High-level Student Modeling with Machine Learning

Joseph E. Beck and Beverly Park Woolf

Computer Science Department
University of Massachusetts
Amherst, MA 01003
U.S.A.
{beck,bev}@cs.umass.edu

Abstract. We have constructed a learning agent that models student behavior at a high level of granularity for a mathematics tutor. Rather than focusing on whether the student knows a particular piece of knowledge, the learning agent determines how likely the student is to answer a problem correctly and how long he will take to generate this response. To construct this model, we used traces from previous users of the tutor to train the machine learning agent. This agent used information about the student, the current topic, the problem, and the student's efforts to solve this problem to make its predictions. This model was very accurate at predicting the time students required to generate a response, and was somewhat accurate at predicting the likelihood the student's response was correct. We present two methods for integrating such an agent into an intelligent tutor.

1 Introduction

AnimalWatch is an intelligent tutor for teaching arithmetic to grade school students[3]. The goal of the tutor is to improve girls' self-confidence in their ability to do math, with the long-term goal of increasing the number of women in mathematical and scientific occupations. AnimalWatch has been shown to increase the self-confidence of girls who use it[3]. The tutor maintains a student model of how students perform for each topic in the domain, and uses this model to select a topic on which the student will work, to construct a problem at the appropriate level of difficulty for the student, and to determine which feedback is best for the student.

Most student models are concerned with representing the student's ability on portions of the domain. Although useful, it is not always obvious how to map this low-level knowledge to higher level teaching actions. Given that the main purpose of student models for intelligent tutors is to support such decision-making, this is an odd situation.

To overcome this difficulty, we have implemented a machine learning (ML) architecture that reasons at a coarser grain size. We are not interested in low-level cognitive information, rather, we want something that can directly apply to the tutor's decision-making. Specifically, the agent learns to predict the probability the student's next response will be correct, and how long it will take the student to generate that response.

An advantage of this framework is that it permits the centralization of the tutor's reasoning. All of the teaching decisions of AnimalWatch (which makes similar decisions to many intelligent tutors) can be made via this module. First we will discuss

related research. Next we describe how our ML agent is constructed. We then discuss specific implementation details for using the agent with our tutor. Finally we conclude with a discussion of the results of our tests, and probable future work.

2 Background

Most intelligent tutors track how well a student is doing on each element in the domain to be taught (for simplicity, we will refer to these elements as "topics"). This is a good starting point, and many ITS have used this framework. Unfortunately, not much progress has been at modeling higher-level information about the student.

It is possible to use the tutor's beliefs of the student's abilities to make higher level predictions. For example, if there are 4 steps required to solve a problem, the probability the student will produce the correct response can be found by multiplying together the probabilities that he knows each rule. This model can be enhanced by accounting for correct guessing and "slips" by a proficient student[1]. However, it does not account for information about how the student has performed on this problem. If he has made 10 prior mistakes, the probability the next answer is correct would probably differ from that of a student who has yet to make his first response. Furthermore, the difficulty of the problem must be considered. A student is more likely to answer a question that (in arithmetic) has small numbers. It is difficult to see how these types knowledge could be embedded in the tutor's beliefs of the student's ability on a topic. Some means of extending "student" modeling[1] to the domain must be made.

Work on the ANDES system[8] has addressed some of these issues. ANDES constructs a Bayesian network representing potential solutions to the current problem. The student's proficiencies and actions are used with this network to determine on which steps the student will need help, and his probable method for solving the problem. This combination of reasoning about problem structure and student knowledge is powerful.

Other work at abstracting higher level information about the student includes NeTutor[11], which determined under which teaching conditions a student best performed. For example, the system determined whether the student learned best with a directed or exploratory learning tasks, and used that information to guide its teaching decisions.

There has also been research in automatically constructing executable models of student behavior. Input-output agent modeling (IOAM) examines student solutions to problems, and constructs a set of rules that describe the student's behavior. This requires a detailed description of the state, but produces an agent that makes similar mistakes (and correct answers) as the actual student[7].

3 Architecture

Given our goal of learning whether a student will generate a correct response, and how long he will take to generate this response, it would be nice to find a low-cost method for doing this. It is certainly possible to explicitly model how students generate their

[1] Finding a good term is problematic. For the tutor to better reason about the student, it needs to model information beyond what the student knows/believes..

answers, but this would be time consuming. Therefore, we have constructed a machine learning agent that acts as a "black box" when making predictions. The exact mechanism used by the learning agent is secondary[2]. Any machine learning method for performing function approximation will (in theory) work. The critical issues are what are the model's inputs/outputs, and how the model can be integrated into an ITS.

