A Bayes Net Toolkit for Student Modeling in Intelligent Tutoring Systems

Kai-min Chang, Joseph Beck, Jack Mostow, and Albert Corbett

Project LISTEN, School of Computer Science
Carnegie Mellon University, Pittsburgh PA 15213, USA
kkchang@cs.cmu.edu
http://www.cs.cmu.edu/~listen

Abstract. This paper describes an effort to model a student's changing knowledge state during skill acquisition. Dynamic Bayes Nets (DBNs) provide a powerful way to represent and reason about uncertainty in time series data, and are therefore well-suited to model student knowledge. Many general-purpose Bayes net packages have been implemented and distributed; however, constructing DBNs often involves complicated coding effort. To address this problem, we introduce a tool called BNT-SM. BNT-SM inputs a data set and a compact XML specification of a Bayes net model hypothesized by a researcher to describe causal relationships among student knowledge and observed behavior. BNT-SM generates and executes the code to train and test the model using the Bayes Net Toolbox [1]. Compared to the BNT code it outputs, BNT-SM reduces the number of lines of code required to use a DBN by a factor of 5. In addition to supporting more flexible models, we illustrate how to use BNT-SM to simulate Knowledge Tracing (KT) [2], an established technique for student modeling. The trained DBN does a better job of modeling and predicting student performance than the original KT code (Area Under Curve = 0.610 > 0.568), due to differences in how it estimates parameters.

1 Introduction

Intelligent Tutoring Systems (ITS) derive much of their power from having a student model [3] that describes the learner's proficiencies at various aspects of the domain to be learned. For example, the student model can be used to determine what feedback to give [4] or to have the students practice a particular skill until it is mastered [2]. Unfortunately, assessing student knowledge is difficult because 1) we can only infer student knowledge from observation of student performance, 2) student performance may not be a perfect reflection of student knowledge (e.g. performance is prone to guessing and slipping), and 3) the state of student knowledge changes over time.

Dynamic Bayes Nets (DBNs) [5] address these difficulties in assessing student knowledge by providing a powerful way to represent and reason about uncertainty in time series data [4, 6]. Section 2 demonstrates how DBNs can model

M. Ikeda, K. Ashley, and T.-W. Chan (Eds.): ITS 2006, LNCS 4053, pp. 104–113, 2006.

student knowledge and performance. Unfortunately, constructing DBNs typically requires a complicated coding effort. Section 3 describes BNT-SM, a tool designed to reduce the cost of developing and evaluating Bayesian student models. Section 4 illustrates how to use BNT-SM to simulate Knowledge Tracing, an established technique for student modeling. Section 5 evaluates BNT-SM by comparing it to the original Knowledge Tracing code. Finally, section 6 summarizes BNT-SM's contributions and limitations.

2 Dynamic Bayes Nets and Student Modeling

We will begin by applying DBNs to an example. Suppose we want to assess the probability of a student knowing a skill. Since we cannot read the student's mind, we can only infer the knowledge state from a set of observable events, such as student performance (whether the student applies the skill correctly) and tutor intervention (whether the tutor gives assistance). We might have certain assumptions about how the latter two factors interact with student knowledge. For example, we might know that student performance on a task is affected by student knowledge and moreover, tutor intervention affects both student knowledge (by helping students to learn) and student performance (by scaffolding the student's current attempt without necessarily causing long-term learning). Figure 1 depicts such causal relationships in a conditional independence graph where:

K = student knowledge state (whether the student knows the skill or not)
H = tutor intervention (whether the tutor gives help or not)
C = observed student performance (whether correct or incorrect)

In DBNs, we usually model a latent variable (e.g. the unobserved student knowledge state) as a changing state in time series data. A state is represented as a node in the graph (e.g., K), while a time slice consists of a set of nodes that

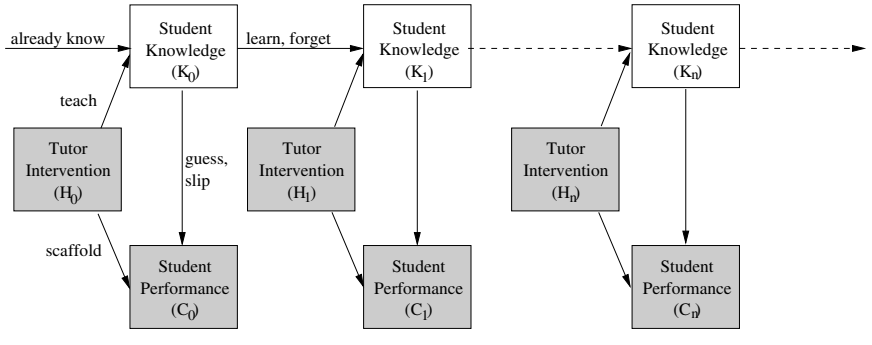

Fig. 1. Dynamic Bayes Net for a student model. Unshaded nodes represent latent variables; shaded nodes represent observed variables.

