Learning Individual Behavior in an Educational Game: A Data-Driven Approach

Seong Jae Lee, Yun-En Liu, and Zoran Popović
Certer for Game Science, Computer Science and Engineering
University of Washington
{seongjae, yunliu, zoran}@cs.washington.edu

ABSTRACT

In recent years, open-ended interactive educational tools such as games have been gained popularity due to their ability to make learning more enjoyable and engaging. Modeling and predicting individual behavior in such interactive environments is crucial to better understand the learning process and improve the tools in the future. A model-based approach is a standard way to learn student behavior in highly-structured systems such as intelligent tutors. However, defining such a model relies on expert domain knowledge. The same approach is often extremely difficult in educational games because open-ended nature of these systems creates an enormous space of actions. To ease this burden, we propose a data-driven approach to learn individual behavior given a user's interaction history. This model does not heavily rely on expert domain knowledge. We use our framework to predict player movements in two educational puzzle games, demonstrating that our behavior model performs significantly better than a baseline on both games. This indicates that our framework can generalize without requiring extensive expert knowledge specific to each domain. Finally, we show that the learned model can give new insights into understanding player behavior.

Keywords

Educational Games, Data-driven Learning, Supervised Learning, Logistic Regression, Learning from Demonstration

1. INTRODUCTION

Open-ended educational environments, especially educational games and game-based learning, have been gaining popularity as more and more evidence suggests that they can help enhance student learning while making the process enjoyable [21, 19]. The interactive nature of this media enables us to track user inputs and provide an immediate personalized response to make learning more effective and efficient. Therefore, modeling and predicting user behavior lies at the heart of improving engagement and mastery for every learner. For example, we can detect if a student is struggling with a certain concept by simulating the learned student behavior model on a related task; if the student model performs poorly, we can then give more tasks related to that concept. The ability to further predict which error the student is going to make can be used to infer how the learner misunderstood related concepts and provide a targeted instruction focusing on that specific misconception.

There has been active research on learning individual behavior in both the education and game community. Most of this research focuses on inferring a meaningful latent structure, such as knowledge or skill, often simplifying the behavior space into a small number of parameters. Hence the choice of a model significantly affects the quality of the learned behavior, forcing researchers to spend time on experimenting with different models and refining various features [6, 20]. Moreover, unlike highly-structured systems such as intelligent tutors, it is difficult to define a behavioral model describing movements in an educational game. This is especially true when the player is given a large number of available moves, resulting in a large scale multi-class prediction problem. For example, predicting the exact moves someone will make while solving a puzzle is more challenging than predicting whether a student will solve a problem correctly.

Therefore, a purely data-driven approach is both a suitable and preferable alternative for learning user behavior in highly open-ended environment, such as educational games. It needs less expert authoring, and it can capture various low-level user movements—mistakes and errors, exploration habits, and adapting a strategy while playing—as they are even without a specific model describing such moves. Also, we can further analyze learned policies to give more interpretable insights into user behavior. For example, we could analyze erroneous movements to figure out their misconceptions. This knowledge can be used to construct more sophisticated cognitive models that will give us a deeper and more accurate understanding of user behavior.

In this paper, we propose a data-driven framework that learns individual movements in a sequential decision-making process. It uses a supervised classification method to predict the next movement of a user based on past gameplay data. For each game state, our framework transforms user play logs into a high-dimensional feature vector, and learns a classifier that predicts the next movement based on this feature vector. To construct this set of features, we start

from a massive set of default features defined in a domain-independent way over a state-action graph, and then pick a small set of relevant ones using a univariate feature selection technique. We apply this framework to two very different educational games DragonBox Adaptive and Refraction, and evaluate the learned behavior by predicting the actions of held-out users.

Our contribution is threefold. First, we propose a data-driven individual behavior learning framework, which does not rely on heavy domain-dependent expert authoring. Second, we apply our framework on two different games and demonstrate our framework improves the prediction quality substantially over a previously proposed algorithm. Finally, combined with a robust feature selection, we show our framework learns an efficient yet powerful set of features, which further gives new insight into understanding player behavior in the game.

2. RELATED WORK

Learning user behavior has been actively researched in the education and game community. One of the most widely used models is the Bayesian network model. Knowledge tracing (KT) [9] and its variations [17, 27] are probably the most widely used student models in the field of educational data mining. This model estimates the user's knowledge as latent variables, or knowledge components (KCs). These KCs represent student mastery over each concept and predict whether a student will be able to solve a task. There are also other approaches using Bayesian networks, such as predicting whether a student can solve a problem correctly without requesting help in an interactive learning environment [16], predicting a user's next action and goal in a multi-user dungeon adventure game [5], or predicting a build tree in a real-time strategy game [22]. Nevertheless, these approaches usually learn the knowledge of users directly from carefully designed network structures, which often need expert authoring to define a task-specific and system-specific network model.

