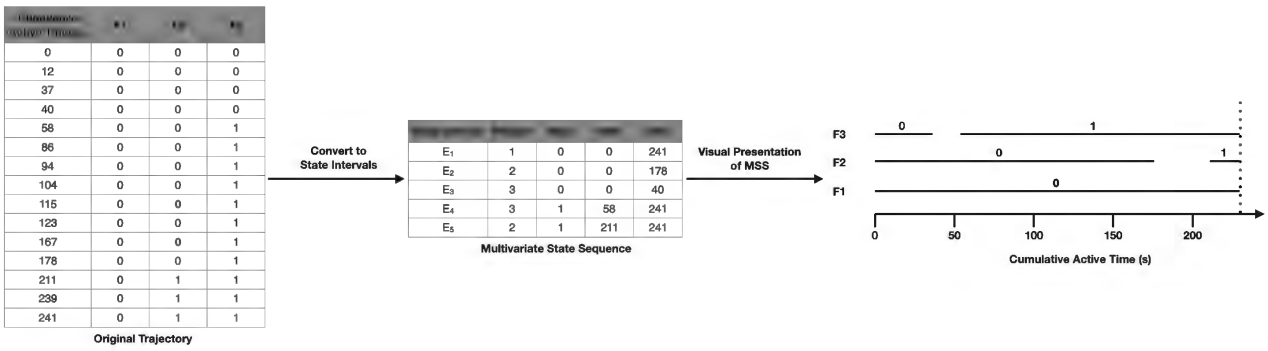


Figure 1: An example of conversion from the original data into a Multivariate State Sequence with three temporal features.

Typically, temporal abstraction involves defining 1) discretized data, 2) multivariate state sequences, 3) temporal relations, and 4) temporal patterns [32]. Our binary data are already discretized, so we describe the remaining three steps.

4.1 Knowledge-based Temporal Abstraction

Multivariate State Sequences: The middle table in Fig. 1 demonstrates how binary data can be converted into Multivariate State Sequences (MSS), z_i where each row presents a state interval, E, extracted from the student's trajectory. We denote a **State** S as (F, V), where F is a temporal feature and V is the value for feature F at a given time and the **State Interval** E is denoted as (F, V, s, e), where s and e refer to the *start* and *end* times of the state (F, V). Thus, we can convert each student's data x_i into a corresponding MSS z_i by sorting all the state intervals by their start times:

$$z_i = \langle E_1, E_2, ..., E_n \rangle : E_j.s \leq E_{j+1}.s, j \in \{1, ..., n-1\}$$

Note that we also define $z_i.end$ as the end of the last state interval in the MSS, i.e. $E_n.e$. For example, the right figure of Fig. 1 is a visualization of the MSS z_i, and $z_i.end$ is 241. Applying this method on all $x_i \in X$ transforms X into a set of MSSs: $Z = \{z_1, z_2, ..., z_N\}$.

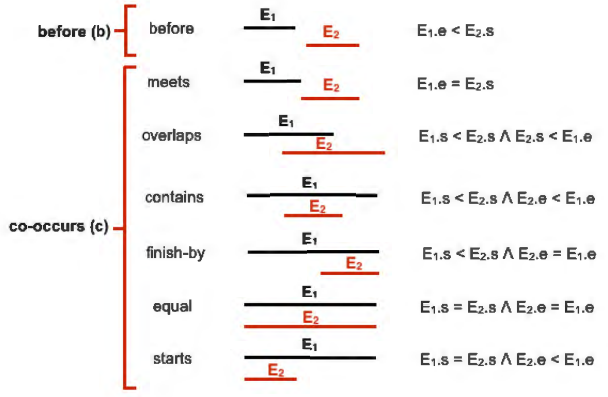


Figure 2: Allen's 7 basic temporal relations, grouped as before, or co-occurs.

Temporal Relations: We define two temporal relations, based on Allen's 7 basic temporal relations between states[5], grouping the last six into one *co-occurs* relation as shown in Fig. 2. Thus, the two temporal relations, before (b) and co-occurs (c), between two instantaneous events E_i and E_j, are defined as :

- E_i **before** (b) E_j: When E_i ends before the start of E_j ($E_i.e < E_j.s$).
- E_i **co-occurs** (c) with E_j: When E_i and E_j have some overlap ($E_i.s \leq E_j.s \leq E_i.e$).

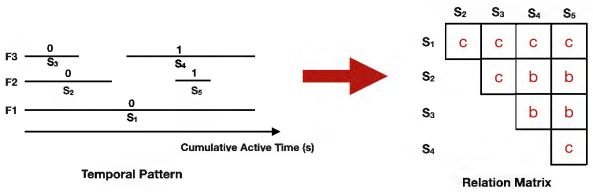


Figure 3: A temporal pattern P with 5 states $\langle S_1 = (\mathbf{F1}, 0), S_2 = (\mathbf{F2}, 0), S_3 = (\mathbf{F3}, 0), S_4 = (\mathbf{F4}, 1), S_5 = (\mathbf{F2}, 1)\rangle$ and temporal relations presented by half matrix R

Temporal Patterns: Temporal patterns are generated by combining states and temporal relations (before (b) and co-occurs (c)) to describe temporal dependencies in data. More specifically, for n states $\langle S_1, ..., S_n \rangle$, we define the corresponding temporal pattern: $P = (\langle S_1, ..., S_n \rangle, R)$, where R is an upper triangular matrix of relations, with $R_{i,j} \in b, c$ corresponding to the relation between S_i and S_j. The size of temporal pattern P is determined by the number of states S it contains. For example, a 5-pattern with three temporal features is shown in Fig. 3, where each state is an abstraction of a variable and the half matrix on the right shows the temporal relations between each pair of states. For example, since $S2 = (\text{Feature } 2, 0)$ happens before $S4 = (\text{Feature } 2, 0)$, $R_{2,4} = b$. In the next step, we describe a method to find the recent temporal patterns (RTPs) from MSSs in Z.