3.1 What are the inputs/outputs?

When the student is presented with an opportunity to provide an answer, the system takes a "snap shot" of its current state. This picture consists of information from 4 main areas:

1. **Student:** The student's level of proficiency and level of cognitive development
2. **Topic:** How hard the current topic is and the type of operand/operators.
3. **Problem:** How complex is the current problem.
4. **Context:** Describes the student's current efforts at answering this question, and hints he has seen.

After the student makes his next response, the system records the time the student required to make this response and whether it is correct. These two items of information are what the learning agent is trying to predict.

A useful feature of this model is that it is executable. That is, when presented with a problem, it returns characteristics of the student's response. This information is enough for the tutor to make a teaching decision (in the current example, presenting a hint). Information about the student's response and the hint presented are used to generate a new set of context features (see Section 4.2for more detail). This represents the student seeing a problem, attempting to answer it, and the tutor providing feedback. At this point the new context information is fed back to the ML agent, and the new prediction represents the student's second attempt at answering the problem. Combining the ML agent with the tutor in this manner results in a simulated student[14].

Figure 1 provides an overview of this process. The student, topic, and problem information are assumed to be fixed (but can be initialized with arbitrary values). The predicted student performance in this situation is combined with the resulting teaching action to produce a new set of context features. Since the context describes all of the information about what the student has seen and done, this is the only information that needs to change. If the ML-agent predicts the student will solve the problem, that terminates the simulation cycle.

3.2 How to use the model?

Simply having a model that predicts properties of the student's next response is of limited use unless there is some method for utilizing these prediction by a decision making agent. There are several possibilities for using model we have described. One is simply to construct the tutor with a set of "if-then" rules that use this model for additional accuracy. For example, a rule could be of the form "if the student has spent

[2] We have successfully used naive Bayesian classifiers, decision trees, and linear regression.

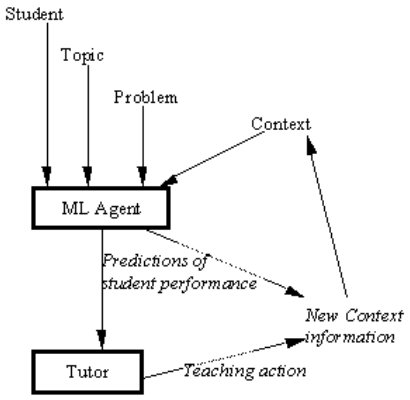

Fig. 1. Schematic for simulated student.

more than 90 seconds on the problem, provide a hint that makes it very likely he will get the problem correct." (as opposed to providing a small amount of information).

One problem with this is the tutor does not take any proactive steps to ensure the student does not approach a bad situation. E.g. the above rule implies taking a long time per problem is not the desired goal, but none of the previous teaching decisions (what problem to present, which topic to work on, etc.) took this knowledge into account.

To overcome this difficulty, we have integrated this model of the student with a reinforcement learning[13] agent whose goal is to learn a teaching policy that keeps the student in "good" learning situations. More details about this agent are available at[6]. This learning agent takes a learning goal and an executable model of the student as input. To follow the above example, the agent could be told it will receive a high reward if the student takes less than 90 seconds to solve a problem, and a low reward otherwise. The agent then uses the simulation of student behavior to experiment with different teaching actions in different contexts. Eventually (hopefully) the agent finds a teaching policy that makes it likely the student will stay in desired states.

4 Implementation issues

Section 3 gives a high-level description of the learning architecture that could be implemented in several ways for a variety of tutors. We now discuss specific implementation details with integrating a learning agent into AnimalWatch.

4.1 AnimalWatch background

AnimalWatch has a series of *topics*, which are items about which it can ask questions. *Operators* include addition, subtraction, multiplication, and division. *Operands* are wholes and fractions. There are also topics on "pre-fraction" skills that do not fit this hierarchy well. Each topic may have a number of *subskills* which are optional steps that

may be needed to solve a problem[4]. For example, simplifying the result is a subskill of add fractions. Subskills have a level of difficulty associated with them. Borrowing across several columns in a subtraction problem is more difficult than borrowing from the column directly to the left.

The tutor provides feedback when students make an incorrect response. There are a variety of types of hints it can provide: message hints, interactive hints that demonstrate the procedure involved, and interactive hints that emphasize the underlying concepts.

