What Time is It? Student Modeling Needs to Know

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

Samiha Marwan
North Carolina State Univ.
Raleigh, NC, USA
amarwan@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

Modeling student learning processes is highly complex since it is influenced by many factors such as motivation and learning habits. The high volume of features and tools provided by computer-based learning environments confounds the task of tracking student knowledge even further. Deep Learning models such as Long-Short Term Memory (LSTMs) and classic Markovian models such as Bayesian Knowledge Tracing (BKT) have been successfully applied for student modeling. However, much of this prior work is designed to handle sequences of events with *discrete timesteps*, rather than considering the continuous aspect of time. Given that time elapsed between successive elements in a student's trajectory can vary from seconds to days, we applied a Time-aware LSTM (T-LSTM) to model the dynamics of student knowledge state *in continuous time*. We investigate the effectiveness of T-LSTM on two domains with very different characteristics. One involves an open-ended programming environment where students can *self-pace* their progress and T-LSTM is compared against LSTM, Recent Temporal Pattern Mining, and the classic Logistic Regression (LR) on the early prediction of student success; the other involves a classic *tutor-driven* intelligent tutoring system where the tutor scaffolds the student learning step by step and T-LSTM is compared with LSTM, LR, and BKT on the early prediction of student learning gains. Our results show that T-LSTM significantly outperforms the other methods on the self-paced, open-ended programming environment; while on the tutor-driven ITS, it ties with LSTM and outperforms both LR and BKT. In other words, while time-irregularity exists in both datasets, T-LSTM works significantly better than other student models when the pace is driven by students. On the other hand, when such irregularity results from the tutor, T-LSTM was not superior to other models but its performance was not hurt either.

1. INTRODUCTION

Student Modeling sits at the epicenter of educational data mining. It monitors a student's progress, ability, or knowledge over a set of skills and can predict the student's future performance based on historical sequence data. In recent years, recurrent neural network architectures, such as Long Short-Term Memory (LSTMs), have become the workhorses for modeling sequence data in a variety of tasks involving sequential data, such as video processing, climate change detection, and patient disease progression prediction [20, 19, 25, 12]. Deep Knowledge Tracing [35, DKT], the first LSTM approach in student modeling, reported an impressive improvement over a classical statistical model Bayesian Knowledge Tracing [10, BKT]. Both LSTM/DKT and BKT are designed to handle sequences of events with *discrete timesteps*, not considering the *continuous* aspect of time.

On the other hand, student response time, the elapsed times between consecutive elements of a sequence can vary greatly by student, from seconds to days. Ever since the mid-1950s, student response time has been used as a preferred educational assessment to evaluate how active and accessible student knowledge is in cognitive psychology [43]. For example, it has been shown that response time reveals student proficiency [40] and there is a significant negative correlation between student average response time and student final exam score taken at the end of the semester [16]. Additionally, response time has been suggested as an indicator of student engagement in answering questions [21] as well as an important factor for predicting motivation in learning environments [9]. Also, by leveraging time information, BKT prediction performance can be improved [38, 44]. Therefore, by not taking the time intervals into consideration, the design of traditional LSTM and BKT may lead to sub-optimal performance for modeling student learning.

Previous work for modeling sequence data has explored several ways to handle time irregularity [3, 34, 8, 6] and among them, Time-aware LSTM (T-LSTM) is one of the most state-of-the-art models [3]. T-LSTM transforms time intervals between successive elements into weights and uses them to adjust the memory passed from previous moments. In this work, we apply T-LSTM to model the dynamics of student knowledge state *in continuous time* and conduct two empirical comparisons between T-LSTM and the standard LSTM, Recent Pattern Mining [23], and classical student modeling methods such as BKT and logistic regression models on two real-world data sets collected from two learning environments with very different characteristics. One is an open-ended block-based programming environment for a novice programming task where students are free to explore the environment with minimal system support or constraints. Each student's log file is a trajectory of actions with corresponding time stamps and time intervals calculated between the two consecutive student actions. The other probability tutor is *tutor-driven* in that *the tutor* decides what to do next. Each student's log file is a trajectory of student-ITS interactions. In each interaction, the tutor first *elicits* the subsequent step from a student with prompting, and when the student performs a step, the tutor records its success or failure and may give feedback (e.g. correct/incorrect markings); if the student's answer is incorrect, the tutor provides a series of hints from general to specific and the bottom-out hint tells the student exactly what to do. The interaction is ended only when a step is correctly answered and the tutor moves to the next interaction. As a result, each student's log file is a trajectory of tutor actions mixed with student's responses with corresponding time stamps. In this environment, the time intervals are calculated between the student's *first attempt* on one problem and the next. Our research question is: ***By taking time-awareness into consideration, would T-LSTM outperform other traditional student modeling methods on both self-paced and tutor-driven learning environments?***

2. METHODS

2.1 Long Short-Term Memory

Long Short Term Memory [18, LSTM] is a special type of RNN which is explicitly designed to avoid the long-term dependency problem. LSTM can avoid the vanishing (and exploding) gradient problem and works tremendously well on a large variety of problems.

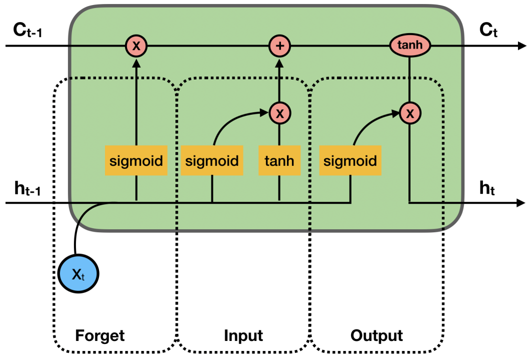


Figure 1: The Structure of a LSTM Unit

The internal structure of each LSTM module is shown in Figure 1. There are three major components: a forget gate, an input gate, and an output gate in a standard LSTM unit cell, where these components interact with each other to control how information flows. In the first step, a function of the previous hidden state h_{t-1} and the new input X_t passes through the forget gate, indicating what is probably irrelevant and can be taken out of the cell state. The forget component will calculate a weight f_t between 0 to 1 for each element in hidden state vector C_{t-1}. An element with a weight of 0 should be completely forgotten whereas an element with a weight of 1 needs to be entirely remembered. The formula to calculate f_t is shown below where W_f and b_f are the weights and intercepts, respectively, for the forget component.