RTP Mining: We selected the RTP mining algorithm proposed by Batal et al. [7], because it incorporates a maximum gap parameter to consider the recency of patterns, it is efficient, and it prevents incoherent patterns. Next, we define

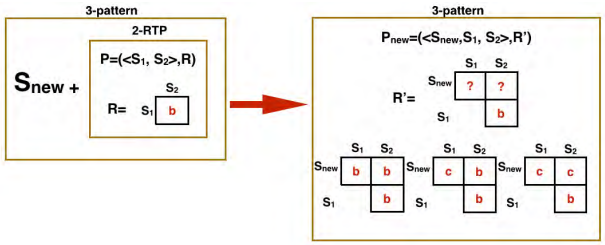


Figure 4: An example of generating 3-patterns out of a single 2-RTP, by appending a new state.

Recent Temporal Patterns (RTP) and briefly explain how they are applied in our work.

Recent Temporal Patterns: First, we call state interval $E = (F, V, s, e)$ a *Recent State Interval* of MSS z_i if: 1) E is the last state interval for feature F; that is, for all $E^{'} = (F, V^{'}, s^{'}, e^{'})$, we have $E^{'}.e \leq E.e$; or 2) E is less than g time units away from the end time of the last state interval: $z_i.end$; that is, $z_i.end - E.e \leq g$.

Given an MSS z_i, a temporal pattern $P = (\langle S_1, ..., S_n \rangle, R)$, and a maximum gap parameter g, we say P is a recent temporal pattern (RTP) in z_i, denoted as $R_g(P, z_i)$, if *all 3* of the following conditions hold: 1) z_i contains P, where $P \in z_i$ if: (a) z_i contains all k states of P, and (b) all temporal relations of P are satisfied in z_i; 2) $S_n = (F_n, V_n)$ matches a recent state interval in z_i; and 3) Every consecutive pair of states in P maps to a state interval less than g time units apart. That is, each pair of temporal sequences should not be g time units apart. In short, parameter g forces patterns to be close to the end of the sequence z_i, and forces consecutive states to be close to each other.

Mining Algorithm: For each MSS z_i, its outcome $y_i = 1$ when the z_i sequence is unsuccessful/needs intervention, and $y_i = 0$ otherwise. Taking student success classification as an example, we will have two sets of labeled MSSs: $Z_1 = \{z_i : y_i = 1\}$ for all unsuccessful sequences and $Z_0 = \{z_j : y_i = 0\}$ for all successful ones. Given Z_1, the mining algorithm applies a level-wise search to find frequent RTPs. More specifically, it first starts with all frequent 1-RTPs, and then extends the patterns by adding a new state to each sequence, one at a time, until no new patterns are discovered. That is, at each level k, the algorithm finds frequent $(k+1)$-RTPs by repeatedly extending k-RTPs through Backward candidate generation, and the Counting phase, as described below.

Backward $(k+1)$-pattern candidates are generated from a k-pattern $P = (\langle S_1, ..., S_k \rangle, R)$, by adding a new frequent state, S_{new}, to the beginning of the sequence to create $P^{'} = (\langle \mathbf{S_{new}}, S_1, ..., S_k \rangle, \mathbf{R}^{'})$. Then we specify the new before (b) or co-occurs (c) relations $R^{'}$ between S_{new} and all original k states, restricted by the following two criteria: 1) Two state intervals of the same temporal feature cannot co-occur. That is, if $S_{new}.F = S_i.F$ for $i \in \{1, ..., k\}$, then $R'_{new,i} \neq c$. 2) Since the state sequence in pattern P is sorted by the start time of the states, once a relation becomes *before*: $R'_{new,i} = b$ for any $i \in \{1, ..., k\}$, all the following relations have to be *before*, so $R'_{new,j} = b$ for $j \in \{i+1, ..., k\}$.

In the Counting phase, candidate $(k+1)$-patterns are removed if they do not meet the minimum support threshold by occurring at least σ times as RTP in Z_1. The same procedure is carried out for Z_0. Finally, we combine all the frequent RTPs into a final Ω set of RTPs.

Binary Matrix Transformation: We transform each MSS $z_i \in Z$ into a binary vector v_i of size $|\Omega|$, such that each 0 and 1 indicates whether the pattern $P_j \in \Omega$ is a recent temporal pattern in Z_i or not. This will result in a binary matrix of size $N \times |\Omega|$, which represents our original dataset.

Learning Models: Once the binary matrix is built, we apply different machine learning models including KNN, logistic regression (LR), and SVM to perform each target task.