Another widely used approach to learn individual behavior is using factor analysis. Item response theory (IRT) models have been extensively used in psychometrics and educational testing domains [11, 8]. They represent an individual's score as a function of two latent factors: individual skill and item difficulty. The learned factors can be used to predict user performance on different items. Adapting matrix factorization techniques from recommender systems is also popular. Thai-Nghe et al. predicted personalized student performance in algebra problems by modeling user-item interactions as inner products of user and item feature vectors [24]. This model can even incorporate temporal behavior by including time factors, both in the educational community [25] and in the game community [28]. Nevertheless, such approaches do not easily fit our goal, which requires predicting a fine granularity of action instead of predicting user's single valued performance.

Another line of research on learning a low-level user policy is optimizing a hidden reward or heuristic function from user trajectories in a state-action graph. Tastan and Sukthankar built a human-like opponent from experts' demonstrations in a first-person shooter game using inverse reinforcement learning in a Markov decision process context [23]. Jansen et al. learned a personalized chess strategy from individual play records by learning the weights of multiple heuristics on finite depth search models [12]. Similarly, Liu et al. learned player moves in an educational puzzle game by learning the weights of heuristics in a one-depth probabilistic search model [15]. However, such methods usually define a user reward as a combination of pre-defined heuristics. The quality of the learned policy is strongly dependent on these heuristics, which are often system-specific and need a lot of time to refine. Furthermore, these models assume that players do not change over time. Describing how people learn, which is frequently observed in interactive environments, would be another challenge to trying this approach in our domain.

Unlike the research listed above, our framework focuses on a data-driven approach with less system-specific authoring. Our work is a partial extension of that of Liu et al. [15], which is a mixture of a one-depth heuristic search model and a data-driven Markov model with no knowledge of individual history other than the current game state. We focus on the latter data-driven approach and build upon that model. Unlike their work, which makes the same prediction for all players in a state, we build a personalized policy that considers the full history of the user.

3. PROBLEM DEFINITION

Our framework works on a model that consists of a finite set of states S; a finite set of actions A, representing the different ways a user can interact with the game; and rules of transitioning between states based on an action. This paper considers domains with determinstic transitions ($f : S \times A \to S$), but this approach could also apply to domains with stochastic transitions ($f : S \times A \to \Pr(S)$). The demonstration of a user u, or a trajectory τ_u, is defined as a sequence of state-action pairs: $\tau_u = \{(s_u^0, a_u^0), \cdots, (s_u^{t_u}, a_u^{t_u})\}$. We will note $A_s \subseteq A$ as a set of valid actions on the given state, T as a set of trajectories, V and U as the set of users in the training set and test set, respectively.

Defining such a model is often intuitive in games, because actions and transition rules are already defined in game mechanics. For example, in blackjack, the configurations of the visible cards can be states, and available player decisions such as hit or stand will be actions. The transition function here will be stochastic, because we do not know which card will appear next. Defining a good state space remains an open problem. A good, compact state representation will capture the information most relevent to user behavior; this helps greatly reduce the size of the required training data. For the example above, using the sum of card scores for each side would be a better state representation than the individual cards on the table. Note that a state refers to the observable state of the game, not the state of the player. In many games, players base their decisions not just on what they currently see in front of them, but also on past experience. This non-Markovian property of human players motivates our framework, which leverages a user's past behavior to improve prediction quality.

Our objective is to learn a stochastic policy $\pi : S \times T \to \Pr(A_s)$ describing what action a user will take on a certain

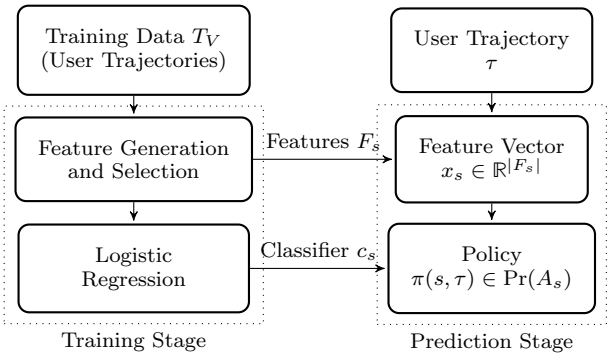

Figure 1: An overview of our framework. For each state, we learn a set of features and a classifier in the training stage. With these, a trajectory is converted into a feature vector, and further into a stochastic policy in the prediction stage.

state based on his trajectory. We use a supervised learning approach, which attempts to predict the next action given a dataset of thousands of user trajectories in the training set $T_V = \{\tau_v | v \in V\}$.

4. EVALUATION

We evaluate the learned policy on every movement of each user trajectories in the test set $T_U = \{\tau_u | u \in U\}$. We use two evaluation metrics: log-likelihood and accuracy rate.

The log-likelihood is defined as

$$\sum_{u \in U} \sum_{t \in [0, t_u]} \log(\tilde{\pi}(a_u^t | s_u^t, \tau_u^t)),$$

while $\tilde{\pi}$ is the learned policy given a trajectory observed so far $\tau_u^t = \{(s_u^0, a_u^0), \cdots, (s_u^{t-1}, a_u^{t-1})\}$. Since the log function is undefined for the case $\pi(a|s, \tau) = 0$, we smooth the policy by ϵ: $\pi(a|s, \tau) = (1 - \epsilon)\pi(a|s, \tau) + \epsilon/|A_s|$, while ϵ is set to 0.001 in our experiments unless otherwise specified. The log-likelihood is always smaller than or equal to zero, with 0 indicating perfect prediction.