$$f_t = \text{sigmoid}(W_f \cdot [h_{t-1}, x_t] + b_f) \qquad (1)$$

There are two steps involved in input component's calculation. In the first step, a *tanh* layer calculates a candidate vector $\tilde{C_t}$ that could be added to the current hidden state. In the second step, the input components calculate a weight vector i_t (ranging from 0 to 1) to determine to what extent $\tilde{C_t}$ should update the current memory state.

$$\tilde{C_t} = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \qquad (2)$$

$$i_t = \text{sigmoid}(W_i \cdot [h_{t-1}, x_t] + b_i) \qquad (3)$$

With the forget and input components, the module is able to throw away the expired information in the previous cell state by calculating $C_{t-1} \cdot f_t$, and process new information by computing $\tilde{C_t} \cdot i_t$. Consequently, the formula to update the current memory cell is shown below. Note that the current memory cell state C_t is then passed to the next LSTM module.

$$C_t = C_{t-1} \cdot f_t + \tilde{C_t} \cdot i_t \qquad (4)$$

Finally, the output component is simply an activation function that filters elements in C_t. The C_t can be converted to a value between -1 to 1 by the tanh function. The output component calculates a weight vector

$$o_t = \text{sigmoid}(W_o \cdot [h_{t-1}, x_t] + b_o) \qquad (5)$$

that determines how much information is allowed to be revealed.

$$C_t = o_t * \tanh(C_t) \qquad (6)$$

With such a gated structure, LSTM is capable of handling long-term dependencies.

2.2 Time-Aware Long Short Term Memory

The standard LSTM assumes that the elapsed times between elements of a sequence are uniformly distributed, and therefore it is designed to handle sequences with *discrete timesteps*. However, in the educational domain, the interval between two consecutive steps during a student trajectory can span from seconds to days. In general, the events that occurred long ago tend to have less impact to the current state and thus we should properly reduce their contributions. Therefore, it is important to consider the elapsed time when predicting the current event's output. In this work, we applied Time-aware LSTM [3, T-LSTM], which is proposed to handle the temporal dynamics of sequential data with time irregularities, to model student knowledge states *in continuous time*.

The T-LSTM architecture is shown in Figure 2. To fit in our domain, we represent the input sequence by the student trajectories. Apart from the three gates in standard LSTM: forget, input, and output; T-LSTM also integrates the time elapsed between successive records into the network architecture, and we call this as the time decay component. The information stored in the memory of the previous

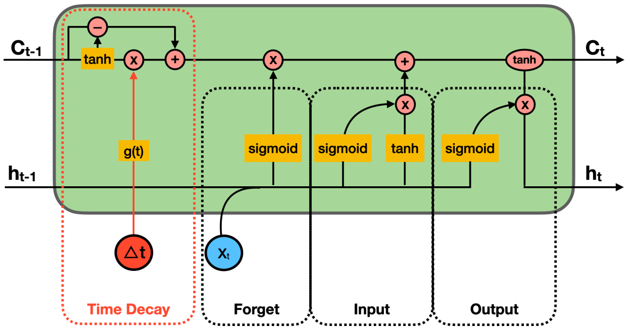


Figure 2: The Structure of a T-LSTM Unit

hidden state C_{t-1} is decomposed into two parts: long-term memory and short-term memory. Without losing the long-term memory contained in C_{t-1}, the time decay component mainly plays a role to adjust the short-term memory by employing the elapsed time between successive steps. If the gap between two steps is significantly huge, e.g. few hours in our domain, it means there has been a long time with no interaction between students and the tutor/computer. In that case, there is not much point to heavily rely on the previous short-term memory to predict the current output. In the framework of T-LSTM, a non-increasing function of the elapsed time is applied to transform the time duration into an appropriate weight. And in this work, we applied $g(\Delta t) = 1/\log(e + \Delta t)$ to get the corresponding weights.

The following calculations are involved in the time decay component of T-LSTM. First, short-term memory C^S_{t-1} is calculated.

$$C^S_{t-1} = \tanh\left(W_d \cdot C_{t-1} + b_d\right) \tag{7}$$

The long-term memory can be obtained by deducting short-term memory from the previous hidden state.

$$C^L_{t-1} = C_{t-1} - C^S_{t-1} \tag{8}$$

Then C^S_{t-1} is discounted by the elapsed time weight to obtain the discounted short-term memory $\hat{C}^S_{t-1}$.

$$\hat{C}^S_{t-1} = C^S_{t-1} * g(\Delta t) \tag{9}$$

Finally, the adjusted previous hidden state C^*_{t-1} is composed by adding long-term memory and discounted short-term memory.

$$C^*_{t-1} = C^L_{t-1} + \hat{C}^S_{t-1} \tag{10}$$

The following parts are very similar to standard LSTM. Following the steps in Section 2.1, we first calculate the forget gate f_t, candidate vector $\tilde{C}_t$ and input gate i_t by applying Equation (1), (2) and (3). For the calculation of the current memory cell state C_t, the adjusted previous hidden state C^*_{t-1} instead of C_{t-1} is applied in the T-LSTM framework.

$$C_t = C^*_{t-1} \cdot f_t + \tilde{C}_t \cdot i_t \tag{11}$$

The final output for the current state can be achieved using the following Equation (6). In this work, we investigate the effectiveness of T-LSTM via the early prediction of both student success and learning gains. As far as we know, no prior studies have explored T-LSTM on both computer-based programming systems and intelligent tutoring systems.

2.3 Recent Temporal Pattern Mining

The Recent Temporal Pattern mining (RTP) framework [2] was originally proposed to find predictive patterns from complex multivariate time series data. This framework first converts time series into time-interval sequences of temporal abstractions, and then constructs more complex temporal patterns backwards. The following part will explain how the RTP framework is applied in our work.

Multivariate State Sequences: 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 Multivariate State Sequence (MSS) z_i by sorting all the state intervals by their start times: $z_i = \langle E_1, E_2, ..., E_n \rangle : E_j.s \le E_{j+1}.s, j \in \{1, ..., n-1\}$. And we apply two temporal relations in this work: 1)E_i **before(b)** E_j: When E_i ends before the start of E_j ($E_i.e < E_j.s$); 2) E_i **co-occurs(c)** with E_j: When E_i and E_j have some overlap ($E_i.s \le E_j.s \le E_i.e$).

Recent Temporal Patterns: Here, we call a 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 $E^{'}.e \le E.e$; or 2) E is less than g time units away from the end time of the last state interval: $z_i.end$; that is, $z_i.end - E.e \le g$.