5. EXPERIMENTS

5.1 Seven models in Three Categories

To evaluate the RTP-based models for early prediction of student success and student difficulty, we conducted a series of experiments. For each task, we used grid search to investigate the optimum value for the maximum gap (g) and minimum support (σ) parameters and we have $g = 60$ minutes and $\sigma_0 = 0.2$ for both prediction tasks. For each task, we compared RTP against two categories of baselines: the three *Classic ML models* and *LSTM*. Thus in total, we explored seven classification models in three categories.

Three Classic Machine Learning Models: We explore three classic machine learning models: KNN, LR, and SVM. Since these models do not handle sequence data directly, we used a "Last Value" approach to treat the last measurement of each attribute within the given observation window as the input to train models. Note that "Last Value" approach is also the baseline approach in [7] and many student modeling research. For early prediction settings, we truncated all the sequences in the training dataset in the same fashion as the testing dataset and then applied the Last Value approach on the truncated training dataset. For example, when our observation window is 1 minute, we apply the last value before 1 minute for each sequence and treat them as inputs for each model. For each of the three models, we explored different parameters to obtain the best results, in that, for KNN, we have $k = 10$, for LR we used $L1$ regularization, and for SVM we used *linear* kernel.

One Deep Learning Model: LSTM LSTM is a variation of Recurrent Neural Networks (RNNs) that prevents the vanishing gradient problem among other forms of RNNs [13]. LSTM has a chain-like structure, which enables information to flow among different blocks at different time points. Each block of the LSTM consists of a memory cell state and three gates: Forget, Input, and Output. These three gates interact with each other to control the flow of information. More specifically, the Forget gate determines what information from the previous memory cell state is expired and should be removed; the Input gate selects information from the candidate memory cell state for updating; the Output gate filters the information from the memory cell so that the model only considers information relevant to the prediction task. Therefore, the memory cell plays a crucial role in memorizing previous experiences. In our task, the input is a multivariate temporal sequence from student

work, and the output from the last step is used to make a prediction. We implemented LSTM in Keras with Tensorflow as the back-end engine, and we used one hidden layer with 100 hidden neurons and set the maximum length to accommodate the longest sequence in our data. Typically for LSTM, the whole multivariate time series from student sequence data is used as input data. However, for early prediction, only those events happening within our observation window from each sequence were used. We applied 5-fold cross-validation in order to tune the parameters of the model, including the optimizer, initializer, number of epochs, and number of batches.

Three RTP-based Models: The RTP-based models would first generate the binary matrix through RTP mining and then applied the classical machine learning models, KNN, LR, and SVM, (the same parameter settings as used in classic machine learning models described above) and thus, they are referred as RTP_KNN, RTP_LR, and RTP_SVM, respectively. Prior research on RTPs for prediction have all used entire sequences to extract meaningful temporal patterns. In this work, we explored the effectiveness of RTP for early prediction, by applying the *truncated* training sequences included in observation window to find RTPs. For example, when our observation window is 1 minute, only the first 1 minute of sequences were used for pattern extraction.

5.2 Evaluation Metrics

We evaluated our models using Accuracy, and Recall, F1 Score, and AUC (Area Under ROC curve). Accuracy represents the proportion of students whose labels were correctly identified. Recall tells us what proportion of students who will actually be unsuccessful (or need intervention) were correctly recognized by the model. F1 Score is the harmonic mean of Precision and Recall that sets their trade-off. AUC measures the ability of models to discriminate groups with different labels. Given the nature of the taks, we mainly use Accuracy and AUC to compare different models. All models were evaluated using 5-fold cross validation.

6. RESULTS

We present our results in three parts. First, we compare the effectiveness of the three categories of models on trajectory-level early prediction of student success. Second, we explore their performance on event-level early prediction of student difficulty. Finally, we discuss the extracted interpretable and meaningful temporal patterns discovered by RTP mining. For both early prediction tasks, we analyze two feature sets: *expert-based features (EF)* and *data-drive features (DDF)*.

6.1 Trajectory-Level: Student Success

[Observation Window = 1 min] Table 1 shows the performance of all models using the first-1-minute training sequences to predict students' success. The first row is the baseline model using simple Majority vote; note that we ignored the Recall and F1-measure of the simple Majority baseline. For the three main categories of models, we reported their performance on both EF and DDF. For Classic ML, KNN with DDF generates the highest scores on Recall (0.955) and F1-measure (0.793), SVM with DDF has the best AUC (0.563), and LR with EF contributes the best Accuracy (0.678). Thus, there is no clear winner among the three models here. Between the two types of features, it seems that DDF is slightly better or as good as EF for KNN and SVM, but much worse than EF for LR. Between the two LSTM models, LSTM with DDF works better than LSTM with EF. For RTP-based models, the best Accuracy (0.971) is from RTP_SVM and RTP_LR and both with EF, the best Recall (0.973) is generated by RTP_KNN and RTP_LR and both with EF, the best F1-measure (0.978) comes from RTP_LR with EF again, and the best AUC (0.978) is generated by RTP_SVM with EF. So for RTP-based models, EF generally works better than DDF. However, for RTP_LR and RTP_SVM, the performance of the EF and DDF are very close and competitive.