are modeled at a point of time (e.g. the set $\{K, H, C\}$). Normally, we assume that the parameters (or more precisely, the conditional probability distributions) in a DBN do not change over time; rather, what changes is the value of the latent variable. In the case of student modeling, we assume that student knowledge changes over time, but not the parameters of the model itself. Thanks to this assumption, we can model any number of time slices without estimating an infinite number of parameters.

The conditional independence graph is a graphical representation of the joint probability distribution specified in Equation 1. The joint probability formulation ensures that inferences (estimates of the values of latent variables) are mathematically sound and computationally efficient. This representation makes it easy to include certain kinds of parameters in the model. For example, the guess and slip rates for the vertical edges in Figure 1 are modeled as $P(C = true|K = false)$ and $P(C = false|K = true)$, respectively.

Given the conditional independence graph and the values of the observed variables (evidence), DBNs can infer the values of the latent variables. For example, the DBN uses the observed evidence to compute the posterior probability of the student knowing a skill by applying the Bayes rule as in Equation 2.

$$P(K, L, C) = P(H) * P(K|H) * P(C|K, H) \tag{1}$$

$$P(K = true|H, C) = \frac{P(H, K = true, C)}{P(H, K = true, C) + P(H, K = false, C)} \tag{2}$$

In summary, DBNs provide a powerful way to represent and reason about uncertainty in time series data, and are therefore well-suited to model student knowledge. Indeed, others have applied DBNs to student modeling [4, 6].

3 BNT-SM: Bayes Net Toolkit for Student Modeling

Many general-purpose Bayes net packages have been implemented and distributed. For example, BNT[1], BUGS[2] and GMTK[3] are three popular Bayes net packages that implement different inference and learning algorithms. We decided to use BNT because it supports the most inference algorithms (including junction tree, variable elimination, and Gibbs sampling) and learning algorithms (including both parameter learning and structure learning). More importantly, it supports DBNs, which are essential to student modeling. This software is distributed under the GNU Library General Public License and is implemented using Matlab, a widely used and powerful mathematical software package.

We have extended BNT to reduce the cost to develop and evaluate student models. We call this extension BNT-SM. A researcher can examine the factors that may affect student knowledge simply by specifying the hypothesized causal relationships in an XML specification file and providing empirical data.

[1] BNT is available at http://bnt.sourceforge.net
[2] BUGS is available at http://www.mrc-bsu.cam.ac.uk/bugs/
[3] GMTK is available at http://ssli.ee.washington.edu/~bilmes/gmtk

By hiding most of the coding detail in constructing and training DBNs, BNT-SM lets the researcher focus on the modeling aspect of the problem and quickly experiment with alternative models. BNT-SM reduces the coding overhead by providing a simpler language to specify DBN than the generic BNT code. BNT-SM inputs a student model specified in an XML file. BNT-SM outputs BNT Matlab code to train and evaluate this model. Its XML input is shorter than its BNT output by a factor of at least 5, based on the model that we constructed.

4 Modeling Knowledge Tracing with BNT-SM

To provide a real example of using BNT-SM, we first introduce Knowledge Tracing, a student modeling technique that is related to DBNs.

4.1 Knowledge Tracing

Knowledge Tracing (KT) [2] is an established technique for student modeling and was first used in the ACT Programming Languages Tutor [2]. The goal of KT is to estimate the student's knowledge from his or her observed actions. At each successive opportunity to apply a skill, KT updates its estimated probability that the student knows the skill, based on the skill-specific learning and performance parameters and the observed student performance (evidence).

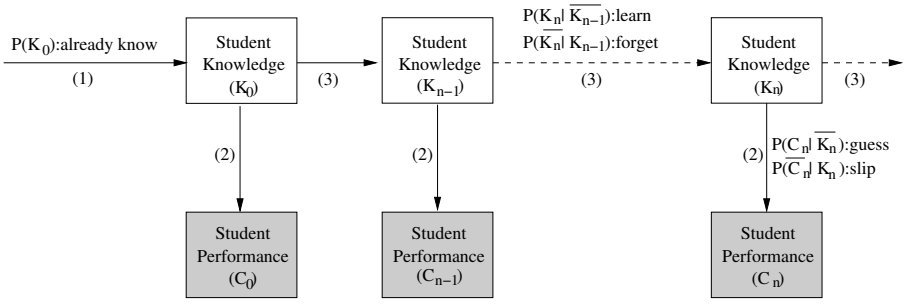

Fig. 2. A DBN that simulates Knowledge Tracing. (1), (2), and (3) refer to identifiers in the XML specification.

