Reasoning from data rather than theory

Joseph E. Beck and Beverly P. Woolf

Computer Science Department University of Massachusetts Amherst, MA 01003

Abstract

The current framework for constructing intelligent tutoring systems (ITS) is to use psychological/pedagogical theories of learning, and encode this knowledge into the tutor. However, this approach is both expensive and not sufficiently flexible to support reasoning that some system designers would like intelligent tutors to do. Therefore, we propose using machine learning to automatically derive models of student performance. Data are gathered from students using the tutor. These data can come from either the current student or from previous users of the tutor. We have constructed a set of machine learning agents that learn how to predict "high-level" student actions, and use this knowledge to learn how to teach students to fit a particular learning goal. We discuss other related work at using machine learning to construct models within ITS. By combining several different systems, nearly all of an ITS's decision-making could be performed by machine learner derived models. An open question is the complexity of combining these diverse architectures into a single system.

Introduction

This paper discusses an alternate approach to constructing intelligent tutoring systems (ITS). Typically, such systems are built using cognitive and pedagogical models to guide the tutor's reasoning. Our concern is the cost of constructing such models as well as their accuracy. We discuss using machine learning to automate much of this process. First, we will describe our results at using machine learning to make teaching decisions. Second, we will examine related work that addresses other components of an ITS. We will then discuss how to merge these different systems together to construct an architecture for ITS construction that is data-driven and uses little *a priori* reasoning.

Classical design

The intelligence in intelligent tutors has been largely done by implementing various theories of skill acqui-

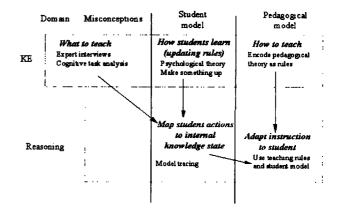


Figure 1: High-level view of ITS components.

sition and teaching. The "cognitive tutors" (Anderson 1993) are perhaps the best known example of this. A detailed cognitive model of how people perform a task is constructed, and the tutor's goal is to determine how the student has deviated from this model. A hard problem is finding some method of mapping a student's interactions with the tutor to this mental state.

Figure 1 provides an overview of a fairly typical ITS architecture. Several types of knowledge engineering (KE) must be performed, as it is necessary to specify the material to be taught, potential misconceptions, how the student will learn the material, and a set of pedagogical rules for teaching. Once this is complete, some method of reasoning with the data collected is needed. This results in a complex design that is very knowledge intensive.

Motivation

One problem with using cognitive/pedagogical models to construct ITS is the expense. It is a complex undertaking to build a detailed cognitive model of a task, and mapping overt user actions to this internal mental model is also non-trivial. If it were possible for ITS to be deployed widely and used by many students, this high up-front cost might be acceptable, as the perstudent cost would be low. However, for a variety of

Copyright © 2000, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

reasons, this has not occurred. One reason is institutional acceptance, which is a major factor blocking the deployment of ITS(Bloom 1996).

Few organizations are willing to accept teaching software that cannot be modified. Given the expense in changing how an ITS performs, overcoming this problem is difficult. We have started to see broader acceptance of ITS(e.g. (Koedinger *et al.* 1997)), largely through a cooperative effort with educators to meet the latest educational standard. Although, one is left to wonder what will happen when the latest "hot" educational philosophy changes. There is no strong consensus on what is useful for students to learn. Is collaborative/cooperative learning going to be considered useful in ten years? Is memorization ever appropriate? One wonders how models that prescribe how to teach can be accurate when the goals they must meet are not stable.

In addition to expense and lack of flexibility, it is unclear how well models developed for the classroom apply to one-on-one computer instruction. Humans and computers have very different communication strength (for example, gesturing and voice intonation, and threedimensional animation, respectively). Methods that appear obvious to the ITS community, such as student modeling, although useful, are frequently not used by human tutors(Ur & VanLehn 1995).