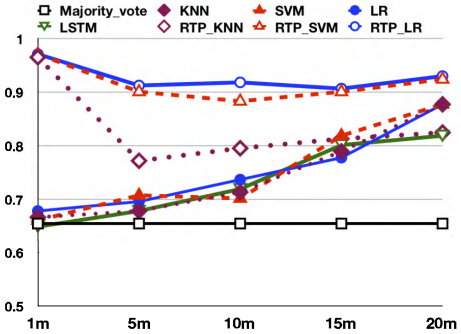
Table 1: Student success classification performance for the *one-minute* observation window

Model		Feature	Accuracy	Recall	F1	AUC
Majority Baseline		EF/DDF	0.655	-	-	0.5
Classic ML	KNN	EF	0.667	0.946	0.788	0.541
		DDF	0.673	**0.955**	**0.793**	0.545
	SVM	EF	0.661	0.946	0.785	0.533
		DDF	0.673	0.920	0.786	**0.563**
	LR	EF	**0.678**	0.938	0.792	0.562
		DDF	0.520	0.464	0.559	0.546
Deep Learning	LSTM	EF	0.649	**0.991****	0.787	0.496
		DDF	**0.655**	**0.991****	**0.790**	**0.504**
RTP	RTP_KNN	EF	0.965	**0.973**	0.973	0.961
		DDF	0.906	0.902	0.927	0.909
	RTP_SVM	EF	**0.971****	0.955	0.977	**0.978****
		DDF	0.965	0.955	0.973	0.969
	RTP_LR	EF	**0.971****	**0.973**	**0.978****	0.970
		DDF	0.959	0.964	0.969	0.957

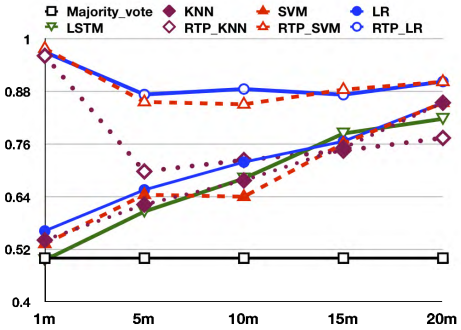
Note: best model for each group in **bold**, overall best model marked **

Finally, across the three categories, RTP-based models have the highest score on every measure except that LSTM has the highest Recall. While all the highest measurements come from using EF, the performance of DDF is very close to EF especially when using LSTM, RTP_LR and RTP_SVM. More importantly, the performance of all the RTP-based models are very high, above 95% on every measure.

[Observation Window = $1 \sim 20$ mins] Fig. 5 and Fig. 6 report Accuracy and AUC performance for all models using EF and DDF respectively. For each graph, we vary the observation window from the first 1 minute up to the first 20 minutes. Both Fig. 5 and Fig. 6 show that RTP_LR and RTP_SVM were the best models for both using EF and DDF as they stay on the top across all sizes of the observation window. It is not surprising that for the three classic models and LSTM, the longer the observation windows, the better performance they achieve. This is because the training data includes more and more information and closer to their final state. For RTP_KNN, both EF and DDF first decrease dramatically from 1 to 5 minutes and then increase slightly but still, the best performance comes from using only the first *one* minute. For RTP_LR and RTP_SVM, their performances are not only the best but also very steady. The fact that the best prediction comes from using just the first *one* minute of the sequences and using RTP-based models really suggest that *how* students try to solve the problem, what actions they take and in what order in that minute really matters for determining their final success. However, this is only observation from one programming task and more research is needed for further investigation.

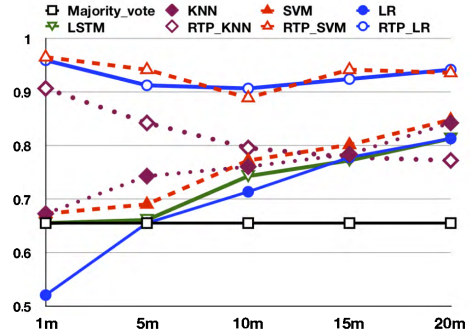


(a) Accuracy performance

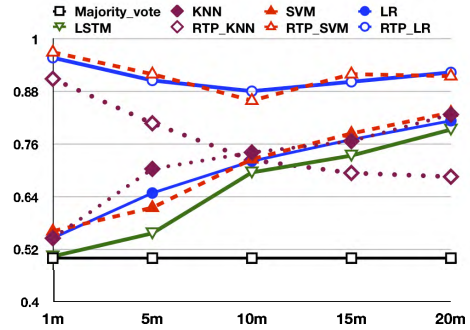


(b) Area under ROC performance

Figure 5: Student success early prediction on *expert* feature set



(a) Accuracy performance



(b) Area under ROC performance

Figure 6: Student success early prediction on *data-driven* feature set

When comparing using EF and DDF, Fig. 5 and Fig. 6 shows that the general patterns of performance of different machine learning models are very similar between using EF and DDF with a few exceptions. In general, using EF seems working better than using DDF and the exceptions are: for RTP_LR, and RTP_SVM, the best two models, the performance of using EF and using DDF are very close and sometimes using DDF is even better than using EF.