4.2 Training data and features

Since the tutor records the state of the system every time the student enters a response, it is easy to gather large amounts of training data. Approximately 120 students have used AnimalWatch in two local elementary schools. This yielded 11,000 training instances[3].

All of the data collected were from rural/suburban schools. One class was in the fourth grade, the other in the fifth grade. A question is how well these data generalize to other populations, such as older/younger students, and students in an urban setting.

There are several (48 with the most recent implementation) different features provided to the learning agent. As outlined in Section 3.1, these features can be broken down into several broad categories. Table 1 itemizes these features.

As is typical for machine learning, the design of the features is dependent on the type of learning algorithm used. For example, the operator and operand were initially stored as simple integers. E.g. 1 for addition, 2 for subtraction, 3 for multiplication, and 4 for division. For ML techniques that reason by categories (e.g. naive Bayesian classifiers) this is sufficient. But for simple linear techniques, it is necessary to create 4 separate inputs only one of which is active at a time (i.e. a "one-hot" encoding scheme). The first inputs would be on for addition (while the other three are off), for a subtraction problem the second input would be on and the first, third, and fourth off, etc.

More generally, this is a tradeoff between complexity of the learning algorithm and complexity of the feature space. General techniques can learn with weaker sets of features. Limited techniques, such as linear regression, require a more explicit encoding scheme. Another example of this is the set of combination features. Since regression, the technique we decided to use, cannot handle interactions between inputs, it is necessary to create these interaction features by hand. In this case, we had data indicating that particular Piaget cognitive development questions were relevant to certain topics[2]. Our eventual goal is to migrate to more powerful learning techniques so this lower level engineering is not required. However, this is a more difficult task.

4.3 Function approximator

Given a set of features to describe a state, some means of predicting what will happen next is needed. We settled on using linear regression for the simple reason that it is quick

[3] Certain data were removed. Specifically, students with learning disabilities (e.g. unable to read the problems without the help of a proctor) were not considered in the analysis. We also trimmed response times in excess of 6 minutes. There were very few responses in this range, and these data are likely confounded by students needing to leave the room for a few minutes.

Feature type	*Information*
Student	– Gender – Performance in 9 tasks of Piagetian level of cognitive development[2] – Combined score for Piaget tasks – Proficiency at current topic
Topic	– Operator (one-hot encoding) – Operand (one-hot encoding)
Problem	– Problem difficulty[4] – Size of operands/answer – Number of subskills required[4] – Difficulty of subskills
Context	– Is this student's first attempt? – Number of prior mistakes – Best hint seen – Current hint depth (hints can create subhints) – Maximum hint depth on this problem – Best hint seen – Has student seen a "strong" hint? – Time spent on this problem
Combination	– Particular piaget tasks X topic type

Table 1. List of features for learning agent

and easy to determine if your model is accurate or not, and has a fast execution speed to learn and make predictions. This is a non-trivial concern. AnimalWatch is written in Java, and since we will be integrating the learning components into the system it would be very useful if they were also written in Java. Most ML schemes either use gradient descent (e.g. linear function approximators, neural networks) or use categorical data (e.g. decision trees, naive Bayesian classifiers). Given that we have a long-term goal of learning online, a simple, fast technique such as regression is a good fit.

Our technique of using a simple function approximator to predict observable variables has the advantage of being straightforward to construct. There is not a large computational or research cost of building our model, and no need to try to understand the student's mental state. This is in keeping with Self's[12] recommendations.

Linear regression was used to construct two equations. One equation used the features described in Section 4.2 as dependent variables to predict the amount of time required, the other equation predicted whether the student's response would be correct. Whether a response is correct is a binary value, and in a sense it is odd to apply regression to this situation. However, we are interested in the probability a student will generate a correct response. Consider a situation where 51% of the time the student has no difficulties with a problem, while 49% he makes a mistake and has a difficult time with the problem. We would not want to use a technique that ignored the second possibility. Therefore we use the regression model to predict the mean correct score

(incorrect is 0, correct is 1), and compare this with a random number. I.e. if the model predicts an average correctness of 0.8, 80% of the time the system assumes the student provides a correct response, 20% of the time the system assumes an incorrect response.

5 Results

To evaluate the accuracy of the regression models, we correlated the predicted and actual results. The model for predicting the amount of time correlated with the actual times at 0.629. A 2-fold cross validation resulted in a correlation of 0.619, so the regression model learned something other than statistical noise in the data as it was able to predict times for instances which it had not seen. Figure 2 shows the model's performance.