Furthermore, there has been little work at evaluating the accuracy or effectiveness of these components. If the models are inaccurate, or only a minimal amount of accuracy is necessary for the tutor to act effectively, the work is going for naught.

With computer hardware in classrooms becoming capable of running tutors that were previously research prototypes, large scale trials become possible. With the advent of network-based tutors¹, it is possible to gather a large amount of information from human trials with tutors. Previously, trials were small in nature, and collecting data from remote locations was difficult. With the possibility of automatically uploading the results of students using the tutor to a central repository, there are significantly more data with which to reason.

Ideally, such data could be used to direct the ITS's teaching and student modeling. Unfortunately, a collection of data is fairly useless without a model/theory to provide organization. Thus, some means of constructing a model must be found. Fortunately, machine learning is well-suited for this task.

Machine learning

Machine learning allows computers to reason and make predictions about situations they have not encountered previously. One such application is a pattern matcher or a classifier. The machine learner takes a set of inputs describing the object, and tries to determine to which category the objects belongs. This can be ap-

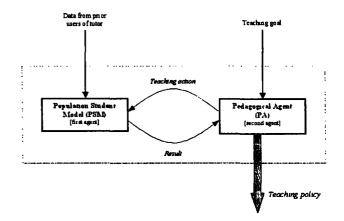


Figure 2: Overview of ADVISOR architecture.

plied, for example, to stereotypical student models(Kay 1994). A slightly more general view is to think of a machine learner as an automatic model generator. A linear regression is an example of this concept: the goal is to determine a function that best predicts the environment. A possible use of such a machine learner could be to automatically derive the equations used by Shute(Shute 1995) for updating a student's knowledge.

Many machine learning algorithms are robust to noise in the data, and construct models of a domain with a varying degree of human readability. This bring up an interesting point: is it acceptable to construct a "black box" model for use in an ITS? For a cognitive scientist, the answer would be no. A computer scientist, or someone trying to build an ITS cheaply would probably not care as much.

Our current work

To address issues of cost and accuracy of models used in an ITS, we have constructed a set of machine learning agents that learns how to teach students. Figure 2 provides an overview of this process. First, students using our tutor, named AnimalWatch, are observed, and these data provided to a learning agent that models student behavior. This Population Student Model (PSM)(Beck & Woolf 2000) is responsible for taking a context, and predicting how students will act in this situation.

The second component is the pedagogical agent (PA), that is given a high-level learning goal and the PSM. An example of such a high-level goal is "Students should make precisely 1 mistake per problem." The PA's task is to experiment with the PSM and find a teaching policy that will meet the teaching goal provided.

We gathered approximately 11,000 training instances from students using our AnimalWatch(Beal *et al.* 2000) tutor. Whenever the student entered a response, the system logged the correctness of the response, and the amount of time the student required. In addition, the system recorded the current "state" when the student attempted this response. This state is composed of 48

¹Tutors operating over a network, possibly the internet. Such a system may be web-based, or may simply use sockets to send information back to a central server.

features in four main areas:

- 1. Student: The student's level of prior proficiency and level of cognitive development(Arroyo *et al.* 2000)
- 2. **Topic:** How hard the current topic is and the type of operand/operators.
- 3. Problem: How complex is the current problem(Beck, Stern, & Woolf 1997).
- 4. Context: Describes the student's current efforts at answering this question, and hints he has seen.

High-level student modeling

Most student models are concerned with representing the student's level of ability on distinct 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 lowlevel 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 he will take to generate that response. Our goal is to find a method for automatically constructing accurate models of high-level student behavior.

We have applied several machine learning algorithms to attempt to construct a model to predict time/correctness of student response. Our current approach is to use linear regression. It is unlikely this will be our final choice, but classifiers (e.g. naive Bayesian classifiers) require discrete data. Discretizing all of our continuous variables would introduce a loss of accuracy. Numerical techniques tend to require parameter setting (e.g. α , or stepsize) and significant time to learn-and in the end provide no guarantees of finding a good solution. With linear regression it is possible to quickly consider different models and sets of features.