6.2 Event-Level: Student Difficulty

Table 2: Student difficulty classification performance for *one-minute* observation window

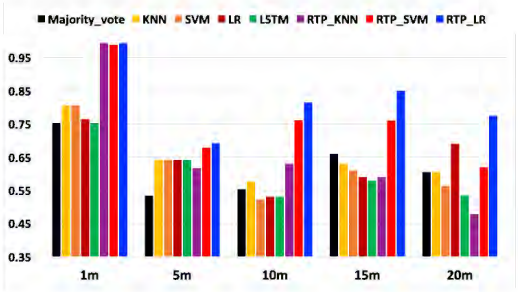
Model		Feature	Accuracy	Recall	F1	AUC
Majority Baseline		EF/DDF	0.753	-	-	0.5
Classic ML	KNN	EF	**0.806**	**0.238**	**0.377**	**0.615**
		DDF	0.800	0.214	0.346	0.603
	SVM	EF	**0.806**	0.214	0.353	0.607
		DDF	0.800	**0.238**	0.370	0.611
	LR	EF	0.765	**0.238**	0.333	0.588
		DDF	0.771	**0.238**	0.339	0.592
Deep Learning	LSTM	EF	0.753	0.224	0.344	0.596
		DDF	**0.871**	**0.269**	**0.389**	**0.624**
RTP	RTP_KNN	EF	**0.994****	**1****	**0.988****	**0.996****
		DDF	0.971	0.976	0.943	0.972
	RTP_SVM	EF	0.988	0.976	0.976	0.984
		DDF	0.971	0.952	0.941	0.964
	RTP_LR	EF	**0.994****	**1****	**0.988****	**0.996****
		DDF	0.988	**1****	0.976	0.992

Note: best model for each group in **bold**, overall best model marked **

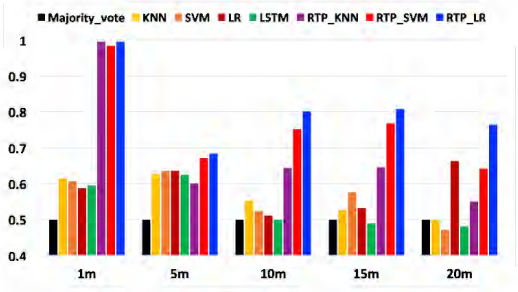
[Observation Window = 1 min] Table 2 shows the performance of all models using the first-1-minute-training sequences to predict students' difficulty in the next five minutes. As with Table 1, it is not very meaningful to present the Recall and F1-measure of the simple Majority baseline (row 1). For Classic ML models, KNN using EF generates the highest scores on every measure but not much better than the other two. In fact, all three models perform very closely regardless of using EF or DDF, and they all performed pretty poorly in that their performances on recall, F1 and AUC are all below 62%. Again for this task, the LSTM models outperform the classic models but not by much. Although LSTM with DDF works better than EF, the resulted models are still not very effective. Finally, for RTP-based models, RTP_KNN and RTP_LR reach the highest scores on every measure. Both of them have the best Accuracy (0.994), the best Recall (1), the best F1-measure (0.988), and the best AUC (0.996). RTP_SVM can make comparable predictions with over 97% on every measure.

In general, the best model among all the seven models uses RTP, which has the highest score on every measure. Comparing the models on different feature sets, we can find that the models with DDF are able to generate very similar results as those with EF. And sometimes, models with DDF had better performance on the first 1 minute, such as LR and LSTM.

[Observation Window = $1 \sim 20$ mins] Fig. 7 and Fig. 8 show the results from EF and DDF respectively. We mainly reported the performance of Accuracy and AUC on the observation sets {1m, 5m, 10m, 15m, 20m}. Note that different from trajectory-level early prediction, our majority baseline for event-level early prediction on student difficulty is changing moment by moment and for an observation window n,