With our stochastic policy, the accuracy rate is defined as

$$\frac{\sum_{u \in U} \sum_{t \in [0, t_u]} \tilde{\pi}(a_u^t | s_u^t, \tau_u^t)}{\sum_{u \in U} \sum_{t \in [0, t_u]} 1}.$$

We want to note that even though the accuracy rate is intuitive and widely used for measuring the performance of a classifier, using it as a single evaluation metric can be misleading, especially when outputs are highly skewed [13]. For example, a degenerate constant classifier that predicts only the dominant class may produce a high accuracy rate. Nevertheless, in all of our experiments, the ordering of performances in log-likelihoods is preserved with that in accuracy rates.

5. ALGORITHM

Figure 1 provides an overview of our data-driven framework. In the training stage, our framework learns a set of features and a classifier from training data. A simple way to do this might be to train a global classifier that takes a player trajectory and predicts an action. However, we suspect that the current state is the most important feature; in fact, the set of available actions A_s differs per state. Thus we train a separate classifier for each state, reducing the amount of data in the training set but increasing the relevancy. In the prediction stage, we take a trajectory and convert it into a feature vector using the learned features. Then we convert the feature vector into a policy from that state, using the learned classifier. Here, a feature is a function that takes a trajectory and returns a value $f : T \to \mathbb{R}$; a set of features converts a trajectory into a multi-dimensional feature vector $F : T \to \mathbb{R}^{|F|}$.

Algorithm 1 Training Stage

Require: a state s, training data T_V
1: $F_s \leftarrow$ a default set of features defined on s
2: $T_s, y_s \leftarrow$ a list of τ_v^{t-1} and a_v^t such that $s = s_v^t \ \forall v, t$
3: $X_s \leftarrow F_s(T_s)$
4: $F_s \leftarrow$ SelectFeatures(F_s, X_s, y_s)
5: **if** $F_s = \emptyset$ **then**
6: $\quad c_s \leftarrow$ LearnMarkovClassifier(y_s)
7: **else**
8: $\quad X_s \leftarrow F_s(T_s)$
9: $\quad c_s \leftarrow$ LearnClassifier(X_s, y_s)
10: **end if**
11: **return** F_s, c_s

Algorithm 1 describes a detailed process in the training stage. The SelectFeatures function takes a set of features, a set of feature vectors, and a set of performed actions as inputs and filters irrelevant features out. The LearnMarkovClassifier function takes a set of performed actions and returns a static classifier. The LearnClassifier function takes a set of feature vectors and performed actions as inputs and returns a classifier. We will explain each function in detail.

The training stage starts from a default set of features defined on a state-action graph. We use three kinds of binary features:

- whether the user has visited a certain state s: $\mathbb{1}[s \in \tau]$,
- whether the trajectory contains a certain state-action pair (s, a): $\mathbb{1}[(s, a) \in \tau]$, and
- whether the dth recent move is a certain state-action pair (s, a): $\mathbb{1}[(s, a) = (s^{|\tau|-d}, a^{|\tau|-d}) \in \tau]$.

The maximum number of d is set to 10 in our experiments. The features with sparsely-visited states and transitions are not counted. These features summarize which states and actions have been visited by the user, both in the full and recent history. We built a feature package defined on an abstract state-action graph so that it can be used generally in multiple systems without extra authoring.

The default set of features contains a huge amount of irrelevant features for the task, making our learning suffer from overfitting as well as prolonged training time. To remedy this, we apply a feature selection method using training data. For the SelectFeatures function, we use a chi-squared statistic for each feature on T_v and select features

State s	Action a	Next State $s' = T(s, a)$	$\pi(a\|s, \tau)$
`lv.3.4.1.x+a=b`	Subtract `a` on both sides with merging	`lv3.4.1.x+0=b-a`	0.4
	Subtract `a` on both sides	`lv3.4.1.x+a-a=b-a`	0.4
	Add `a` on both sides	`lv3.4.1.x+a+a=b+a`	0.2

Table 1: Examples of states, actions, transitions, and policies in DragonBox Adaptive

with p-value lower than 0.001. We used a univariate feature filtering approach because it is relatively fast compared to other feature selection techniques, such as L1-regularization. Also, we used a chi-squared test because it is one of the most effective methods of feature selection for classification [26].

Finally, we learn a classifier with the selected features. If any features are selected, we learn a supervised learning classifier using the LearnClassifier. We used a multi-class logistic regressor as a classifier, because it gives a natural probabilistic interpretation unlike decision trees or support vector machines. If no features are selected, we use a predictor built on a Markov model from Liu et al. [15], which learns the observed frequency of state-action pairs in training data:

$$\tilde{\pi}(a|s) = \frac{\sum_{v \in V} \sum_{t \in [0, t_v]} \mathbb{1}[(s, a) = (s_v^t, a_v^t)]}{\sum_{v \in V} \sum_{t \in [0, t_v]} \mathbb{1}[s = s_v^t]}.$$

When there are zero samples, i.e., when the denominator is zero, this equation is undefined and it returns a uniform distribution instead. For convenience, we will call this predictor the Markov predictor.