For predicting the amount of time a student will require to generate a response, our model's predictions have a correlation of 0.619 (train with half of dataset, test with the other half) with the actual amount of time students required to generate a response. For time data, accounting for 38% of the variance is fairly impressive. The model had mixed results at predicting student accuracy. The correlation was only 0.243, but as can be seen in Figure 3, this is somewhat misleading. On average the model had high performance, but for individual cases it may mispredict. In other words, the bias of the learning agent is fairly small, but the variance in its predictions is large. Depending on the use of the agent this may or may not be acceptable.

One benefit of using ML is the ability to reason about phenomena that are much closer to the level at which we wish to make teaching decisions. If we are considering presenting a particular hint in a certain situation,

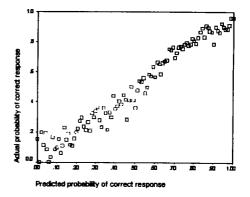


Figure 3: Accuracy of predicting student responses.

knowing how the student will react (in terms of time required to respond and probable accuracy of response) is more useful than knowing his ability on the skill. A second benefit is that we are able to include types of data not normally found in an ITS. For instance, we do not have a good theory for how level of cognitive development should impact hint selection. How then should such data be added to the model? Prior research has come across this same difficulty(Shute 1995; Beck, Stern, & Woolf 1997). By automating model construction, we bypass this issue entirely.

Also, our model of high-level performance is executable. Both previous mistakes and time spent on the problem are part of the *context* features maintained about a problem. Therefore, the model can predict the student's probability of a correct answer and expected time, update the context features appropriately, and continue making prediction of the student's efforts on this problem. This continues until the model predicts the student will answer the question correctly.

Teaching policies

The pedagogical agent (PA) uses the simulation of the student as described by the PSM, and experiments with different teaching actions/strategies to achieve an externally provided teaching goal.

We have implemented the pedagogical agent using reinforcement learning (RL)(Sutton & Barto 1998). Reinforcement learning uses a model of the environment (the Population Student Model), and a reward function (the pedagogical goal the agent is trying to achieve). This architecture of using a simulation is similar to that used in other complex RL tasks such as TDgammon(Tesauro 1995) and elevator dispatching(Crites & Barto 1998). For our RL mechanism, we decided to use temporal-difference (TD) learning(Sutton & Barto 1998). Specifically, TD(0) with state-value learning.

We have constructed an RL agent whose goal is minimizing the amount of time students spend per problem. Specifically, when the (simulated) student solved a problem, the RL agent was given a reward inversely proportional to the amount of time the student required to

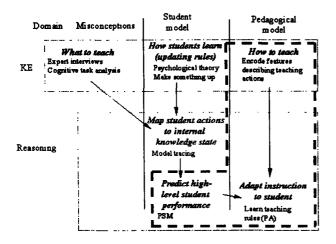


Figure 4: Adding ADVISOR into ITS architecture.

solve the problem. By attempting a variety of teaching actions with the simulated student, the PA was able to determine under which circumstances teaching actions resulted in students finishing problems quickly.

The trained PA was tested in an elementary school with 120 sixth-graders (≈ 11 years old). One group of students (the control) used the classical AnimalWatch tutor. A second group of students (the experimental group) used a tutor that reasoned with the machine learning architecture described here. Students in the experimental group required 27.7 seconds to solve a problem, while students in the control group required an average of 39.7 seconds to solve a problem. This difference is significant at P $\approx 10^{-23}$ (2-tailed t-test).

Revising the architecture

We will now consider how to replace much or all of the theory driven reasoning in an ITS by data-driven, machine learning methods. Figure 4 provides an overview of how ADVISOR fits into a traditional ITS architecture. Some of the reasoning and knowledge engineering has been replaced with ML derived models. However, much of the knowledge engineering and some of the reasoning are still being done with traditional mechanisms.

