General Features in Knowledge Tracing: Applications to Multiple Subskills, Temporal Item Response Theory, and Expert Knowledge

* José González-Brenes[†], * Yun Huang[♯]
[†]Digital Data, Analytics & Adaptive Learning
Pearson Research & Innovation Network
Philadelphia, PA, USA
jose.gonzalez-brenes@pearson.com

Peter Brusilovsky[♯]
[♯]Intelligent Systems Program
University of Pittsburgh
Pittsburgh, PA, USA
{yuh43, peterb}@pitt.edu

ABSTRACT

Knowledge Tracing is the de-facto standard for inferring student knowledge from performance data. Unfortunately, it does not allow modeling the feature-rich data that is now possible to collect in modern digital learning environments. Because of this, many ad hoc Knowledge Tracing variants have been proposed to model a specific feature of interest. For example, variants have studied the effect of students' individual characteristics, the effect of help in a tutor, and subskills. These ad hoc models are successful for their own specific purpose, but are specified to only model a single specific feature.

We present FAST (Feature Aware Student knowledge Tracing), an efficient, novel method that allows integrating general features into Knowledge Tracing. We demonstrate FAST's flexibility with three examples of feature sets that are relevant to a wide audience. We use features in FAST to model (i) multiple subskill tracing, (ii) a temporal Item Response Model implementation, and (iii) expert knowledge. We present empirical results using data collected from an Intelligent Tutoring System. We report that using features can improve up to 25% in classification performance of the task of predicting student performance. Moreover, for fitting and inferencing, FAST can be 300 times faster than models created in BNT-SM, a toolkit that facilitates the creation of ad hoc Knowledge Tracing variants.

Keywords

knowledge tracing, feature engineering, IRT, subskills

1. INTRODUCTION

Various kinds of e-learning systems, such as Massively Open Online Courses and intelligent tutoring systems, are now producing large amounts of feature-rich data from students solving items at different levels of proficiency over time. To analyze such data, researchers often use Knowledge Tracing [7], a 20-year old method that has become the de-facto standard for inferring student knowledge. Unfortunately, Knowledge Tracing uses only longitudinal performance data and does not permit feature engineering to take advantage of the data that is collected in modern e-learning systems, such as student or item differences. Prior work has focused on ad-hoc modifications to Knowledge Tracing to enable modeling a specific feature of interest. This has led to a plethora of different Knowledge Tracing reformulations for very specific purposes. For example, variants have studied measuring the effect of students' individual characteristics [15, 18, 21, 29], assessing the effect of help in a tutor system [3, 25], controlling for item difficulty [10, 20, 26], and measuring the effect of subskills [28]. Although these ad hoc models are successful for their own specific purpose, they are single-purpose and require considerable effort to build.

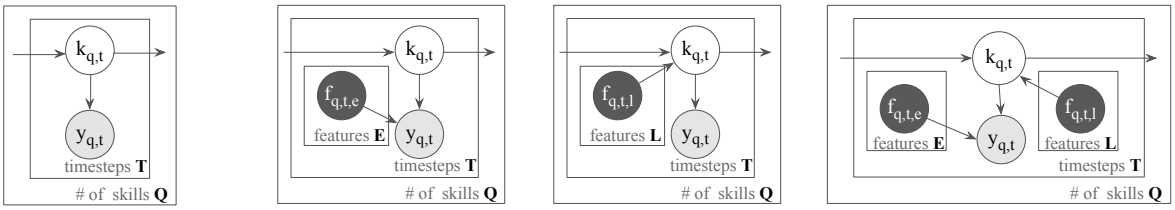
We propose *Feature-Aware Student knowledge Tracing* (FAST), a novel method that allows efficient general features into Knowledge Tracing. We propose FAST as a general model that can use features collected from digital learning environments. The rest of this paper is organized as follows: Section 2 describes the scope of the features FAST is able to model; Section 3 describes the FAST algorithm; Section 4 reports examples of using features with FAST; Section 5 compares FAST's execution time with models created by BNT-SM; Section 6 relates to prior work; Section 7 concludes.

*Both authors contributed equally to the paper.

2. KNOWLEDGE TRACING FAMILY

In this section we define a group of models that we call the Knowledge Tracing Family. We argue that a significant amount of prior work has reinvented models in the Knowledge Tracing Family for very diverse uses, yet their structures when represented as a graphical model are very similar. As we will see, by design, FAST is able to represent all the models in the Knowledge Tracing Family.

Figure 1 uses plate notation to describe the graphical models for the Knowledge Tracing Family of models. In plate notation, the clear nodes represent latent variables; the light gray nodes represent variables that are observed only in training; dark nodes represent variables that are both visible in train-



(a) Original formulation (b) Emission features (c) Transition features (d) Emission and Transition features

Figure 1: Plate diagrams of the Knowledge Tracing Family models.