Reye [6] showed that KT is special case of a DBN which assumes parameters do not change across time slices. More specifically, the conditional independence graph of KT can be drawn as Figure 2. (We rename the original KT variables $L0$ and t as *already know* and *learn* for consistent naming with other sections.)

4.2 Example of Simulating KT with BNT-SM

Given that KT is a special case of DBNs, we now provide an example of simulating KT with BNT-SM. To use BNT-SM, a researcher needs to follow four steps:

1. Specify the data source in an XML specification[4].
2. Specify the network structure in an XML specification.
3. Specify and initialize parameters in an XML specification.
4. Call `RunBnet.m` in Matlab.

There are two sections in the XML specification file. First, we specify the data source as a tabulated file where each row represents a student's attempt to apply a skill. The columns of the data source file specify the values of the observed variables (e.g. `asr_corr` provides data for the *correct* node) according to empirical data and leave the values of the latent variables (e.g. *pknow*) as a question mark (so that the input and output files follow the same format).

```
user    utterance_start_time  skill  pknow  asr_corr
mTR4-6  2002-08-14_10:22:56   my     ?      1
mTR4-6  2002-08-14_10:22:56   cat    ?      1
...
```

After we have specified the data source, we specify the network structures. As Figure 2 shows, KT has two nodes per time slice. The first node, named *pknow*, represents student knowledge as a latent binary variable. The second node, named *correct*, represents student performance as an observed binary variable. *Pknow* has a `within` time slice connection to *correct* since we assume that student knowledge affects student performance. Moreover, *pknow* has a `between` time slice connection to *pknow* in the next time slice since we model student knowledge as a changing state in the time series data. The following XML (in two column format) instantiate the preceding discussion.

```
<nodes>
 <node>
  <name>    pknow   </name>
  <values>  2       </values>
  <type>    discrete</type>
  <observed>latent  </observed>
  <within>  correct </within>
  <between> pknow   </between>
 </node>
```

```
 <node>
  <name>    correct </name>
  <values>  2       </values>
  <type>    discrete</type>
  <observed>asr_corr</observed>
  <within></within>
  <between></between>
 </node>
</nodes>
```

After we have specified the graphical model, we specify the parameters. For example, we define the parameter named *guess* as the conditional probability $P(C = true|K = false)$ (abbreviated `P2(T|F)`) and initialize it to a random value. `P1`, `P2`, and `P3` denote the conditional probability distributions labeled (1), (2), and (3) in Figure 2. Notice that some parameters are initialized to something like $1 - P1(T)$ such that probability distributions sum to one. Also, KT assumes there is no forgetting once a student learns a skill, so the $forget$ parameter is set to 0.

[4] For the detailed Document Type Definition of our XML specification file, please see http://www.cs.cmu.edu/~listen/BNT-SM.

```
<eclasses>
 <eclass>
  <id>     1 </id>
  <values> 2 </values>
  <eq>     P1(pknow) </eq>
  <cpd>
   <eq>    P1(T)   </eq>
   <init>  rand    </init>
   <param> already know </param>

   <eq>    P1(F)   </eq>
   <init>  1-P1(T) </init>
   <param> null    </param>
  </cpd>
 </eclass>

 <eclass>
  <id>     2 </id>
  <values> 4 </values>
  <eq> P2(correct|pknow) </eq>
  <cpd>
   <eq>    P2(T|F)   </eq>
   <init>  rand      </init>
   <param> guess     </param>

   <eq>    P2(F|T)   </eq>
   <init>  rand      </init>
   <param> slip      </param>

   <eq>    P2(F|F)   </eq>
   <init>  1-P2(T|F) </init>
   <param> null      </param>
   <eq>    P2(T|T)   </eq>
   <init>  1-P2(F|T) </init>
   <param> null      </param>
  </cpd>
 </eclass>

 <eclass>
  <id>     3 </id>
  <values> 4 </values>
  <eq> P3(pknow|pknow) </eq>
  <cpd>
   <eq>    P3(T|F)   </eq>
   <init>  rand      </init>
   <param> learn     </param>

   <eq>    P3(T|T)   </eq>
   <init>  0.000     </init>
   <param> forget    </param>

   <eq>    P3(F|F)   </eq>
   <init>  1-P3(T|F) </init>
   <param> null      </param>

   <eq>    P3(F|T)   </eq>
   <init>  1-P3(F|T) </init>
   <param> null      </param>
  </cpd>
 </eclass>
</eclasses>
```