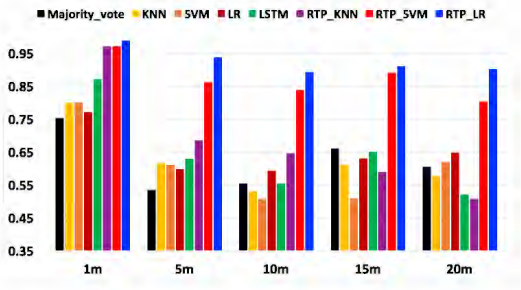


(a) Accuracy performance

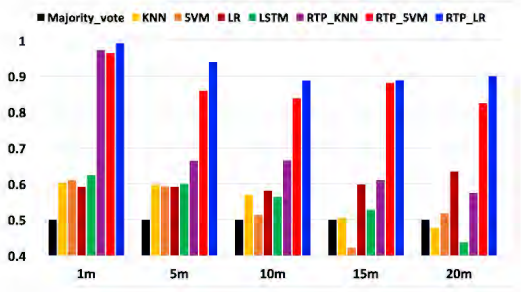


(b) Area under ROC performance

Figure 7: Student difficulty early prediction on *expert* feature set



(a) Accuracy performance



(b) Area under ROC performance

Figure 8: Student difficulty early prediction for *data-driven* feature set

all the students whose trajectories are shorter than n are excluded from the evaluation. So we have different baseline majority accuracy for different observation window (shown in black bar). For AUC, majority vote would give us a baseline of 0.5. Both Fig. 7 and Fig. 8 show that RTP_LR is the best one among all the models no matter which feature sets were used, with the highest scores on both Accuracy and AUC for all observation windows. Additionally, when comparing RTP_LR with EF and RTP_LR with DDF, we can see that RTP_LR with DDF generally perform much better than RTP_LR with EF. By comparing the results from Fig. 7 and Fig. 8, RTP_LR and RTP_SVM really benefit from using DDF instead of EF in that while RTP_LR and RTP_SVM with DDF perform very closely with RTP_LR and RTP_SVM with EF when the observation window = 1m, the RTP_LR and RTP_SVM with *DDF* performed much better and more stable than RTP_LR and RTP_SVM with *EF* on all the following observation windows and the difference are large. For the rest of machine learning models, the difference between using DDF and using EF is not noticeable. Finally, note that Fig. 7 shows when the observation window = 5m, the best performance using EF is 65% accuracy and $<$ 70% of AUC achieved through RTP_LR ($x - axis = 5m$); however, when we use the same observation window, RTP_LR with DDF can achieve the accuracy close to 95% and AUC above 90% (shown in Fig. 8). So overall, for the task of event level early prediction for student difficulty, RTP_LR with DDF consistently achieve the best performance and its performance is pretty steady across different observation windows.

6.3 Knowledge Discovery

One substantial advantage of pattern-based classification over deep learning models is the *interpretability* of the discovered patterns. In RTP, the patterns need to be representative of the original time series data while predictive of the future outcomes. In this study, most of the patterns extracted using RTP mining are in accordance with the student classification tasks and some of them reveal latent patterns towards the progression of student success or difficulty. Table 3 and 4 present a number of interesting patterns and their corresponding support among the training group from the first 20 minutes, where the support of pattern P is calculated as the proportion of students in the dataset which contains P.

Table 3 shows some patterns related to student success. $P_1 - P_4$ describe the frequent patterns found among unsuccessful students, and most of them are related to the feature *MoveVariably*. P_1, P_2 and P_3 illustrate that students did not complete feature *CreateUseParameterCorrectly*, *RepeatCorrectNumberOfTimes*, or *MoveSquirally* when feature *MoveVariably* has been completed. In P_1, students may start to work on *DrawAnything* and *MoveSquarelikeThing*, which is observable among 29.5% of unsuccessful students from their sequences in the first 20 minutes. Different from P_1, students with pattern P_2 and P_3 were probably working on *CustomBlock* before they finished feature *MoveSquirally*. P_4 indicates that students had finished features *MoveVariably* and *MoveSquarelikeThing* but failed at the end, which might be the case that they had done some movements but did not have pen down. And still, neither *CreateUseParameterCorrectly* nor *RepeatCorrectNumberOfTimes* was finished. $P_5 - P_8$ describes the frequent patterns found among successful students, and most of them are related to the feature *RepeatCorrectNumberOfTimes*. For example, P_5 shows that students had finished feature *CustomBlock*, *CreateUseParameterCorrectly* and *RepeatCorrectNumberOfTimes* at some point, and before that, within 20 minutes, they should be working

Table 3: Recent temporal patterns for student success, with observation window = 20m

RTP	If	Then	Support
P_1	(CreateUseParameterCorrectly,0) c (((DrawAnything, 0) c (MoveSquarelikeThing,0)) b (MoveVariably,1))	Unsuccessful	0.295
P_2	((CustomBlock,0) b (MoveVariably,1)) c (CreateUseParameterCorrectly,0) c (RepeatCorrectNumberOfTimes,0)	Unsuccessful	0.286
P_3	((CustomBlock,0) b (CreateUseParameterCorrectly,0)) c (RepeatCorrectNumberOfTimes,0) c (MoveSquirally,0) c (MoveVariably,1)	Unsuccessful	0.233
P_4	((DrawAnything,0) b ((MoveSquarelikeThing,1) c (MoveVariably,1))) c (CreateUseParameterCorrectly,0) c (RepeatCorrectNumberOfTimes,0)	Unsuccessful	0.205
P_5	(CustomBlock,0) b (CustomBlock,1) c (CreateUseParameterCorrectly,1) c (RepeatCorrectNumberOfTimes,1)	Successful	0.729
P_6	(CustomBlock,0) c (MoveSquarelikeThing,0) b (MoveSquarelikeThing,1) c (RepeatCorrectNumberOfTimes,1)	Successful	0.542
P_7	(DrawAnything,0) b (CreateUseParameterCorrectly,0) b (CreateUseParameterCorrectly,1) c (MoveVariably,1)	Successful	0.423

Table 4: Recent temporal patterns for student difficulty, with observation window = 20m