We can also interpret the Markov predictor as another logistic regressor with no features but only with an intercept for each action. With no regularization or features, logistic regression optimizes the log-likelihood on training data with a policy only dependant on the current state, whose optimal solution is the observed frequency for each class. This is exactly what the Markov predictor does.

In the prediction stage, we take a trajectory and a state as an input, and first check the type of the learned classifier on the given state. If the classifier is the Markov predictor, it does not need a feature vector and returns a policy only dependent on the current state. Otherwise, we use the learned features F_s and classifier c_s to convert a user's trajectory τ into a feature vector x, and then into a state-wise stochastic policy $\pi(s, \tau)$.

6. EXPERIMENT AND RESULT

6.1 DragonBox Adaptive

6.1.1 *Game Description*

DragonBox Adaptive is an educational math puzzle game designed to teach how to solve algebra equations to children ranging from kindergarteners to K-12 students [2], which is evolved from the original game DragonBox [1]. Figure 2 shows the screenshots of the game. Each game level represents an algebra equation to solve. The panel is divided into two sides filled with cards representing numbers and variables. A player can perform algebraic operations by merging cards in the equation (e.g., '`-a+a`' → '`0`'), eliminating identities (e.g. '`0`' → ' '), or using a card in the deck to perform addition, multiplication, or division on the both sides of the

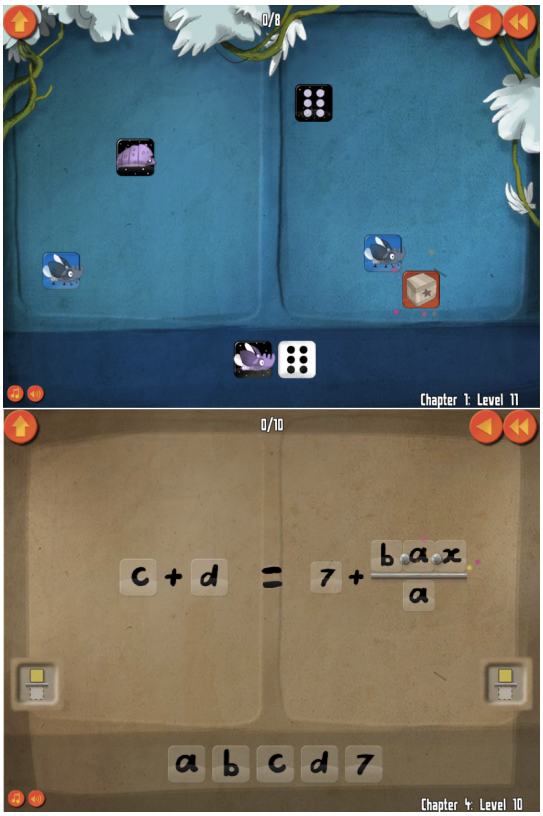

Figure 2: Screenshots of DragonBox Adaptive. (Top) The early stage of the game. It is equivalent to an equation `a-b=-6+a+x`. The card with a starred box on the bottom right is the DragonBox. (Bottom) The game teaches more and more concepts, and eventually kids learn to solve complex equations.

equation. To clear a level, one should eliminate all the cards on one side of the panel except the DragonBox card, which stands for the unknown variable.

Table 1 shows the examples of states, actions, transitions, and policies in the game. Since DragonBox Adaptive is a card puzzle game, the game state and available actions are well discretized. A state is a pair of level ID and the current equation (e.g. `lv3.x-1=0`). Available actions for a specific state are the available movements, or algebraic operations, that the game mechanics allow the players to perform (e.g. add `a` on both sides or subtract `a` on both sides). Performing an action moves a user from one state to another state (e.g. an action adding `a` on both sides moves a user from `lv3.x-a=0` to `lv3.x-a+a=0+a`), which naturally defines a transition function. Since the transition is deterministic and injective, we use a notation $s \to s'$ for a transition from s to $s' =$

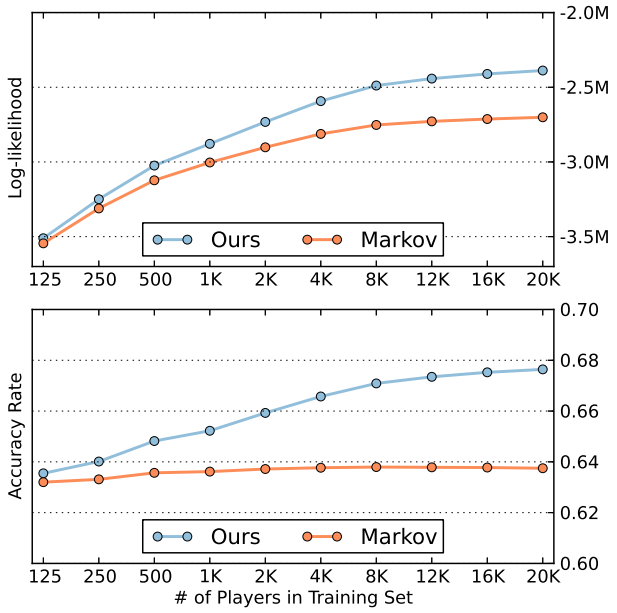

Figure 3: Performances of our predictor and that of the Markov predictor with different sizes of training data.