After we have specified the graphical model in an XML specification file and provided the empirical data, we then call the `RunBnet.m` script in Matlab to start the training (estimating the parameters) and evaluating procedure (estimating the values of the latent variables). The student model's estimation of student knowledge is output to a file similar to the data source file, except that the question marks are replaced by the estimates. The resulting skill-specific parameters are output as follows:

```
skill  #students  #cases  already know  learn  guess  slip
my     179        4490     0.904        0.034  0.605  0.073
cat    178        1579     0.953        0.045  0.274  0.074
...
```

To demonstrate BNT-SM's flexibility and ease of use, we point out that it is easy to extend our two node student model (KT) to the three node student

model in Figure 1. To model the new tutor intervention node and its causal relationships to student knowledge and student performance, it suffices to add just 13 lines to the XML specification file. In contrast, KT code is limited to the simple 2-node model.

5 Evaluation of Model Fit

KT and DBNs estimate model parameters differently. The original KT code treats parameter estimation as a curve-fitting problem and uses a conjugate gradient search method by Powell [7]. In contrast, DBNs typically use Expectation Maximization to estimate the parameter values. We now perform an empirical comparison of the two parameter estimation procedures.

5.1 Data Collection

Our data came from 360 children between six and eight years old who used Project LISTEN's Reading Tutor [8] in the 2002-2003 school year. Over the course of the school year, these students read approximately 1.95 million words (as heard by the automatic speech recognizer). On average, students used the tutor for 8.5 hours.

During a session with the Reading Tutor, the tutor presented one sentence (or fragment) at a time for the student to read aloud. The student's speech was segmented into utterances delimited by silences. Each utterance was processed by the Automatic Speech Recognizer (ASR) and aligned against the sentence. This alignment scored each word of the sentence as either accepted (read correctly) or rejected (misread or omitted). For modeling purposes, this paper treats each English word as a separate skill.

5.2 Evaluation Metric

Since student knowledge is a latent variable that cannot be directly observed, we have no gold standard to compare against. Instead, we used the trained student model to predict whether the ASR would accept or reject a student's next reading of the word. An ROC (Receiver Operating Characteristic) curve measures the performance of a binary classifier by plotting the true positive rate against the false positive rate of varying decision threshold. The area under the ROC curve (AUC) is a reasonable performance metric for classifier systems, assuming no knowledge of the true ratio of misclassification costs.

To evaluate the model fits, we computed the correlation and the AUC between the student model's estimate of the student knowledge and the actual performance (as scored by the ASR).

5.3 Method

To determine which parameter estimation method provided a better model fit to student performance data, we first separated the training and testing set by

splitting the students into two groups. The split was done by sorting the students according to their amount of Reading Tutor usage and alternately assigning students to the two sets. Then, we set up a DBN using BNT-SM to simulate KT and compared it against the curve fitting code of the original KT code.

The curve fitting algorithm is a general purpose routine for the minimization of a function in several variables. The algorithm implemented is a modification of conjugate gradient search by Powell [7]. The legacy code was part of the original KT code[5].

For DBN, we used the Expectation Maximization (EM) algorithm to optimize the data likelihood (i.e. the probability of observing our student performance data). EM is the standard algorithm used in the machine learning community to estimate DBN parameters when the structure is known and there exist latent variables. EM is guaranteed to converge to a local maximum on the likelihood surface. We used the junction tree algorithm for exact inference.

After training with each method, we cross validated by using the resulting student models to predict student performance in the testing set. We then computed the correlation and AUC between predicted student knowledge and observed performance.

5.4 Results

As Table 1 shows, EM outperforms the original KT's curve fitting code on both the correlation and the AUC evaluation metrics. Their two correlation coefficients are reliably different at $p < 0.01$. The values of model fit appear low because we are predicting individual student performance data rather than aggregated performance. It is difficult to predict a student's individual responses. To determine an upper bound on the best possible correlation, we did a cheating experiment that can peek at the future data when it makes a wrong prediction. The cheating experiment further assumed monotonicity constraints. That is, a correct response always increases the student model's estimate of student knowledge, whereas an incorrect response decreases the estimate. The cheating experiment revealed that the maximum correlation of any model that obeyed monotonicity constraints was only 0.5. Note that this maximum performance requires peeking at the data to be predicted and is not necessarily attainable by any actual model.