Table 1: Variants of the Knowledge Tracing model

Feature	Emission	Transition	Both
Student ability		[21]	[18, 29]
Item difficulty	[10, 20]		[26]
Subskills		[28]	
Help		[25]	[3]

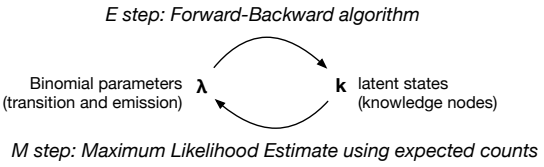


Figure 2: EM algorithm.

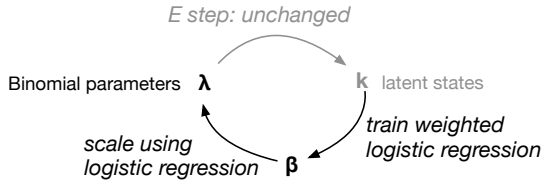


Figure 3: EM with Features algorithm [4].

ing and testing; plates represent repetition of variables.

Figure 1a describes the original Knowledge Tracing formulation. Knowledge Tracing uses Hidden Markov Models [23] to model students' knowledge as latent variables. The binary observation variable ($y_{q,t}$) represents whether the student gets a question correct at the t^{th} learning opportunity of skill q. The binary latent variable ($k_{q,t}$) represents whether the student has learned the skill q at the t^{th} learning opportunity. In the context of Knowledge Tracing, the *transition probabilities* between latent states are often referred to as learning and forgetting probabilities. The *emission probabilities* are commonly referred to as guess and slip probabilities. Figures 1b and 1c describe two common modifications of Knowledge Tracing: adding features to parametrize the emission probabilities ($f_{q,t,e}$), and adding features to parametrize the transition probabilities ($f_{q,t,l}$). In this context, the features nodes are discrete or continuous variables that affect the performance of students or their learning. It is also possible to parametrize both the emission and the transition features. Table 1 summarizes some prior work that has reinvented the same graphical model with a different interpretation of the feature nodes, and are in fact part of the Knowledge Tracing Family.

To assist with the creation of models of the Knowledge Tracing Family, previous research has proposed a dynamic Bayesian network toolkit for student modeling [6]. Unfortunately, extending Knowledge Tracing using dynamic Bayesian networks is tractable only for the simplest models – exact inference on dynamic Bayesian networks is exponential in the number of parents a node has [19]. More specifically, as the number of features increases (**E** in Figure 1b and **L** in Figure 1c), the time and space complexity of the model grows exponentially. We believe that this exponential cost is the reason that although there is a plethora of Knowledge Tracing variants, they are only used for a single purpose. In the next section we describe FAST, a method that is able to generalize all models of the Knowledge Tracing family using a large number of features, but with a complexity that only grows linearly to the number of features.

3. FEATURE-AWARE STUDENT KNOWLEDGE TRACING

FAST extends Knowledge Tracing to allow features in the emissions and transitions using the graphical model structure in Figure 1d. Unlike conventional Knowledge Tracing that uses conditional probability tables for the guess, slip and learning probabilities, FAST uses logistic regression parameters. Conditional probability tables make inference exponential in the number of features, while FAST's performance is only linear in the number of features.

For parameter learning, FAST uses the *Expectation Maximization with Features* algorithm [4] – a recent modification of the original Expectation Maximization (EM) algorithm that is used in Knowledge Tracing. For simplicity, in this paper we focus on only emission features. In preliminary experiments we discovered that emission features outperform transition features, and using both did not yield a statistically significant improvement.

The rest of this section discusses parameter learning. Section 3.1 reviews the EM algorithm used in the Knowledge Tracing Family, and Section 3.2 describes the EM with Features algorithm.

3.1 Expectation Maximization

The EM algorithm is a popular approach to estimate the parameters of Knowledge Tracing. Figure 2 shows the two steps of the algorithm. The "E step", uses the current parameter estimates (λ) for the transition and emission proba-

bilities to infer the probability the student has mastered the skill at each practice opportunity. Inferring mastery can be efficiently computed with the Forward-backward algorithm. The "M step", recomputes the parameter estimates (λ) given the estimated probabilities of mastery computed in the E step. For example, the estimate of the emission parameter of answering y' at latent state k' can be estimated as:

$$\lambda^{y',k'} = p(y = y'|k = k') \quad (1)$$

$$= \frac{\text{expected counts}\,(y = y' \wedge k = k')}{\text{expected counts}\,(k = k')} \quad (2)$$