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 $R_g(P, z_i)$, 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: 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.

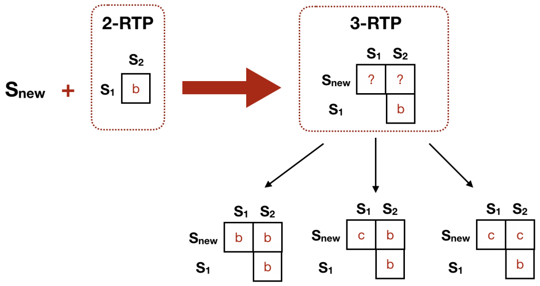


Figure 3: An example of generating 3-patterns out of a single 2-RTP, by appending a new state.

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 of 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 RTPs 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.

2.4 Bayesian Knowledge Tracing

BKT is a student modeling method extensively used in ITSs. Figure 4 shows a graphical representation of the model and a possible sequence of student observations. The shaded nodes S represent hidden knowledge states. The unshaded nodes O represent observation of students' behaviors. The edges between the nodes represent their conditional dependence.

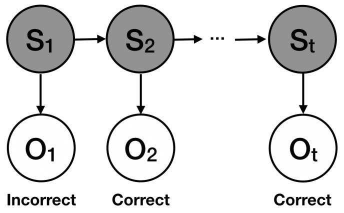


Figure 4: The Bayesian network topology of the standard Knowledge Tracing model

Fundamentally, the BKT model is a two-state Hidden Markov Model [11, HMM] characterized by five basic elements: 1) **N**, the number of different types of hidden state; 2) **M**, the number of different types of observation; 3) $\mathbf{\Pi}$, the initial state distribution $P(S_0)$; 4) **T**, the state transition probability $P(S_{t+1}|S_t)$ and 5) **E**, the emission probability $P(O_t|S_t)$. Note that both **N** and **M** are predefined before training occurs, while $\mathbf{\Pi}$, **T** and **E** are learned from the students' observation sequence.

Conventional BKT assumes there are two types of hidden knowledge states (**N**=2) corresponding to student knowledge states of *unlearned* and *learned*. It also assumes there are two types of student observation (**M**=2) corresponding to student performance of *incorrect* and *correct*. BKT makes two assumptions about its conditional dependence as reflected in the edges in Figure 4. The first assumption BKT makes is a student's knowledge state at a time t is only contingent on her knowledge state at time $t-1$. The second assumption is a student's performance at time t is only dependent on her current knowledge state. These two assumptions are captured by the state transition probability **T** and the emission probability **E**.In the context of student learning, BKT further defines five parameters:

Prior Knowledge = P(S_0=learned)
Learning Rate = P(learned|unlearned)
Forget = P(unlearned | learned)
Guess = P(correct | unlearned)
Slip = P(incorrect | learned)

In order to apply BKT to our dataset, we captured and mapped all students' actions based on the learning opportunities of knowledge components (KCs) step by step. For each of the KC, the Baum-Welch algorithm (or EM method) is used to iteratively update the model's parameters until a maximized probability of observing the training sequence is achieved.

3. EXPERIMENTS

In this work, we explored different student modeling tasks based on characteristics of two different learning environments. One was the task of early prediction of student success in an open-ended, self-paced programming environment while the other is the task of early prediction of student learning gains within a tutor-paced probability tutor.

3.1 Predicting Student Success on iSnap

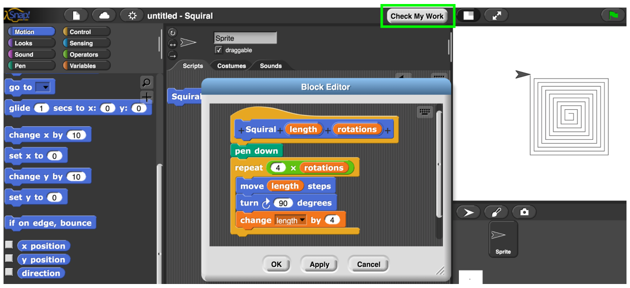
3.1.1 iSnap

iSnap[1] is an extension to Snap*!* [15], a block-based programming environment, used in an introductory computing course for non-majors in a public university in the United States [37]. iSnap extends Snap*!* by providing students with data-driven hints derived from historical correct student solutions [36]. In addition, iSnap logs all students actions while programming (e.g. adding or deleting a block), as a *trace*, allowing us to detect the sequences of all student steps, as well as the *time* taken for each step. In this work, we focused on one homework exercise named Squiral, derived from the BJC curriculum [15]. In Squiral, students are asked to write a procedure that draws a square-like spiral. As shown in Figure 5, correct solutions require procedures, loops, and variables using at least 7 lines of code. We collected students' data for Squiral from Spring 2016, Fall 2016, Spring 2017, and Fall 2017. We excluded students who requested hints from iSnap to eliminate factors that might affect students' problem-solving progress, leaving a total of 65, 38, 29,

[1] All tutors and assignments names have been blinded for anonymous review

and 39 student code traces from each semester, respectively. The detailed statistics for iSnap dataset are shown in Table 1.

The data collected from iSnap consists of a code trace for each student's attempt. This code trace represents a sequence of timestamped snapshots of student code. We used an expert feature detector (EFD), described in [49], that automatically detects 7 features of a correct solution in a student snapshot. For example, for each snapshot in a student code trace, the EFD outputs a *feature state*, which is a series of 0s and 1s (e.g. 10000001) indicating the absence or presence of each feature, such that *feature-state: 1000001* shows that feature 1 and feature 7 are present, while the other 5 features are not. We ran the expert-feature detector to tag each snapshot in all 171 code traces, making a total of 31,064 tagged snapshots.



```
+Squiral+ length + rotations +
pen down
repeat 4 x rotations
  move length steps
  turn ↻ 90 degrees
  change length by 4
```

Figure 5: The iSnap interface, with the blocks palette on the left, the output stage on the right, the scripting area in the middle, and the hints button on top.

3.1.2 Student Success

In the context of iSnap, all the models were measured on the task of predicting student success. We classify the students who finished the programming assignment in one hour or less and got full credit as *successful* and labeled with "0", those who either failed to complete or submit the assignment within one hour as *unsuccessful*, labeled with "1". The one-hour cutoff was chosen based on a distribution showing that the vast majority of students (around 94%) who complete the assignment with full credit do so within one hour. 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 the 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. Note that this is a homework assignment that counts for only a small portion of a student's overall grade, and this behavior (of not attempting to obtain full credit) is typical in this introductory level.