As Table 2 shows, EM estimates much higher *already know* and lower *learn* parameters than curve fitting. EM's *already know* estimate of 0.68 is more plausible than curve fitting's estimate of 0.33. The 20 most frequent words of English account for about 33% of text words. Since we are using the weighted average for parameter values (weighted by the number of times students encountered the word in the Reading Tutor), an initial knowledge of 0.33 would be comparable to six through ten year old knowing only these 20 words–an unlikely proposition at best. EM's estimate of 0.68 would be comparable to students knowing only the 431 most frequent words in English, which is much more believable.

[5] Source code is courtesy of Albert Corbett, http://www.cs.cmu.edu/~rsbaker/curvefit.tar.gz

Table 1. KT vs DBNs Model Fit

Method	Correlation	AUC
Curve-fitting	0.07	0.568
DBN's EM	0.16	0.610

Table 2. KT vs DBNs Parameters

Method	already know	learn	guess	slip
Curve-fitting	0.33	0.19	0.72	0.07
DBN's EM	0.68	0.14	0.64	0.07

6 Conclusion and Future Work

In summary, this paper describes how DBNs provide a powerful way to model student knowledge in an Intelligent Tutoring System. We introduce a tool called BNT-SM that helps training and evaluating DBNs. The main advantages of BNT-SM are that it is 1) flexible, 2) easy to use, and 3) provides a better model fit and a more plausible model parameters, at least on our data set. We invite the student modeling community to download BNT-SM at http://www.cs.cmu.edu/~listen/BNT-SM.

Currently, BNT-SM handles Bayes nets with any number of latent and observed variables, but the variables are limited to discrete values. Moreover, there exist some restrictions on the possible causal relationships that a researcher may model. For instance, BNT-SM does not allow links that go backward in time or that skip forward past the next time slice. Although generality was a top priority when we designed BNT-SM, we have tested it on only three simple networks (two of them discussed in this paper), so it may have limitations of which we are unaware. Thus, we encourage researchers to try out BNT-SM and provide valuable feedback for us to improve its generality.

As future work, we wish to explore the generality of BNT-SM by providing XML specification files for various popular student models. Moreover, we wish to investigate more complex models such as models with continuous variables, as well as hierarchical models. Hierarchical Bayes nets allow researchers to model more than one level of factors, where a higher level factor (such as overall reading proficiency) affects multiple lower level factors (such as individual words).

Another issue that we would like to address in the future is the computational complexity of training the DBN. Since we're using an exact inference engine (junction tree), training time is rather long. As it currently stands, EM takes 25 hours to estimate parameters for the DBN that simulates KT, while the curve-fitting code in the original KT code takes only 2 hours. There are several ways to speed up training. For instance, we can use approximate inference algorithms that trade off accuracy for efficiency. BNT already implements several approximation algorithms , including Gibbs Sampling and variational methods. Another way to speed up the training process is to formulate the network

using some specialized, well-studied formalism such as Hidden Markov Models, for which more efficient algorithms exist.

Acknowledgement. This work was supported by the National Science Foundation, ITR/IERI Grant No. REC-0326153. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation or the official policies, either expressed or implied, of the sponsors or of the United States Government. We also acknowledge members of Project LISTEN who contributed to the design and development of the Reading Tutor, and the schools that used the tutor.

References

1. Murphy, K.: Bayes Net Toolbox for Matlab. <http://bnt.sourceforge.net> Accessed 2006 March 21.
2. Corbett, A., Anderson, J.: Knowledge tracing: Modeling the acquisition of procedural knowledge. User Modeling and User-Adapted Interaction **4** (1995) 253-278
3. Woolf, B.: Artificial Intelligence in Education. Encyclopedia of Artificial Intelligence. John Wiley & Sons: New York (1992) 434-444
4. Conati, C., Gertner, A., VanLehn, K.: Using Bayesian networks to manage uncertainty in student modeling. User Modeling and User-Adapted Interaction **12** (2002) 371-417
5. Dean, T., Kanazawa, K.: A model for reasoning about persistence and causation. International Journal of Computational Intelligence **5** (1989) 142-150
6. Reye, J.: Student modeling based on belief networks. International Journal of Artificial Intelligence in Education **14** (2004) 1-33
7. Powell, M.: An efficient method for finding the minimum of a function in several variables without calculating derivatives. Computer Journal **7** (1964) 155-162
8. Mostow, J., Aist, G.: Evaluating tutors that listen: An overview of Project LISTEN. Smart Machines in Education. K. Forbus and P. Feltovich, Editors, MIT/AAA Press: Menlo Park, CA. (2001) 169-234