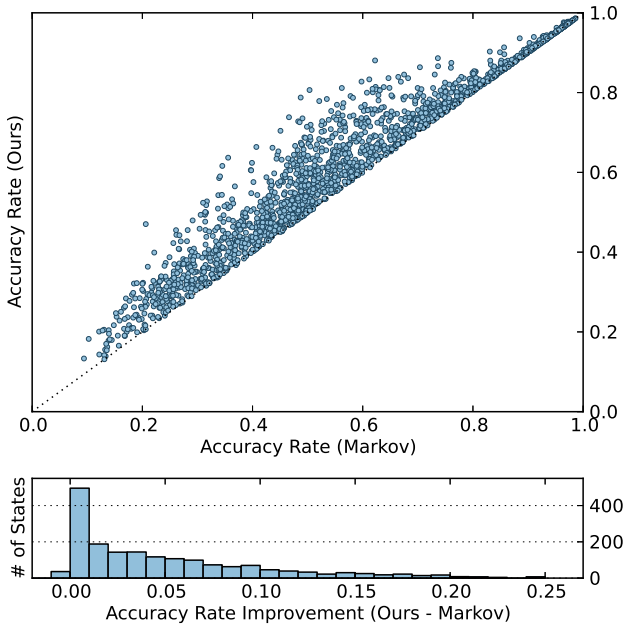

Figure 4: (Top) Scatter plot of accuracy rates of our predictor versus that of the Markov predictor for each state with learned features. We see the performance of our predictor is strictly better than that of the Markov predictor in most cases. (Bottom) Histogram of accuracy rate improvement, while one count is a state with learned features.

$f(s, a)$ (e.g. `lv3.x-a=0` → `lv3.x-a+a=0+a`).

We collected the game logs from the Norway Algebra Challenge [3], a competition solving as many algebra equations as possible in DragonBox Adaptive. About 36,000 K-12 students across the country participated in the competition, which was held on January 2014 for a week. One characteristic of this dataset is a low quit rate, which is achieved because students intensively played the game with classmates to rank up their class. About 65% of the participants played the game more than an hour, and more than 200 equations were solved per student on average. This is important in our task because we can collect a lot of training data even in higher levels. After cleaning, we collected 24,000 students' logs with about 280,000 states, 540,000 transitions, and 21 million moves. 4,000 student's logs were assigned to test data and the rest as training data. As the parameters of our framework were determined with another Algebra Challenge dataset separate from Norway Challenge, there was no learning from the test data.

6.1.2 *Overall Performance*

Figure 3 shows the performances of our framework and those of the Markov predictor with different sizes of training data. We use the Markov predictor as a baseline, because it is another data-driven policy predictor working on a state-action graph structure, and because our model is using the Markov predictor when it could not find relevant features to the given state.

In both metrics, the performance of our predictor increases with the size of training data. As it has not converged yet, we can even expect better performance with additional training data. For the Markov predictor, its performance notably improves with the size of training data only in log-likelihood metric. We believe this is because the accuracy rate mostly comes from frequently visited states, which the predictor already reached to the point with no improvement, while a significant portion of the log-likelihood value comes from sparsely visited states, which improves as it gathers more data.

The difference between the two methods is increasing as the size of training data increases. With 20,000 user trajectories as training data, the log-likelihood of our predictor is almost 12% lower than that of the Markov predictor and the accuracy rate improves from 64% from 68%. Since the Markov predictor gives a static policy, we can say this gain comes from considering individual behavior differences. Also, even with a relatively small size of training data, the performance of our method is still higher than that of the Markov predictor. We believe our predictor performs strictly better than the Markov predictor even with insufficient data, because our method switches to the latter when it does not have enough confidence on selecting features.

6.1.3 *Statewise Performance*

In this subsection, we further analyze the performances of two methods in the smaller scope. Our predictor learned 1,838 logistic regression classifiers with 20,000 player trajectories as training data. Considering there are about 280,000 game states in total, our method selected no features for more than 99% of the states and decided to use the Markov predictor instead. However, more than 60% of the move-