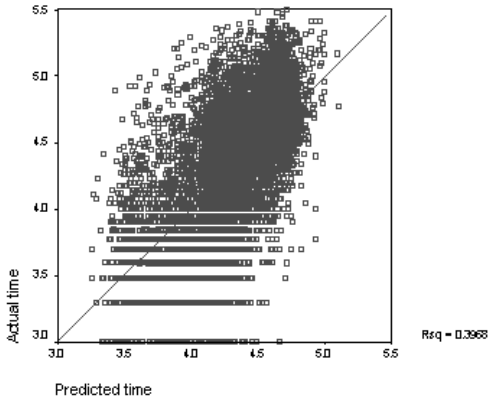

Fig. 2. Predicted vs. actual time for a student's response.

The x-axis is the predicted log-time (measured in milliseconds) by the regression model; the y-axis is the actual log-time[4]. The best-fit linear regression gave an r-square of 0.397. I.e. this model reduces the squared error by approximately 40% when compared to guessing randomly.

The linear regression model was less accurate at predicting whether the student would generate a correct response. The correlation was only 0.496, which gives an r-squared of 0.243. A model that always guessed the student would give a correct response would be correct 60% of the time, while this model's predictions are correct 74% of the time. This does not seem overly encouraging. However, Figure 3 shows the model's performance from a different perspective. The model's predicted values were broken into 101 bins (0, 0.01, 0.02, ... , 0.99, 1.0), the student's probability of generating

[4] The logarithm of time was used as this linearized the data.

a correct response for all of the cases in each bin was computed, and the resulting set of $< predicted, actual >$ pairs scatterplotted[5].

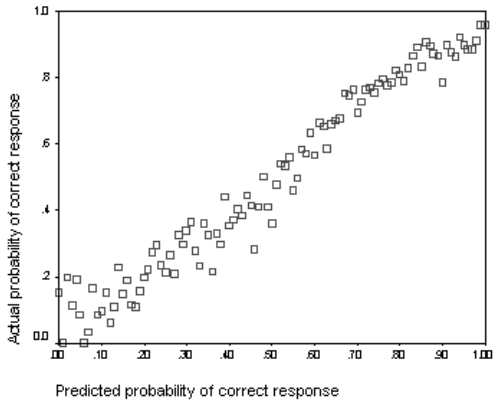

Fig. 3. Accuracy of predicting student responses.

The x-axis is the model's predictions for the probability the student's response will be correct. The y-axis is the actual probability the student's response was correct. For example, when the model thought there was a 100% chance the student would generate a correct response (the right most dot in the graph), the student in fact had roughly a 95% chance of generating a correct response.

According to this graph, the model appears to be very accurate. As it believes the student is more likely to give a correct response, the student is in fact more likely to do so. The disparity between the correlation coefficient, which seems fairly weak, and the graphical display of the model, which seems quite strong, is striking. For an explanation, consider the cases in the middle of the graph (say predicted values of 0.4 to 0.6). Even if the model is correct, and students have a 40% to 60% chance of generating correct responses, the model *cannot* have a high degree of accuracy. Assume the model is deterministic, and always guesses the most likely category (the fact that it in fact selects randomly random does not effect the following results, but would complicate the argument). For cases in the 40% correct range, the model will guess the student will make a mistake, and will be correct 60% of the time. For cases in the 50% range, the model will be right 50%. For cases in the 60% range, the model will guess the student's response is correct, and will be right 60% of the time. Assuming an even distribution of cases, this is an accuracy of 56.7%.

Considering all possible probability values from 0 to 1, such a model would have a maximum accuracy of 75%. Since there are no cases where a student answers a questions with a particular probability (he gets it right or he doesn't, there is no in between),

[5] We are not trying to perform a statistical sleight of hand. We cannot simply graph a scatterplot of predicted vs. actual values since the actual value has only 2 possible values.

it is an interesting question as to whether such a model is useful or useless. As we discuss later, this depends on how the model will be used.

6 Conclusions and future work

We have constructed a model of the student at a coarse grain size and are using this executable model as a simulated student. Our belief is that there is much to be gained by modeling higher level descriptions of students. This has a more natural mapping onto rules describing teaching decisions, and can be used as a simulation to train an agent how to teach students.

The ML agent has done a good job at modeling how long the student will require to generate a response. Accounting for 40% of the variance of time data is difficult, and performance should improve if a more complex function approximator is used. The data for each individual student were not critical for this performance. Each student comprised less than 1% of the training data and could not have a large impact on the model. A question is whether attempting to learn online about each individual would significantly improve accuracy. Prior work[5] found a benefit from learning about individual students to predict time to solve a problem.