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 remainder 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 their entire sequence *except* the last one.

3.1.3 Four Models

In the task of early prediction of student success, we have four models involved: Logistic Regression (LR), RTP, LSTM and T-LSTM. Note that BKT is not included here because for the open-ended domain like iSnap, there are no predefined steps or knowledge components that students must achieve to complete a given program. Thus, it is hard to map student actions on iSnap to learning opportunities defined in BKT.

Logistic Regression (LR): Since LR 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. 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 6 minute, we apply the last value before 6 minutes for each sequence and treat them as inputs for LR.

RTP: For the RTP-based model, we first used RTP mining to generate the binary matrix and then applied LR to learn from the generated binary matrix. For early prediction, we only apply the *truncated* training sequences included in observation window to find RTPs. For example, for our 6-minute observation window, only the first 6 minutes of sequences were used for pattern extraction.

LSTM and T-LSTM: For LSTM the input is a multivariate temporal sequence from student work, and the output from the last step is used to make a prediction. While for T-LSTM, we also feed it with another sequence indicating time intervals for each student. As shown in Table 1, the time intervals of iSnap range from 1 to 291 seconds across four semesters, with $\mu = 0.613$ and $\sigma = 0.217$ for the overall decayed intervals. For both LSTM and T-LSTM, we used one hidden layer with 128 hidden neurons and set the maximum length to accommodate the longest sequence in our data. Typically for deep learning models, 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.

3.2 Predicting Learning Gains on Pyrenees

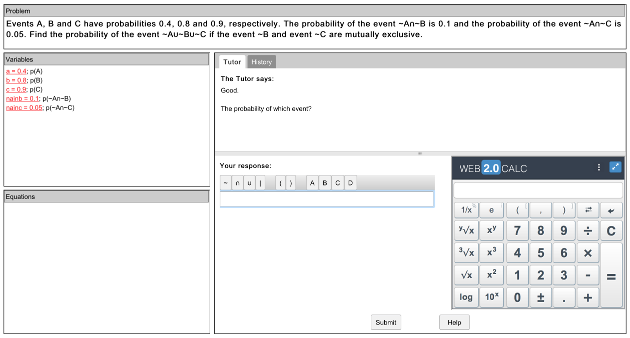
3.2.1 Pyrenees

Pyrenees is a web-based ITS teaching probability, which covers 10 major knowledge components (KCs), such as the Addition Theorem, the Complement Theorem, and Bayes' Rule, etc. Domain experts both identified the 10 KCs and labeled each step/exercise with the corresponding KCs, kappa > 0.9. Figure 6 shows the interface of Pyrenees which consists of a problem statement window, a variable window, an equation window, and a tutor-student dialogue window. Through the dialogue window, Pyrenees provides messages to the students. It can explain a worked example or prompt

Table 1: Detailed data statistics for iSnap, including total steps, total time spent in minutes, time intervals in seconds, corresponding decayed time intervals, and the success labels distribution for each of the four semesters.

Semester	Total Steps				Total Time (minutes)				Time Intervals (seconds)				Decayed Time Intervals	Success Labels	
	min	max	median	mean(std)	min	max	median	mean(std)	min	max	median	mean (std)	mean(std)	S	U
S16	10	1024	169	199 (175)	0.533	95.667	20.733	22.777 (17.149)	1	209	2	6.739 (13.75)	0.628 (0.217)	23	42
F16	28	884	121	167 (168)	3.283	119.083	16.325	22.379 (24.177)	1	189	3	7.919 (14.12)	0.594 (0.217)	15	23
S17	15	439	75	112 (94)	2.817	62.983	14.167	16.347 (11.872)	1	177	3	8.512 (16.14)	0.599 (0.225)	12	17
F17	10	2276	100	219 (376)	1.65	189.667	19.1	28.224 (33.869)	1	291	3	7.597 (15.61)	0.609 (0.215)	9	30

the student to complete the next step. Students can enter their inputs in the text area. Any variable or equation that is defined through this process is displayed on the left side of the screen for reference. Pyrenees can also provide on-demand hints. The bottom-out hint tells the student exactly how to solve a problem. Different from iSnap, the Pyrenees tutor provides immediate feedback for correctness/incorrectness whenever an answer is submitted.



Problem

Events A, B and C have probabilities 0.4, 0.8 and 0.9, respectively. The probability of the event ~A∩~B is 0.1 and the probability of the event ~A∩~C is 0.05. Find the probability of the event ~A∪~B∪~C if the event ~B and event ~C are mutually exclusive.

Variables

a = 0.4; p(A)
b = 0.8; p(B)
c = 0.9; p(C)
nainb = 0.1; p(~A∩~B)
nainc = 0.05; p(~A∩~C)

Equations

The Tutor says:

Good.

The probability of which event?

Your response:

Figure 6: The Pyrenees interface, with the problem statement on the top, the variable window in the middle, the equation window at the bottom, and the dialog window on the right.

When training on Pyrenees, students were required to complete 4 phases: 1) pre-training, 2) pretest, 3) training, and 4) post-test. During the pre-training phase, all students studied the domain principles from a probability textbook. The students then took a pretest which contained 10 problems. The textbook was not available. Students were not given feedback on their answers, nor were they allowed to go back to earlier questions. During the training phase, students received the same 12 training problems in the same order on Pyrenees. Each domain concept was applied at least twice. The minimum number of steps needed to solve each training problem ranged from 10 to 50. The number of domain principles required to solve each problem ranged from 3 to 10. Finally, all of the students took a post-test with 20 problems. Both pretests and post-tests were graded in a double-blind manner by a single experienced grader (not the authors), and were normalized in the range of [0,1]. We collected six semesters of data from Pyrenees, including Fall 2016, Spring 2017, Fall 2017, Spring 2018, Fall 2018, and Spring 2019. The overall dataset comprises 102,948 data points from 1190 students, with 207, 159, 215, 161, 261 and 187 from each semester, respectively. The detailed statistics for Pyrenees dataset are shown in Table 2.