We will now consider other instances where components of an ITS's reasoning have been replaced by machine learning, and consider the possibility of integrating these attempts into one system.

Reasoning performed by machine learning

In order to automate (most of) the remaining reasoning in an ITS, some means of listing misconceptions, enumerating domain skills, and mapping over student actions to internal mental states are needed.

Teaching policies NeTutor (Quafafou, Mekaouche, & Nwana 1995) has a learning component that maps features of the teaching action (interactivity level, type of presentation, etc.) to expected amounts of the learning. The system uses this knowledge to select a teach-

ing activity that will (should) cause the student to learn the most. Rough set theory is used to control the list of possible teaching rules. One set contains the "certain" rules, while another contains "possible" rules.

An interesting component of this system is the use of "views". These are ways of partitioning the system's data into smaller, easier to understand components. For example, one view may contain features such as the topic being learned, and whether the student is solving a problem or reading some text. A second view may contain information such as what strategy is being used to present the information, the style of presentation, etc. This can be useful if humans are involved with the decision making, but is less beneficial if the machine is to perform all of the learning on its own.

It is interesting to note that this system's learning was done on-line while being used by an individual. This is a hard problem than our approach of first learning about a population of users offline.

Misconception detection/construction Prior research(Burton 1982) demonstrated that significant work is needed to explicitly enumerate all possible bugs in the student's knowledge. The ASSERT system (Baffes & Mooney 1996) uses theory refinement to build its model of student misconceptions. The system observes when students obtain an incorrect result, and modifies a correct rule base to be consistent with the student's behavior. Thus a runnable model of the student's behavior is maintained. This is a powerful technique, as it does not require bug libraries to be constructed ahead of time.

Furthermore, ASSERT could automatically modify teaching examples to demonstrate what was wrong with student misconceptions. Typically, a specific remediation lesson must be created for each misconception the tutor can detect. By automatically creating remediation, and having an open-ended set of possible misconceptions, ASSERT achieves very powerful teaching capabilities with low knowledge engineering costs.

Deriving student's knowledge Modeling a student's internal knowledge state is a challenging task, which has largely been done via model tracing. However, research using feature based modeling(Webb, Chiu, & Kuzmycz 1997; Chiu & Webb 1997) has managed to use machine learning techniques to construct executable models that describe student behavior. First, the ML agent is provided with a set of problems on which the student has worked, and a set of features to characterize problems and responses. The learning agent then automatically constructs a set of rules to explain the student's behavior. This model is custom-built for each student.

This modeling technique has been found to accurately predict student performance at a fine-grained level. In fact, it predicts each "step" in the problem solving process. The technique is general, with the possibility of using different induction engines to derive rules describing the student's performance(Webb, Chiu, & Kuzmycz 1997).

Another application of this work(Chiu & Webb 1997) has been to construct models of student misconceptions. To do this, several decision trees were constructed, one for each misconception. The nodes of the decision trees were the features of the problem, and the student's actions thus far in solving the problem.

Updating a student's estimated ability Many intelligent tutors use some form of Bayesian updating rule to estimate the student's current degree of proficiency on various topics within the domain. However, such update rules require some assumptions about how students learn the material, or at the very least, the models have parameters that must either be derived or estimated. Item response theory(IRT)(Hambleton, Swaminathan, & Rogers 1991) is a principled way of determining a student's ability at a particular skill based on his performance on test questions.

A powerful aspect of IRT is that, making no assumptions about the ability of the students or the difficulty of the questions, it can estimate both how hard the test questions are, and how skilled the students are. Typically this is done by field-testing questions with a pool of students. This serves to determine how difficult test items are. When an ITS is deployed, it is possible to use a student's performance on these items to predict his ability score (usually referred to as θ) in this area. This update rule takes into account both the prior estimate of the student's ability, the difficulty of the current test question, and his performance on this test question.