RTP	If	Then	Support
P_1	(((DrawAnything,0) c (MoveSquarelikeThing,0)) b (CustomBlock,1)) c (RepeatCorrectNumberOfTimes,0) c (MoveSquirally,0)	Intervention	0.372
P_2	(((CustomBlock,0) c (DrawAnything,0)) b ((CustomBlock,1) c (MoveVariably,1))) c (MoveSquirally,0)	Intervention	0.302
P_3	((DrawAnything,0) c (MoveSquarelikeThing,0)) b ((CustomBlock,1) c(DrawAnything,1)) c (MoveSquirally,0)	Intervention	0.279
P_4	((MoveSquarelikeThing,0) b ((MoveSquarelikeThing,1) c (MoveSquirally,1))) c (RepeatCorrectNumberOfTimes,0)	Non-intervention	0.461
P_5	(((DrawAnything,0) c (MoveSquarelikeThing,0)) b (MoveSquirally,1)) c (RepeatCorrectNumberOfTimes,0)	Non-intervention	0.422
P_6	(DrawAnything,0) c (MoveSquarelikeThing,0) c (CreateUseParameterCorrectly,0) c (RepeatCorrectNumberOfTimes,0) c (CustomBlock,1)	Non-intervention	0.320

on *CustomBlock*. Actually, for the students in successfully group, they should be able to finish all the features in one hour. Glancing over the extracted patterns among the successful group, we can see that most of the students ended up completing at least two important features, like patterns P_6 and P_7 in the Table 3, in the first 20 minutes. It is important to note that these patterns are only discovered among the successful group.

Table 4 shows some patterns related to student difficulty. $P_1 - P_3$ describe the frequent patterns found among the intervention group, and most of them are related to the feature *CustomBlock*. P_1 describes a pattern that discovered among 37.2% of students who need intervention in the next five minutes. In this case, students had not completed the *DrawAnything* or *MoveSquarelikeThing* features at first, but, within 20 minutes, they completed the *CustomBlock* feature. And at the same time, they did not complete feature *RepeatCorrectNumberOfTimes* or *MoveSquirally*. P_2 and P_3 are quite similar, they indicate that students finished *CustomBlock* along with *DrawAnything* or *MoveVariably* when the feature *MoveSquirally* had not been finished. P_4 to P_6 describe the frequent patterns found among students who do not need intervention, and most of them are related to the feature *MoveSquirally*. $P_4 - P_5$ show that students successfully completed feature *MoveSquirally* and before that, they may work on *MoveSquarelikeThing*. As in P_1, they have '0' on the feature *RepeatCorrectNumberOfTimes*. Comparing P_1 with P_5, we can find that instead of having incomplete *DrawAnything* and *MoveSquarelikeThing* before completing *CustomBlock*, students who have incomplete *DrawAnything*, incomplete *MoveSquarelikeThing*, and complete *CustomBlock* at the same time will not need intervention in the next five minutes. Again, these patterns are uniquely inferred among the non-intervention group.

7. CONCLUSIONS

Early prediction of trajectory-level student success and the event-level difficulty during long-term activities are challenging tasks due to the open-ended nature of programming tasks. In this study, we explored the EF identified by domain experts, as well as DDF identified automatically, to build a model that is able to predict student success/difficulty with high accuracy, and to provide valuable insights for educators. We employed an RTP-based classification framework and compared it with various baselines including classic and deep learning models, in two different tasks including trajectory-level early diagnosis and event-level early prediction. Our results suggest that the RTP-based models consistently outperform the non-temporal classic baselines as well as LSTM in both tasks, at all observation windows from first 1 minute to 20 minutes. Moreover, with only the first-1-minute sequences, RTP-based models can make strong predictions on both tasks. Additionally, by applying the data-driven features, RTP-based models are able to achieve comparable performance on the task of predicting student success. For the task of predicting event-level student difficulty, data-driven features can even improve their predictions further.

As future work, we plan to apply progressive features with multiple values (eg. not started, in progress, complete) to discover more definitive patterns and obtain more advantageous knowledge discovery as well as improved prediction performance. And we are planning to employ different feature transformations other than binary, such as vertical support (i.e. the number of times a pattern occurred in a sequence) or recency measure (i.e. how distant a pattern occurred from the prediction time). Also, this work will be applied to larger groups of students and longer programming tasks, along with integration of more informative features such as intervention and demographic features to develop more robust models. Additionally, we plan to expand our evaluations to longer programs with more complex constructs from both text-based and block-based programming languages.

8. ACKNOWLEDGEMENTS

This research was supported by the NSF Grants 1432156, 1623470, 1651909, and 1726550.

9. REFERENCES

[1] Hints: Is it better to give or wait to be asked? In *ITS*, volume 6094 LNCS, pages 349–358, 2010.

[2] V. Aleven and K. R. Koedinger. Limitations of Student Control: Do Students Know When They Need Help? In *ITS*, pages 292–303, 2000.

[3] V. Aleven, B. M. McLaren, I. Roll, and K. R. Koedinger. Toward Meta-cognitive Tutoring: A Model of Help Seeking with a Cognitive Tutor. *IJAIED*, 16:101–128, 2006.

[4] V. Aleven, E. Stahl, S. Schworm, F. Fischer, and R. Wallace. Help seeking and help design in interactive learning environments. *Review of educational research*, 73(3):277–320, 2003.

[5] J. F. Allen. Towards a general theory of action and time. *Artificial intelligence*, 23(2):123–154, 1984.

[6] R. S. Baker, A. T. Corbett, and V. Aleven. More accurate student modeling through contextual estimation of slip and guess probabilities in bayesian knowledge tracing. In *ITS*, pages 406–415, 2008.

[7] I. Batal, D. Fradkin, J. Harrison, F. Moerchen, and M. Hauskrecht. Mining recent temporal patterns for event detection in multivariate time series data. In *SIGKDD*, pages 280–288. ACM, 2012.