3.2.2 *Quantized Learning Gain*

In the context of Pyrenees, we applied all the models for student learning gains prediction. The concept of *learning gain* is formally defined as the difference between the skills, competencies, content knowledge and personal development demonstrated by students at two points in time [28]. we used a qualitative measurement called *Quantized Learning Gain* [24, QLG] to determine whether a student has benefited from our learning environment. QLG is a *binary* qualitative measurement on students' learning gains from pretest to the posttest: high vs. low. To infer QLGs, students were split into "low", "medium", and "high" based on whether they scored below the 33rd percentile, between the 33rd and 66th percentile, or higher than the 66th percentile in pre-test and post-test respectively. Once a student's pre- and post-test performance groups are decided, the student is a "high" QLG if he/she moved from a lower performance group to a higher performance group from pre-test to post-test or remained in "high" performance groups; whereas a "low" QLG is assigned to the student if he/she either moved from a higher performance group to a lower performance group from pre-test to post-test, or stayed at a "low" or "medium" groups (as shown in Figure 7). In Figure 7, solid lines represented the formation of the *high* QLG groups and dashed lines represents the formation of the *low* QLG groups, and they will be coded with "1" and "0" respectively for QLG prediction. As a result, we have 487 of 1190 students in the *high* learning gain group, and the remaining 703 students in the *low* learning gain group.

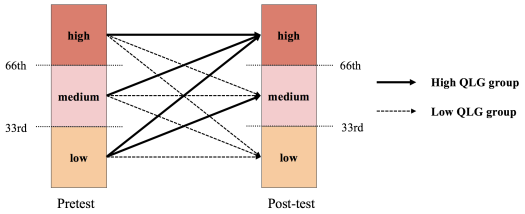


Figure 7: Quantized Learning Gain

Students usually need to spend 2-4 hours to complete the Pyrenees tutor. Thus we are given the first *up to n percentage* of a student's sequence data to predict student QLG, and our goal is to predict whether the student will benefit from our tutoring system in the end. As with the success prediction in iSnap, we left-aligned all the students' trajectories by their starting times and our *observation window* includes the data from the very beginning to the first n percent of the whole sequence.

Table 2: Detailed data statistics for Pyrenees, including total steps, total time spent in hours, time intervals in seconds, corresponding decayed time intervals, and the QLG labels distribution for each of the six semesters.

Semester	**Total Steps**				**Total Time (hours)**				**Time Intervals (seconds)**				**Decayed Time Intervals**	**QLG Labels**	
	min	max	median	mean(std)	min	max	median	mean (std)	min	max	median	mean (std)	mean (std)	*low*	*high*
F16	12	144	78	75 (25)	0.545	173.553	4.142	15.039 (25.56)	1	542136	31	731.799 (10876.82)	0.298 (0.11)	80	127
S17	59	152	88	94 (23)	0.642	240.661	2.643	17.492 (38.29)	1	861636	25	685.094 (14329.39)	0.314 (0.11)	59	100
F17	38	148	113	105 (25)	0.773	576.055	5.100	24.335 (64.29)	1	1287547	24	844.083 (20941.73)	0.313 (0.11)	105	110
S18	23	138	73	71(21)	0.587	135.597	2.682	9.431(18.33)	1	354272	27	486.703 (7021.54)	0.307 (0.10)	47	114
F18	26	162	86	88 (23)	0.679	165.559	4.024	14.914 (22.54)	1	438986	28	613.861 (8924.83)	0.301 (0.10)	98	163
S19	12	138	81	83 (21)	0.571	170.116	4.613	16.909 (27.56)	1	609641	28	738.505 (11439.02)	0.305 (0.11)	98	89

3.2.3 Four Models

In the task of early prediction of student QLG, we have four models involved: LR, BKT, LSTM and T-LSTM. Note that we do not compare RTP here because, in Pyrenees, students' responses are determined not only by their underlying knowledge state, but also by the pre-designed turn-taking nature of the system, which could obscure the temporal patterns found by RTP.

Logistic Regression (LR): As with student success prediction, the "Last Value" approach was applied to the non-temporal LR for the task of predicting student learning gains, as well as the early prediction setting. For example, when the training data is the *first 30% sequence*, only the last value before 30% of each sequence was applied for both training and testing.

BKT: To train the BKT model for QLG prediction, two steps were involved. In the first step, the probability of a student being in the *learned* state on each KC at the last attempt was learned from the BKT model. And in the second step, the output of the first step was computed as features for our prediction tasks. That is, the number of features involved here equals to the total number of KCs involved. The logistic regression was applied to predict QLG. As with early prediction setting of student success, only the *truncated* training sequences were applied to learn student learning probabilities from BKT.

LSTM and T-LSTM: In order to better compare LSTM and T-LSTM performance with BKT, the same two types of features were applied here for QLG prediction: 1) the assignment of KCs corresponding to each step, and 2) student performance at each step, i.e, correct or incorrect. As shown in Table 2, the time intervals of Pyrenees range from 1 second to 14 days across the six semesters, with $\mu = 0.307$ and $\sigma = 0.107$ for overall decayed intervals. For both LSTM and T-LSTM, we used one hidden layer with 64 hidden neurons and also set the maximum length to accommodate the longest sequence in our data. Again, only those events happening within our observation window from each sequence were applied for training and testing of early prediction.

3.3 Evaluation Metrics

Our models in this work were evaluated using Accuracy, Precision, Recall, F1 Score, and AUC (Area Under ROC curve). Accuracy represents the proportion of students whose labels were correctly identified. Precision is the proportion of students who were predicted to be successful by each model who were actually in the *successful* (or *high* QLG) group. Recall tells us what proportion of students, who will actually be unsuccessful (or in *low* QLG group), who 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 tasks, we mainly use Accuracy and AUC to compare different models. Finally, it is important to emphasize that all models were evaluated using semester-based temporal cross-validation for both tasks, which just applied data from previous semesters for training and is a much stricter approach for time series data than the standard cross-validation.

4. RESULTS

4.1 Predicting Student Success in iSnap

Table 3 shows the performance of all models using the first-6-minute training sequences to predict students' success in the programming task. The first column indicates the models including majority baseline model using simple Majority vote, Logistic Regression (LR), RTP, LSTM and T-LSTM. Columns 2-5 report all of the models' performance for the first-6-minute observation window. We evaluated the models on different metrics including Accuracy, Precision, Recall, F1 and AUC score; note that we ignored the Precision, Recall and F1-measure of the simple Majority baseline. The last column reports the mean AUC score of all models from 0 - 20 minutes, with standard deviations between brackets. At first-6-minute, we can observe that T-LSTM outperforms all the other models and it contributes the highest score on every measurement except that the best Recall comes from RTP. LSTM and RTP have very similar performance at first-6-minute, and both of them get better performance than LR except on Precision and AUC. On the other hand, when comparing the overall AUC score among all the models, T-LSTM still achieves the highest score. These results suggests T-LSTM can better learn the difference between successful/unsuccessful groups with the help of time-awareness.