It is unclear if the equation for predicting whether the student will answer a question correctly is accurate enough. If the goal is to predict whether the current student will generate a correct response, and the decision is high stakes, then the model is probably not accurate enough. If instead, the goal is to provide a simulation of a student for another learning agent[6], the model is probably accurate enough "on average". As seen in Figure 3 the regression equation generally agrees with the observed data for how often a student will generate a correct response. If a model can determine that for a certain class of situations, the student has a 50% chance of answering correctly, then this should be sufficiently accurate (even though the model is right only half of the time). This is a subtle issue, and needs to be further explored. Intuitively, it is seems likely that online learning would give large increases in accuracy of predicting the student's correctness of response. Informal observation reveals that students have widely varying thresholds for how conscientious they are before providing a response: some recheck their work, others are willing to guess wildly.

We will use more complex machine learning techniques in the future. One possibility is non-linear regression. This is very good for offline learning, but if the model is to be improved it is necessary to store every piece of the data used to construct the model. This is a large drawback if learning is to continue online while the student uses the tutor. Neural networks learn slowly, but incrementally, and can be represented with much less storage space than a comparable non-linear regression and datapoints. As an abstract AI problem, this issue is not important. For deploying a tutor on Pentium and PowerMac class computers in the typical classroom, this is important. To date there has been little discussion of the tradeoffs of lab vs. classroom AI technology.

Although this framework has been presented in the context of AnimalWatch, the architecture is general. Only the details in Section 4 would need to be updated for use with a different tutor. The concept of using a large population of student data to directly

train a learning agent is widely applicable. With the scale up in evaluation studies[9, 10], this technique becomes more feasible.

Acknowledgements

We acknowledge the support for this work from the National Science Foundation, HRD-9714757. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the granting agency.

References

1. J. Anderson. *Rules of the Mind*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1993.
2. I. Arroyo, J. E. Beck, K. Schultz, and Woolf. Piagetian psychology in intelligent tutoring systems. In *Proceedings of the Ninth International Conference on Artificial Intelligence in Education*, 1999.
3. Carole R. Beal, Beverly P. Woolf, Joseph Beck, Ivon Arroyo, Klaus Schultz, and David M. Hart. Gaining confidence in mathematics: Instructional technology for girls. In *Proceedings of International Conference on Mathematics/Science Education and Technology*, 2000.
4. J. E. Beck, M. Stern, and B.P. Woolf. Using the student model to control problem difficulty. In *Proceedings of the Seventh International Conference on User Modeling*, pages 277–288, 1997.
5. J. E. Beck and B. P. Woolf. Using a learning agent with a student model. In *Proceedings Third International Conference on Intelligent Tutoring Systems*, 1998.
6. Joseph E. Beck and Beverly P. Woolf. Learning to teach: A machine learning architecture for making teaching decisions. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, 2000.
7. Bark Cheung Chiu and Geoffrey I. Webb. Using decision trees for agent modeling: Improving prediction performance. *User Modeling and User Adapted Interaction*, 8(1-2):131–152, 1998.
8. A. S. Gertner, C. Conati, and K. VanLehn. Procedural help in ANDES: Generating hints using a Bayesian network student model. In *Fifteenth National Conference on Artificial Intelligence*, pages 106–111, 1998.
9. K. R. Koedinger, J. R. Anderson, W. H. Hadley, and M. A. Mark. Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, 8:30–43, 1997.
10. Thomas N. Meyer, Todd M. Miller, Kurt Steuck, and Monika Kretschmer. A multi-year large-scale field study of a learner controlled intelligent tutoring system. In *Proceedings of the Ninth International Conference on Artificial Intelligence in Education*, pages 191–198, 1999.
11. M. Quafafou, A. Mekaouche, and H.S. Nwana. Multiviews learning and intelligent tutoring systems. In *Proceedings of Seventh World Conference on Artificial Intelligence in Education*, 1995.
12. J.A. Self. Bypassing the intractable problem of student modelling. In C. Frasson and G. Gauthier, editors, *Intelligent Tutoring Systems: at the Crossroads of Artificial Intelligence and Education*, pages 107–123, Norwood, NJ, 1990.
13. R.S. Sutton and A.G. Barto. *An Introduction to Reinforcement Learning*. MIT Press, 1998.
14. K. VanLehn, S. Ohlsson, and R. Nason. Applications of simulated students: An exploration. *Journal of Artificial Intelligence in Education*, 5(2):135–175, 1994.