Next State	Feature	Weight
lv.3.4.1.x+0=b-a	$\mathbb{1}[(\texttt{lv.3.5.1.x+a=b} \rightarrow \texttt{lv.3.5.1.x+0=b-a}) \in \tau_u]$	0.583
	$\mathbb{1}[(\texttt{lv.3.4.1.x+a=b} \rightarrow \texttt{lv.3.4.1.x+0=b-a}) \in \tau_u]$	0.568
	$\mathbb{1}[(\texttt{lv.3.4.1.x+a=b} \rightarrow \texttt{lv.3.4.1.x+0=b-a}) = (s_{t-3}, a_{t-3})]$	0.387
	$\mathbb{1}[(\texttt{lv.2.19.x*b+a=c} \rightarrow \texttt{lv.2.19.x*b+0=c-a}) \in \tau_u]$	0.181
lv.3.4.1.x+a-a=b-a	$\mathbb{1}[(\texttt{lv.3.4.1.x+a+a=b-a} \rightarrow \texttt{lv.3.4.1.x+0=b-a}) = (s_{t-3}, a_{t-3})]$	0.466
	$\mathbb{1}[(\texttt{lv.3.4.1.x+a=b} \rightarrow \texttt{lv.3.4.1.x+a-a=b-a}) = (s_{t-4}, a_{t-4})]$	0.457
	$\mathbb{1}[(\texttt{lv.3.3.2.x+a+b=c} \rightarrow \texttt{lv.3.3.2.x+a+b-b=c-b}) = (s_{t-7}, a_{t-7})]$	0.389
	$\mathbb{1}[(\texttt{lv.2.3.x+1=a} \rightarrow \texttt{lv.2.3.x+0=a-1}) \in \tau_u]$	0.119
lv.3.4.1.x+a+a=b+a	$\mathbb{1}[(\texttt{lv.3.4.1.x+a+a+a=b+a+a} \rightarrow \texttt{lv.3.4.1.x+a+a=b+a}) = (s_{t-2}, a_{t-2})]$	0.247
	$\mathbb{1}[(\texttt{lv.3.4.1.x+a=b} \rightarrow \texttt{lv.3.4.1.x+a+a=b+a}) = (s_{t-6}, a_{t-6})]$	0.192
	$\mathbb{1}[(\texttt{lv.1.18.x+a+a=a+b} \rightarrow \texttt{lv.1.18.x+a+0=b+0}) \in \tau]$	0.128
	$\mathbb{1}[(\texttt{lv.2.3.x+(-x)/(-x)+x/x=1+a} \rightarrow \texttt{lv.2.3.x+(-x)/(-x)+x/x-1=a+0}) \in \tau]$	0.115

Table 2: Selected features with high weights for predicting a transition from state '`lv.3.4.1.x+a=b`' to each available action. The selected features closely related to the task it is going to predict. Interesting features mentioned in the text are highlighted.

Transition	Accuracy		Recall	
from `x+a=b`	LogReg	Markov	LogReg	Markov
`x+0=-a+b`	88.0	76.7	88.1	76.7
`x+a-a=-a+b`	60.6	19.2	60.3	19.2
`x+a+a=a+b`	11.5	3.7	11.6	3.8

Table 3: Accuracy and recall rate for each action on state `lv.3.4.1.x+a=b`. The performance of highlighted transitions are almost tripled.

ments in training data starts from the states with corresponding logistic regression classifiers. It means our framework invested its resources on a small set of states, which are so influential that they govern the majority of the prediction tasks.

Figure 4 shows the accuracy rates in both predictors for each state that learned a logistic regression classifier. We can see the performance of our predictor is better than that of the Markov predictor in the most cases. The other case is ignorable: the Markov performs better than ours in less than 2% of the states with logistic regressors, while the average performance drop for those states is 0.03%. This observation further supports our argument that our model performs strictly better than the Markov model because of our robust feature selection process.

Table 3 gives the evaluations on a state that showed an accuracy rate improvement from 63% to 80%. Since we are evaluating stochastic policies, we use the following definition for accuracy and recall for a pair (s, a):

$$\frac{\sum_{u \in U} \sum_{t \in [0, t_u]} \mathbb{1}[(s, a) = (s_v^t, a_v^t)] \cdot \tilde{\pi}(a|s, \tau_u^t)}{\sum_{u \in U} \sum_{t \in [0, t_u]} \mathbb{1}[(s, a) = (s_v^t, a_v^t)]},$$

$$\frac{\sum_{u \in U} \sum_{t \in [0, t_u]} \mathbb{1}[(s, a) = (s_v^t, a_v^t)] \cdot \tilde{\pi}(a|s, \tau_u^t)}{\sum_{u \in U} \sum_{t \in [0, t_u]} \mathbb{1}[s = s_v^t] \cdot \tilde{\pi}_u^t(a|s, \tau_u^t)}.$$

In the table, the starting equation is `x+a=b`, and a player can perform addition or subtraction with a card '`a`' on the deck. There are three possible transitions because the game mechanics allow two ways to subtract: one putting '`-a`' card next to '`a`' card, and another one putting '`-a`' card over '`a`' to merge them to '`0`'. From now on, we will call them 'normal subtraction' and 'subtraction with merging'. Which move to use among them does not affect to clear the game, but the game can be cleared more efficiently with the latter move. The third transition is addition making the current equation to the equation `x+a+a=b+a`. This is not the right way to solve the level, because the DragonBox is not going to be isolated. We will call it 'unnecessary addition'.

For the normal subtraction, the predictive power is more than tripled. We believe our predictor successfully captured this habitual movement, which we also believe is not likely to change once fixed. Indeed, DragonBox Adaptive does not provide a strong reward on decreasing the number of movements, nor suggest a guide to promote the students using the subtraction with merging. For the unnecessary addition, the predictive power is also more than tripled. Because one must have seen similar problems several times, students rarely makes this mistake (3.8% in training data). In spite of such sparsity, our predictor improved its predictive power over that of the Markov predictor.