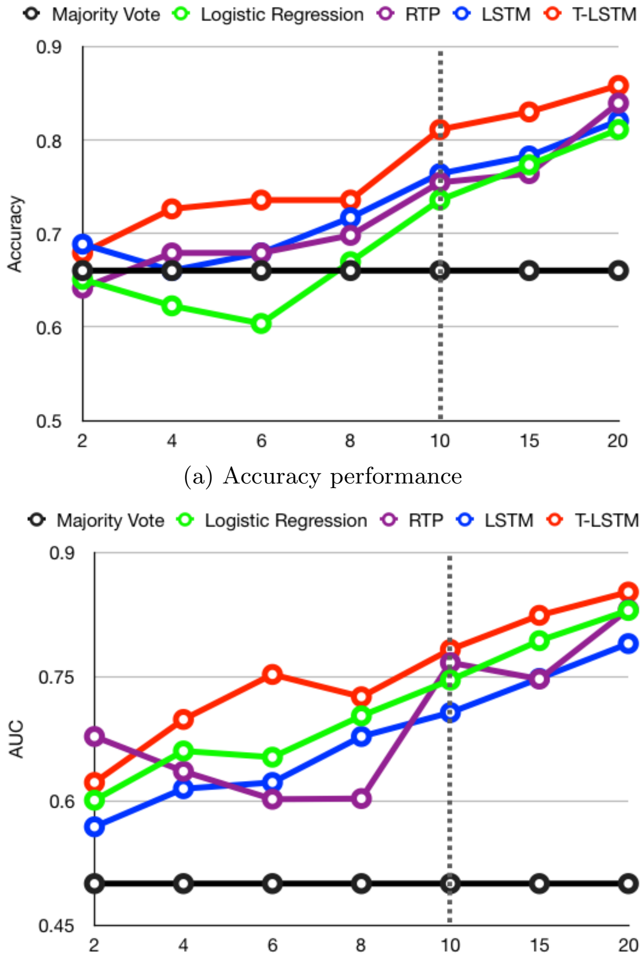
Figures 8 (a) and (b) report Accuracy and AUC performance respectively for all models predicting student success. For each graph, we vary the observation window from the first 2 minutes up to 20 minutes. As shown in Table 1, students generally take 10 to 60 minutes to complete the task and thus we took a measurement every 2 minutes for the first 10 minutes to generate the early stage predictions for each model. T-LSTM is in red, LSTM in blue, RTP in purple, LR in green, and majority baseline in black. Both Figures 8 (a) and (b) show that T-LSTM was the best model for student programming success prediction as it stays on the top across all sizes of the observation window. It is not surpris-

Table 3: iSnap Student Success Prediction at First-6-minute and Overall Time (0 - 20 minutes)

Models	**first-6-minute**					**Overall**
	Accuracy	Precision	Recall	F1-measure	AUC	AUC
Majority	0.6604	-	-	-	0.5000	0.5000
LR	0.6038	0.8333	0.5000	0.6250	0.6528	0.7123(±0.08)
RTP	0.6792	0.7195	**0.8429**	0.7763	0.6020	0.6948(±0.09)
LSTM	0.6792	0.7368	0.8000	0.7671	0.6222	0.6755(±0.09)
T-LSTM	**0.7358**	**0.875**	0.7000	**0.7778**	**0.7528**	**0.7512(±0.07)**

Note: best model on each metric in **bold**

ing that generally for all the models (except majority baseline), the longer the observation windows, the better performance. This is because the training data includes more and more information and students get closer to their final state. The fact that the best prediction comes from T-LSTM really suggests that during the self-paced programming task, taking time-awareness into consideration brings us closer to the truth of the student learning process, especially for the early stage (first 10 minutes). However, this is only one observation from one programming task and more research is needed to understand the full nature of the benefits of time-awareness.



(a) Accuracy performance

(b) Area under ROC performance

Figure 8: Student Success Early Prediction on *iSnap*

4.2 Predicting Learning Gains in Pyrenees

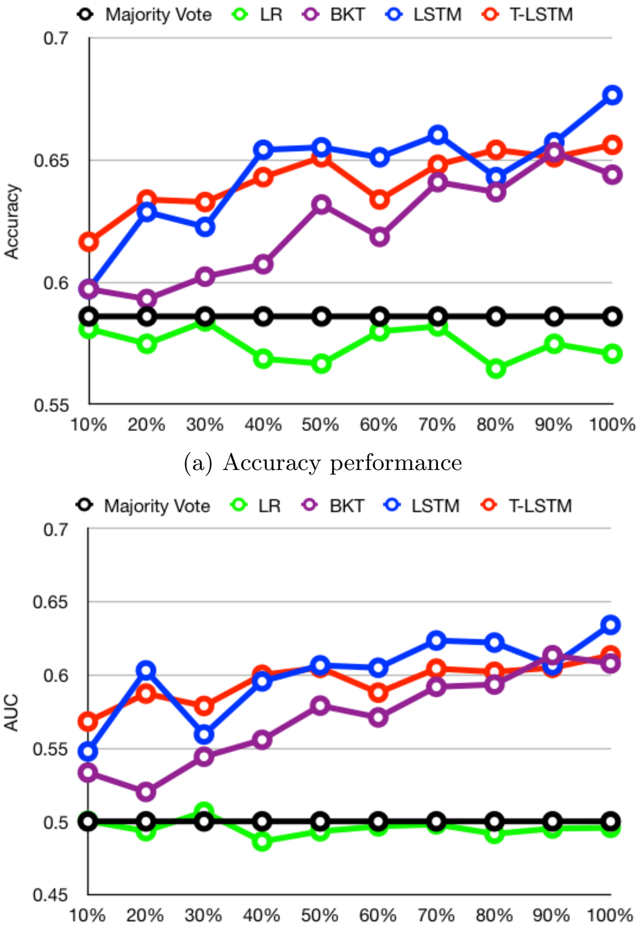
Table 4 shows the performance of all models using the first-30%-sequence to predict students' QLG on the probability tutor. The first column indicates the models including majority baseline model using simple Majority vote, LR, BKT, LSTM, and T-LSTM. Columns 2-5 report the all of the models' performance at the first-30%-sequence observation window. As with Table 3, we evaluated the models on Accuracy, Precision, Recall, F1 and AUC score and ignored the Precision, Recall and F1-measure for the simple Majority baseline. The last column reports the mean AUC score of all models from 0 - 100% sequence, with standard deviations between brackets. When only applying the first-30%-sequence, T-LSTM generates the best performance on every measurement except Recall and F1, where the best Recall is from LR and best F1 from LSTM. Comparing the two deep learning models with classic BKT, we can observe that both LSTM and T-LSTM outperform BKT across all metrics. For the overall AUC performance, LSTM and T-LSTM have very similar scores and are equally good. And still, they achieve higher mean AUC scores than BKT, with a lower standard deviation. Despite the similar overall performance from the two deep learning models, the better early prediction of T-LSTM suggests that time-awareness can help to understand student learning states earlier.