A combined architecture

The systems detailed above have made significant progress at performing much of an ITS's reasoning. Ideally, it would be possible to combine all of these individual efforts into one tutor that used machine learning techniques for nearly all of the system's reasoning. In some cases, this involves stretching a system beyond its initial design, but we will attempt to analyze what problems this may cause, and other alternatives available. IRT can update the tutor's estimate of the student's current level of knowledge, and feature based modeling can predict likely student mistakes, and thus determine probable reasons for student errors.

Figure 5 shows the possible coverage of existing machine learning components in an ITS. We have discussed automatically deriving a high-level model of performance, and learning how to teach. However, with this architecture it is possible to also automatically derive misconceptions. While the ASSERT system was limited to concept formation tasks, other research in this area has not been as restricted.

Discussion and future work

Actually building a system with such a combined architecture is unlikely to be straightforward, and there are several difficulties. First, there is the task of actually integrating all of these techniques together. Different

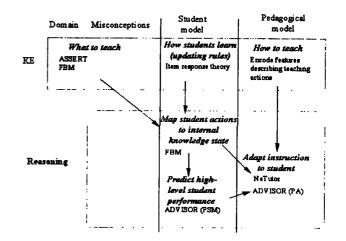


Figure 5: Combined architecture for ML-based ITS.

systems have made different assumptions in terms of amount of time interacting with each user, and types of domains for which they are applicable. Actually implementing all of the agents so that they work together is straightforward compared with this.

There is also the question of how to optimize each component of the learning architecture. This optimization is heavily dependent on both how the component is to be used, and what assumptions the designer is willing to make about the number of interactions with the student. For example, NeTutor and ASSERT assume a fairly small number of interactions with each student. and the ML algorithms have been chosen accordingly. In contrast, ADVISOR assumes a significant amount of prior data have been gathered and used to construct a population model. In addition, ADVISOR's model for predicting the accuracy of a student's response would not be useful for making critical, one-shot decisions. However, this model of accuracy works well when millions of trials are to be run (as in training an RL agent). This mismatch of assumptions may be problematic.

Generally, the choice for learning mechanism (e.g. decision tree, Bayesian classifier, etc.) comes down to how much training data will be available, and what types of predictions are needed. If there is a large amount of data, and predictions must be continuous, a neural network may be a good choice. For low-data situations (such as NeTutor), simpler mechanisms such as Bayesian classifiers are a better choice. In general, the exact reasoning method is not critical in getting systems to work together.

Another issue is what knowledge is still needed to construct tutors with this architecture. The need for some basic set of features to describe a state seems necessary. However, there has been research at having learning agents automatically constructing their own features(Utgoff 1996). Assuming it is necessary to construct a list of features, this could entail a moderate amount of work. It is necessary to describe the domain, the problem solving process, feedback, etc.

In fact *anything* about which the ML agents will reason must be described in this manner. However, finding ways to describe data should be much simpler than writing a collection of rules to reason about it. After all, the rules must have some way of understanding the information about which they are reasoning. In fact, the knowledge engineering tasks are fairly similar: constructing a list of features is analogous to building a language for the left-hand side of a set of rules.

Conclusions

In the past fifteen years there has been a shift away from AI based techniques to a psychological/pedagogical approach to ITS design. However, it may be time for a shift back to a more AI centered approach. Given the potential shown by machine learning techniques, automatic model construction has potentials that are at best hard to realize with traditional cognitive modeling techniques. For instance, the ability to incorporate unusual data (such as cognitive development), or to adapt the model to the current student (as in NeTutor) are not possible with traditional techniques.

There have been several small efforts within the AI&Ed community at applying AI techniques, but no system that integrates many techniques. While understandable, given the maturity of this research area, this is regrettable. By constructing tutors that use automated reasoning for all of their decision-making, we will develop a much clearer understanding of exactly what assumptions, particularly implicit ones, are part of our AI design processes.