[8] M. Chi, K. R. Koedinger, G. J. Gordon, P. Jordon, and K. VanLahn. Instructional factors analysis: A cognitive model for multiple instructional interventions. In *EDM*, pages 61–70, 2011.

[9] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[10] A. T. Corbett and J. R. Anderson. Knowledge tracing: Modeling the acquisition of procedural knowledge. *UMUAI*, 4(4):253–278, 1994.

[11] M. Feng, J. Beck, N. Heffernan, and K. Koedinger. Can an intelligent tutoring system predict math proficiency as well as a standardized test? In *EDM*, pages 107–116, 2008.

[12] D. Garcia, B. Harvey, and T. Barnes. The Beauty and Joy of Computing. *ACM Inroads*, 6(4):71–79, 2015.

[13] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[14] B. P. Iii, A. Hicks, and T. Barnes. Generating hints for programming problems using intermediate output. In *EDM*. Citeseer, 2014.

[15] M. Khajah, R. V. Lindsey, and M. C. Mozer. How deep is knowledge tracing? *arXiv preprint arXiv:1604.02416*, 2016.

[16] C. Lin and M. Chi. Intervention-bkt: incorporating instructional interventions into bayesian knowledge tracing. In *ITS*, pages 208–218. Springer, 2016.

[17] C. Lin and M. Chi. A comparisons of bkt, rnn and lstm for learning gain prediction. In *AIED*, pages 536–539. Springer, 2017.

[18] C. Lin, S. Shen, and M. Chi. Incorporating student response time and tutor instructional interventions into student modeling. In *UMAP*, pages 157–161. ACM, 2016.

[19] Y. Mao, C. Lin, and M. Chi. Deep learning vs. bayesian knowledge tracing: Student models for interventions. *JEDM*, 10(2):28–54, 2018.

[20] V. K. Murray R.C. A comparison of decision-theoretic, fixed-policy and random tutorial action selection. *LNCS*, 4053 LNCS:114–123, 2006.

[21] K. Orphanou, A. Dagliati, L. Sacchi, A. Stassopoulou, E. Keravnou, and R. Bellazzi. Combining naive bayes classifiers with temporal association rules for coronary heart disease diagnosis. In *ICHI*, pages 81–92, 2016.

[22] Z. A. Pardos and N. T. Heffernan. Modeling individualization in a bayesian networks implementation of knowledge tracing. In *UMAP*, pages 255–266. Springer, 2010.

[23] Z. A. Pardos and N. T. Heffernan. Kt-idem: Introducing item difficulty to the knowledge tracing model. In *UMAP*, pages 243–254. Springer, 2011.

[24] P. I. Pavlik, H. Cen, and K. R. Koedinger. Performance factors analysis –a new alternative to knowledge tracing. In *AIED*, pages 531–538, 2009.

[25] C. Piech, J. Bassen, J. Huang, S. Ganguli, M. Sahami, L. J. Guibas, and J. Sohl-Dickstein. Deep knowledge tracing. In *NIPS*, pages 505–513, 2015.

[26] T. W. Price, Y. Dong, and D. Lipovac. iSnap: towards intelligent tutoring in novice programming environments. In *SIGCSE*, pages 483–488, 2017.

[27] T. W. Price, Z. Liu, V. Catete, and T. Barnes. Factors Influencing Students' Help-Seeking Behavior while Programming with Human and Computer Tutors. In *ICER*, 2017.

[28] T. W. Price, R. Zhi, and T. Barnes. Evaluation of a Data-driven Feedback Algorithm for Open-ended Programming. In *EDM*, 2017.

[29] T. W. Price, R. Zhi, and T. Barnes. Hint Generation Under Uncertainty: The Effect of Hint Quality on Help-Seeking Behavior. In *AIED*, 2017.

[30] S. Ritter, A. Joshi, S. Fancsali, and T. Nixon. Predicting standardized test scores from cognitive tutor interactions. In *EDM*, pages 169–176, 2013.

[31] K. Rivers and K. R. Koedinger. Data-driven hint generation in vast solution spaces: a self-improving python programming tutor. *IJAIED*, 27(1):37–64, 2017.

[32] Y. Shahar. A framework for knowledge-based temporal abstraction. *Artificial intelligence*, 90(1-2):79–133, 1997.

[33] S. Tang, J. C. Peterson, and Z. A. Pardos. Deep neural networks and how they apply to sequential education data. In *L@S*, pages 321–324. ACM, 2016.

[34] K. Tatsuoka. Rule space: An approach for dealing with misconceptions based on item response theory. *JEM*, 20(4):345–354, 1983.

[35] K. H. Wilson, Y. Karklin, B. Han, and C. Ekanadham. Back to the basics: Bayesian extensions of irt outperform neural networks for proficiency estimation. *arXiv preprint arXiv:1604.02336*, 2016.

[36] X. Xiong, S. Zhao, E. Van Inwegen, and J. Beck. Going deeper with deep knowledge tracing. In *EDM*, pages 545–550, 2016.

[37] M. V. Yudelson, K. R. Koedinger, and G. J. Gordon. Individualized bayesian knowledge tracing models. In *AIED*, pages 171–180. Springer, 2013.

[38] R. Zhi, T. W. Price, N. Lytle, Y. Dong, and T. Barnes. Reducing the state space of programming problems through data-driven feature detection. In *EDM (Workshops)*, 2018.