The early prediction results for student learning gains in probability are reported in Figure 9. BKT is in purple, and as in Figure 8, T-LSTM, LSTM, and LR are in red, blue and green, respectively. For each graph, the results are measured at every 10% increment of the sequence length. Generally speaking, the three models (BKT, LSTM and T-LSTM) generate better results as the sequence length increases. Both Figures 9 (a) and (b) show that the two deep learning models outperform BKT for probability, no matter on Accuracy or AUC score. While between LSTM and T-LSTM, there is not a clear winner. Sometimes T-LSTM gets better per-

Table 4: Pyrenees Student QLG Prediction at First-30%-minutes and Overall Time (0 - 100%)

Models	**first-30%-sequence**					**Overall**
	Accuracy	Precision	Recall	F1-measure	AUC	AUC
Majority	0.5860	-	-	-	0.5000	0.5000
LR	0.5839	0.5893	**0.9566**	0.7293	0.5066	0.4957(±0.01)
BKT	0.6022	0.6113	0.8819	0.7221	0.5442	0.5690 (±0.03)
LSTM	0.6226	0.6188	0.9271	**0.7422**	0.5594	**0.6013 (±0.02)**
T-LSTM	**0.6328**	**0.6322**	0.8924	0.7401	**0.5789**	0.5950 (±0.02)

Note: best model on each metric in **bold**



(a) Accuracy performance

(b) Area under ROC performance

Figure 9: Student Learning Gain Early Prediction on *Pyrenees*

formance on Accuracy (from 10% to 30%) while sometimes LSTM slightly outperforms T-LSTM (from 40% to 70%). Overall, LSTM and T-LSTM generate very similar results on predicting student QLG; and T-LSTM generally has better performance on the very early stage.

5. RELATED WORK

Student modeling has been widely and extensively explored in previous research. For example, prior research has proposed a series of approaches based on logistic regression including Item Response Theory (IRT) [42], Learning Factor Analysis [5], Learning Decomposition [4], Instructional Factors Analysis [7], Performance Factors Analysis [33], and Recent-Performance Factors Analysis [14]. These models were implemented with different parameters to better understand and model student learning and were shown to be very successful.

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. Many extensions of BKT have been proposed to capture the complex and diverse aspects of student learning. Pardos and Heffernan [31] explored individualized prior knowledge parameters based on students' overall competence. Their results showed that the proposed model outperformed conventional BKT in predicting students' responses to the last question at the end of the entire training. They later introduced problem difficulty to BKT and found substantial performance improvement in predicting student step-by-step responses over BKT [32]. Additionally, Yudelson et al. [48] parameterized student learning rates in BKT models and the results showed that the new model outperformed conventional BKT in predicting whether the students' next responses were going to be correct/incorrect. Baker et al.[1] investigated contextualized guess and slip rates to deal with the issues of identifiability and model degeneracy commonly observed in conventional BKT. Their results suggested that the proposed models achieved better performance in predicting students' next-step response than BKT. 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 Networks (RNN) or RNN-based models such as LSTM. These deep recurrent models have shown great success in many domains such as speech recognition [17], language translation [26], video classification [29], and rainfall intensity prediction [46], etc. Their success in all these domains has opened up a new line of research in educational data mining [35, 41, 22, 45, 47, 24, 30]. Mao et al. [27] have shown that LSTM has superior performance on the early prediction of student learning gains compared with classic BKT-based models. For the task of predicting students' responses to exercises, LSTM was shown to outperform conventional BKT [35] and Performance Factors Analysis [33]. However, RNN and LSTM did not always have better performance when the simple, conventional models incorporated other parameters. For example, Khajah et al. [22] investigated what statistical regularities neural networks can exploit that BKT cannot, and showed that BKT with relaxed assumptions can outperform LSTM. Wilson et al. [45] also show that Bayesian extensions of simple IRT-based models are also equal to or outperform RNN-based models on a variety of datasets.

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 [13, 39]. In this work, we explored the early prediction of student success and learning gains for a computer-based programming system and an intelligent tutoring system, respectively.

6. CONCLUSIONS

Early prediction of student learning state is a crucial component of student modeling, since it allows tutoring systems to intervene by providing needed support, such as a hint, or by alerting an instructor. Both prediction tasks involved in this work are challenging because: 1) the open-ended nature of iSnap hinders the prediction of student final success, and 2) it is extremely hard to track whether a student benefits from a tutoring system or not even in a well-defined domain like Pyrenees. In this work, we investigated the effectiveness of a time-aware model, T-LSTM on the two different prediction tasks and compared it with other student modeling methods including LSTM, RTP, logistic regression models, and BKT. Our results show that T-LSTM consistently outperforms the other models such as LSTM, RTP, and non-temporal logistic regression on the task of predicting student success in iSnap, at all observation windows from first 2 minutes to 20 minutes. On the other hand, for the task of predicting student learning gains in Pyrenees, T-LSTM does not outperform the other models. More specifically, T-LSTM outperforms LSTM and BKT on the early stage with only 30% of the student sequences, and afterward time-awareness does not help much when more data is available. One possible explanation behind this is that in a well-defined domain, the whole learning process is mainly driven by the tutor, which makes the elapsed time less important to student learning gains especially when the step-level performance is available. However, in the open-ended programming environment, students are self-prompted to complete an assignment; and therefore the amount of time they stayed in a state really matters to understand their learning. And therefore, T-LSTM can generate better performance by modeling the student dynamics of knowledge *in continuous time* than other methods in *discrete timesteps*.

One limitation of this work is that we only explored one important student modeling task in each learning environment. An important direction for future work is to investigate the time-aware model on other student modeling tasks in both learning environments to determine whether the same results will hold. In addition, we are planning to employ the time-awareness to other models such as RTP to explore whether it continues to support improvement for the open-ended programming environment. 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.