Acknowledgements

Funding for AnimalWatch was provided by NSF HRD-9714757. We acknowledge the help of David Marshall in implementation, and Ivon Arroyo for both her work in implementation and design of features to describe AnimalWatch's teaching.

References

Anderson, J. 1993. Rules of the Mind. Hillsdale, NJ: Lawrence Erlbaum Associates.

Arroyo, I.; Beck, J. E.; Woolf, B. P.; Beal, C. R.; and Schultz, K. 2000. Macroadapting animalwatch to gender and cognitive differenes with respect to hint interactivity and symbolism. In *Proceedings of Fifth International Conference on Intelligent Tutoring Systems.*

Baffes, P., and Mooney, R. 1996. A novel application of theory refinement to student modeling. In *Proceedings* of the Thirteenth National Ameriacn Association for Artificial Intelligence, 403-408.

Beal, C. R.; Woolf, B. P.; Beck, J.; Arroyo, I.; Schultz, K.; and Hart, D. M. 2000. Gaining confidence in mathematics: Instructional technology for girls. In Proceedings of International Conference on Mathematics/Science Education and Technology. Beck, J. E., and Woolf, B. P. 2000. High-level student modeling with machine learning. In *Proceedings of Fifth International Conference on Intelligent Tutoring Systems*.

Beck, J. E.; Stern, M.; and Woolf, B. 1997. Using the student model to control problem difficulty. In Proceedings of the Seventh International Conference on User Modeling, 277-288.

Bloom, C. P. 1996. Promoting the transfer of adanced training technologies. In Frasson, C.; Gauthier, G.; and Lesgold, A., eds., *Proceedings of Third International Conference on Intelligent Tutoring Systems*, 1–10. Springer.

Burton, R. 1982. Diagnosing bugs in a simple procedural skill. In Sleeman, and Brown., eds., *Intelligent Tutoring Systems*, 157–182. Academic Press.

Chiu, B., and Webb, G. 1997. Using C4.5 as an induction engine for agent modelling: An experiment of optimisation. In Machine learning for user modeling workshop at the Seventh International Conference on User Modeling.

Crites, R., and Barto, A. 1998. Elevator group control using multiple reinforcement learning agents. *Machine Learning* 33:235-262.

Hambleton, R. K.; Swaminathan, H.; and Rogers, H. J. 1991. Fundamentals of Item Response Theory.

Kay, J. 1994. Lies, damned lies and stereotypes: Pragmatic approximation of users. In *Proceedings of Fourth International Conference on User Modeling*, 175–184.

Koedinger, K. R.; Anderson, J. R.; Hadley, W. H.; and Mark, M. A. 1997. Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education* 8:30-43.

Quafafou, M.; Mekaouche, A.; and Nwana, H. 1995. Multiviews learning and intelligent tutoring systems. In Proceedings of Seventh World Conference on Artificial Intelligence in Education.

Shute, V. 1995. Smart evaluation: Cognitive diagnosis, mastery learning and remediation. In *Proceedings* of Artificial Intelligence in Education, 123-130.

Sutton, R., and Barto, A. 1998. An Introduction to Reinforcement Learning. MIT Press.

Tesauro, G. 1995. Temporal difference learning and TD-Gammon. Communications of the ACM 38(3).

Ur, S., and VanLehn, K. 1995. Steps: A simulated, tutorable, physics student. Artificial Intelligence in Education 6(4):405-437.

Utgoff, P. E. 1996. ELF: An evaluation function learner that constructs its own features. Technical Report 96-65, University of Massachusetts.

Webb, G.; Chiu, B. C.; and Kuzmycz, M. 1997. Comparative evaluation of alternative induction engines for feature based modeling. *Artificial Intelligence in Education* 8:97-115.