6.1.4 *Learned Features and Feature Weights*

In this subsection, we take a look at the learned features and classifiers. Most of the learned features with high weights are closely related to the task we are trying to predict when inspected by experts. Table 2 shows some of the learned features with high weights in the classifier for each transition in the previous subsection: subtraction with merging, normal subtraction, and unnecessary addition. Most of the selected features with high weights are actually related to the action that it is going to predict. For example, the movement in the first feature in the table `x+a=b` → `x+0=b-a` is exactly same as the movement we are trying to predict. The only difference is the level IDs. Note that having a feature set of the future level (`lv3.5.1`) is possible because our game progression forces students to visit previous levels when a student fails to clear a level.

For another example, the movement in the eleventh feature in the table `x+a+a=a+b` → `x+a+0=b+0` implies an unnecessary

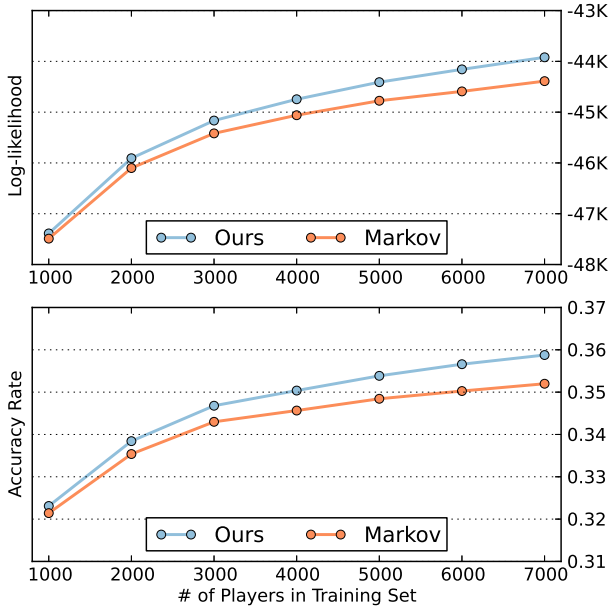

Figure 5: Performances of two methods on Refraction.

addition was performed, because the game level `1.18` starts from `x+a=b`: to reach the equation `x+a+a=a+b`, one has to perform the unnecessary action ahead. It is also interesting that the game level `1.18` is almost 25 levels away from the current level `3.4.1`. It would be a natural assumption that only recent playdata affect a user's behavior, but this provides an evidence that this is not the case. Considering that a player can only proceed a level after correcting previously made mistakes, it also implies the learning curve of this concept is relatively shallow.

One more thing to mention is that when a feature is describing more recent part of the playdata, it tends to have a higher weight. In the table, features from level 3 usually receives higher weights compared to the features from level 1 or 2. This is intuitive because a user is more likely to change his or her behavior as time passes. This implies that our model is also capturing the temporal behavior of individuals.

6.2 Refraction

To demonstrate that our framework can be used on other systems without additional authoring, we also run an experiment with another educational game, Refraction [4]. As we use the same experimental setting and game data as in Liu et al. [15], we omit the details of the game and the state-action model. The dataset contains 8,000 users' gameplay data, with about 70,000 states, 460,000 transitions, and 360,000 moves. 1,000 users' gameplay data is assigned to a test set, and the rest as a training set. We use the gameplay data from level 1 to level 8 to predict the movements in level 8. There are no changes in our framework, except that the smoothing parameter ϵ in the log-likelihood metric is set to 0.3 to match the performance of the Markov predictor used in previous work [15].

Figure 5 shows the performance of our predictor and the Markov predictor. We see our predictor performs better than the Markov predictor, although the improvement is much smaller compared to DragonBox Adaptive. We believe this is because level 8 is an early level, and we do not have enough data. Level 8 is the first level without the tutorial, it would be difficult to detect a confident signal describing individual behavior. In other words, students did not have enough opportunity to show their personality. We could not run another experiment on a later level due to lack of players from the high drop-out rate. Moreover, the Refraction dataset (360,000 moves) is much smaller than the DragonBox Adaptive dataset (21 million moves), while the total number of transitions is similar in both.

Nevertheless, we successfully showed that our framework can be used in a different system with no additional expert authoring, and showed our predictor still performs better than the Markov predictor.

7. CONCLUSION AND FUTURE WORK

Modeling user behavior in open-ended environments has the potential to greatly increase undestanding of human learning processes, as well as helping us better adapt to students. In this paper, we present a data-driven individual policy-learning framework that reduces the burden of hand-designing a cognitive model and system-specific features. Our framework automatically selects relevant features from a default feature set defined on a general state-action graph structure, and learns an individual policy from the trajectory of a player. We apply our method to predict player movements in two educational puzzle games and showed our predictor outperforms a baseline predictor. We also show that the performance improvement comes not only from frequently observed movements, but also from sparsely observed erroneous movements. Finally, we see our robust feature selection makes the predictor more efficient, powerful, and interpretable by investing its resources on a small set of influential states and relevant features.

We see numerous opportunities for further improvement of our framework. First, we can experiment with different classifiers instead of a logistic regressor. Since a logistic regression model is a single layer artificial neural network (ANN), we believe using a multi-layered ANN is a natural extension to improve its predictive power. Using an ensemble of classifiers would be another way to boost the performance. Second, we can add more graph navigation features into the default feature set. A feature specifying whether a transition is not visited only when it was available to the user is the first thing to try, because it specifies whether a certain behavior has been avoided intentionally or if the user simply did not have an opportunity to make such a choice. A visit indicator of a specific chain of transitions or the time spent on a certain state can also be possible features. Finally, we can try other feature selection techniques. Recursive feature elimination or L1-based feature selection might produce a better result because univariate approaches, such as our chi-squared test, do not consider the effect of multiple features working together [10].