Acknowledgements: This research was supported by the NSF Grants: EXP: Data-Driven Support for Novice Programmers (1623470), Generalizing Data-Driven Technologies to Improve Individualized STEM Instruction by Intelligent Tutors (2013502), Integrated Data-driven Technologies for Individualized Instruction in STEM Learning Environments(1726550), and CAREER: Improving Adaptive Decision Making in Interactive Learning Environments(1651909).

7. REFERENCES

[1] 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.

[2] 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.

[3] I. M. Baytas, C. Xiao, X. Zhang, F. Wang, A. K. Jain, and J. Zhou. Paient subtyping via time-aware lstm networks. In *SIGKDD*. ACM, 2017.

[4] J. E. Beck and J. Mostow. How who should practice: Using learning decomposition to evaluate the efficacy of different types of practice for different types of students. In B. P. Woolf, E. Aïmeur, R. Nkambou, and S. Lajoie, editors, *Intelligent Tutoring Systems*. Springer, 2008.

[5] H. Cen, K. Koedinger, and B. Junker. Learning factors analysis – a general method for cognitive model evaluation and improvement. In M. Ikeda, K. D. Ashley, and T.-W. Chan, editors, *Intelligent Tutoring Systems*, pages 164–175. Springer, 2006.

[6] C. Che, C. Xiao, J. Liang, B. Jin, J. Zho, and F. Wang. An rnn architecture with dynamic temporal matching for personalized predictions of parkinson's disease. In *SDM*. SIAM, 2017.

[7] 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.

[8] E. Choi, M. T. Bahadori, A. Schuetz, W. F. Stewart, and J. Sun. Doctor ai: Predicting clinical events via recurrent neural networks. In *MLHC*, pages 301–318, 2016.

[9] M. Cocea and S. Weibelzahl. Can log files analysis estimate learners' level of motivation? In *LWA*, pages

32–35. University of Hildesheim, Institute of Computer Science, 2006.

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

[11] S. R. Eddy. Hidden markov models. *Current opinion in structural biology*, 6(3):361–365, 1996.

[12] C. Esteban, O. Staeck, et al. Predicting clinical events by combining static and dynamic information using recurrent neural networks. In *ICHI*, pages 93–101. IEEE, 2016.

[13] 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.

[14] A. Galyardt and I. Goldin. Recent-performance factors analysis. In J. Stamper, Z. Pardos, M. Mavrikis, and B. McLaren, editors, *Proceedings of the 7th International Conference on Educational Data Mining*, pages 411–412, 2014.

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

[16] W. J. González-Espada and D. W. Bullock. Innovative applications of classroom response systems: Investigating students' item response times in relation to final course grade, gender, general point average, and high school act scores. *Electronic Journal for the Integration of Technology in Education*, 6:97–108, 2007.

[17] A. Graves, N. Jaitly, and A.-r. Mohamed. Hybrid speech recognition with deep bidirectional lstm. In *ASRU*, pages 273–278. IEEE, 2013.

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

[19] X. Jia, A. Khandelwal, G. Nayak, J. Gerber, K. Carlson, P. West, and V. Kumar. Incremental dual-memory lstm in land cover prediction. In *SIGKDD*. ACM, 2017.

[20] X. Jia, S. Li, A. Khandelwal, G. Nayak, A. Karpatne, and V. Kumar. Spatial context-aware networks for mining temporal discriminative period in land cover detection. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, pages 513–521. SIAM, 2019.

[21] E. Joseph. Engagement tracing: using response times to model student disengagement. volume 125, page 88. IOS Press, 2005.

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

[23] F. Khoshnevisan, J. Ivy, M. Capan, R. Arnold, J. Huddleston, and M. Chi. Recent temporal pattern mining for septic shock early prediction. In *2018 IEEE International Conference on Healthcare Informatics (ICHI)*, pages 229–240. IEEE, 2018.

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

[25] Z. C. Lipton, D. C. Kale, et al. Learning to diagnose with lstm recurrent neural networks. *arXiv preprint arXiv:1511.03677*, 2015.

[26] M.-T. Luong and C. D. Manning. Stanford neural machine translation systems for spoken language domains. In *IWSLT*, pages 76–79, 2015.

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

[28] C. H. McGrath, B. Guerin, E. Harte, M. Frearson, and C. Manville. Learning gain in higher education. *Santa Monica, CA: RAND Corporation*, 2015.

[29] J. Y.-H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. In *CVPR*, pages 4694–4702. IEEE, 2015.

[30] S. Pandey and G. Karypis. A self-attentive model for knowledge tracing. *arXiv preprint arXiv:1907.06837*, 2019.

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

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

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

[34] T. Pham, T. Tran, D. Phung, et al. Deepcare: A deep dynamic memory model for predictive medicine. In *PAKDD*. Springer, 2016.

[35] 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.

[36] T. Price, R. Zhi, and T. Barnes. Evaluation of a data-driven feedback algorithm for open-ended programming. *International Educational Data Mining Society*, 2017.

[37] T. W. Price, Y. Dong, and D. Lipovac. iSnap: Towards Intelligent Tutoring in Novice Programming Environments. In *Proceedings of the ACM Technical Symposium on Computer Science Education*, pages 483–488, 2017.

[38] Y. Qiu, Y. Qi, H. Lu, Z. A. Pardos, and N. T. Heffernan. Does time matter? modeling the effect of time with bayesian knowledge tracing. In *EDM*, pages 139–148, 2011.

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

[40] D. L. Schnipke and D. J. Scrams. Exploring issues of examinee behavior: Insights gained from response-time analyses. In C. N. Mills, M. T. Potenza, J. J. Fremer, and W. C. Ward, editors, *Computer-based testing: Building the foundation for future assessments*, pages 237–266, Mahwah, NJ, US, 2002. Lawrence Erlbaum Associates Publishers.

[41] 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.

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

[43] R. D. L. V. S. Thomas et al. *Response Times: Their Role in Inferring Elementary Mental Organization: Their Role in Inferring Elementary Mental Organization*. Oxford University Press, USA, 1986.

[44] Y. Wang and N. T. Heffernan. Leveraging first response time into the knowledge tracing model. *International Educational Data Mining Society*, 2012.

[45] 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.

[46] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *NIPS*, pages 802–810, 2015.

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

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

[49] 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.