Overall, we are also very interested in building applications based on our framework. Integrating the individual behav-

ior predictor into user-specific content generation such as a personalized hinting system or an adaptive level progression would be the first step. Moreover, we believe our framework will be also useful in other fields for learning individual behavior, such as spoken dialogue systems [14], robotic learning from demonstration [7], and recommender systems [18].

8. ACKNOWLEDGMENTS

This work was supported by the University of Washington Center for Game Science, the National Science Foundation Graduate Research Fellowship grant No. DGE-0718124, the Bill and Melinda Gates Foundation, and Samsung Scholarship Foundation. The authors would also like to thank Travis Mandel for helpful discussions.

9. REFERENCES

[1] Dragonbox. `http://www.dragonboxapp.com/`. *WeWantToKnow*, Retrieved 2014-02-27.

[2] Dragonbox adaptive. `http://centerforgamescience.org/portfolio/dragonbox/`. *Center for Game Science*, Retrieved 2014-02-27.

[3] Norway algebra challenge. `http://no.algebrachallenge.org/`. *Algebra Challenge*, Retrieved 2014-02-27.

[4] Refraction. `http://centerforgamescience.org/portfolio/refraction/`. *Center for Game Science*, Retrieved 2014-02-27.

[5] D. W. Albrecht, I. Zukerman, and A. E. Nicholson. Bayesian models for keyhole plan recognition in an adventure game. *User modeling and user-adapted interaction*, 8(1-2):5–47, 1998.

[6] V. Aleven, B. M. McLaren, J. Sewall, and K. R. Koedinger. The cognitive tutor authoring tools (ctat): Preliminary evaluation of efficiency gains. In *Intelligent Tutoring Systems*, pages 61–70. Springer, 2006.

[7] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[8] Y. Bergner, S. Droschler, G. Kortemeyer, S. Rayyan, D. T. Seaton, and D. E. Pritchard. Model-Based Collaborative Filtering Analysis of Student Response Data: Machine-Learning Item Response Theory. In *EDM*, pages 95–102, 2012.

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

[10] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

[11] R. K. Hambleton, H. Swaminathan, and H. J. Rogers. *Fundamentals of item response theory*. Sage, 1991.

[12] A. R. Jansen, D. L. Dowe, and G. E. Farr. Inductive inference of chess player strategy. In *Proceedings of the 6th Pacific Rim International Conference on Artificial Intelligence*, PRICAI'00, pages 61–71, Berlin, Heidelberg, 2000. Springer-Verlag.

[13] L. A. Jeni, J. F. Cohn, and F. De La Torre. Facing imbalanced data–recommendations for the use of performance metrics. In *Affective Computing and Intelligent Interaction (ACII), 2013 Humaine Association Conference on*, pages 245–251. IEEE, 2013.

[14] D. J. Litman and S. Pan. Designing and evaluating an adaptive spoken dialogue system. *User Modeling and User-Adapted Interaction*, 12(2-3):111–137, 2002.

[15] Y.-E. Liu, T. Mandel, E. Butler, E. Andersen, E. O'Rourke, E. Brunskill, and Z. Popović. Predicting player moves in an educational game: A hybrid approach. In *EDM*, pages 106–113, 2013.

[16] M. Mavrikis. Data-driven modelling of students' interactions in an ILE. In *EDM*, pages 87–96, 2008.

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

[18] M. J. Pazzani and D. Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

[19] M. Prensky. Computer games and learning: Digital game-based learning. *Handbook of computer game studies*, 18:97–122, 2005.

[20] S. Ritter, T. K. Harris, T. Nixon, D. Dickison, R. C. Murray, and B. Towle. Reducing the Knowledge Tracing Space. In *EDM*, pages 151–160, 2009.

[21] D. W. Shaffer. *How computer games help children learn*. Macmillan, 2006.

[22] G. Synnaeve and P. Bessière. A bayesian model for plan recognition in rts games applied to starcraft. In *Proceedings of the Seventh Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2011)*, pages 79–84, 2011.

[23] B. Tastan and G. R. Sukthankar. Learning policies for first person shooter games using inverse reinforcement learning. 2011.

[24] N. Thai-Nghe, L. Drumond, A. Krohn-Grimberghe, and L. Schmidt-Thieme. Recommender system for predicting student performance. *Procedia Computer Science*, 1(2):2811–2819, 2010.

[25] N. Thai-Nghe, T. Horváth, and L. Schmidt-Thieme. Factorization Models for Forecasting Student Performance. In *EDM*, pages 11–20, 2011.

[26] Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 412–420, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[27] M. V. Yudelson, K. R. Koedinger, and G. J. Gordon. Individualized bayesian knowledge tracing models. In *Artificial Intelligence in Education*, pages 171–180. Springer, 2013.

[28] A. Zook and M. O. Riedl. A temporal data-driven player model for dynamic difficulty adjustment. In *AIIDE*, 2012.