3.2 Expectation Maximization with Features

The EM with Features algorithm was recently suggested for computational linguistics problems [4]. It uses logistic regression instead of probability tables to model features, which can be discrete or continuous, and are observed during both training and testing. Figure 3 shows the EM with Features algorithm. The E step is unchanged from the original EM algorithm, which gives the probability that the student has mastered the skill at each practice opportunity. However, the M step changes substantially: the parameters λ are now a function of weights $\boldsymbol{\beta}$ and features $\mathbf{f}(t)$. The feature extraction function $\mathbf{f}$ constructs the feature vector $\mathbf{f}(\mathbf{t})$ from the observations (rather than student responses) at the t^{th} time step For example, the emission probability from Equation 1 now is represented with a logistic function:

$$\lambda(\boldsymbol{\beta})^{y',k'} = p(y = y'|k = k'; \boldsymbol{\beta}) \quad (3)$$

$$= \frac{1}{1 + \exp(-\boldsymbol{\beta}^T \cdot \mathbf{f}(t))} \quad (4)$$

We learn $\boldsymbol{\beta}$ from data by training a weighted regularized logistic regression using a gradient-based search algorithm, called LBFGS. Training logistic regression requires a design matrix (a matrix with the explanatory variables). Figure 4 visualizes the design matrix we use. Depending on how features are encoded in the design matrix, FAST allows three different types of features: (i) features that are active only when the student has mastered the skill, (ii) features that are active only when the student has not mastered the skill, or (iii) features that are always be active. The number of features in each type (m_1, m_2, m_3 in Figure 4) can vary. By design, FAST is able to represent the models in the Knowledge Tracing Family. For example, when FAST uses only intercept terms as features for the two levels of mastery, it is equivalent to Knowledge Tracing.

To train the logistic regression, we weight each observation proportionally to the likelihood of the observation being generated from the latent states. Therefore each observation is duplicated during training: the first appearance is weighed by the probability of mastering at current observation; the second appearance is weighted by the probability of not mastering at current observation. This likelihood is calculated during the E step using the forward backward probabilities. More formally, the instance weight is :

$$w_{y',k'} = p(k = k'|\mathbf{Y}; \boldsymbol{\beta}) \quad (5)$$

Then, the Maximum Likelihood estimate $\boldsymbol{\beta}^*$ is:

$$\boldsymbol{\beta}^* = \arg\max_{\boldsymbol{\beta}} \underbrace{\sum_{y,k} w_{y,k} \cdot \log \lambda(\boldsymbol{\beta})^{y,k}}_{\text{data fit}} - \underbrace{\kappa||\boldsymbol{\beta}||_2^2}_{\text{regularization}} \quad (6)$$

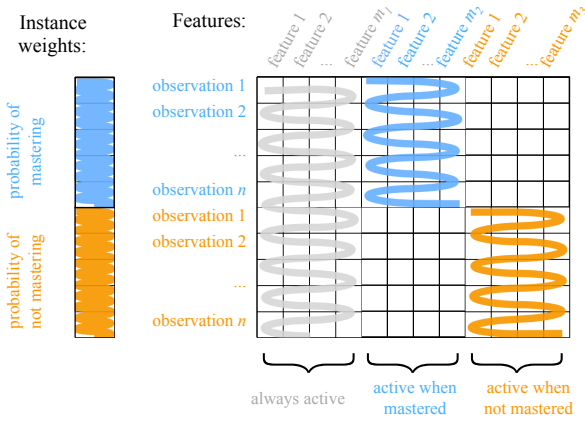


Figure 4: Feature design matrix and instance weights of FAST. During training, observations are duplicated.

where κ is a regularization hyper-parameter to penalize overfitting.

4. EXAMPLES

In this section we describe three case studies that demonstrate FAST's versatility. The rest of this section is organized as follows: Section 4.1 describes our experimental setup; Section 4.2 describes how to model subskills using FAST; Section 4.3 describes how to implement a Temporal Item Response Model using FAST; Section 4.4 describes how to use expert knowledge to improve classification performance.

4.1 Experimental Setup

We used student data collected by an online Java programming learning system called JavaGuide [12]. JavaGuide asks students to answer the value of a variable or the printed output of a parameterized Java program after they have executed the code in their mind. JavaGuide automatically assesses each result as correct or incorrect. The Java programs are instantiated randomly from a template on every attempt. Students can make multiple attempts until they master the template or give up. In total there are 95 different templates.

Experts identified skills and subskills from the templates aided by a Java programming language ontology [11]. Each item is mapped to one of 19 skills, and may use one to eight different subskills. Our dataset was collected during three semesters (Spring 2012 to Spring 2013). It consists of 20,808 observations (from which 6,549 represent the first attempt of answering an item) from 110 students. The dataset is very unbalanced since 70% of attempts are correct (60% of the first attempts are correct).

We evaluated FAST using a popular machine learning metric, the *Area Under the Curve* (AUC) of the Receiver Operating Characteristic (ROC) curve. The AUC is an overall summary of diagnostic accuracy. AUC equals 0.5 when the ROC curve corresponds to random chance and 1.0 for perfect accuracy. We reported two ways of calculating the AUC: (i) overall AUC across all data points of all skills, and (ii) average AUC of the skills as:

$$\text{average AUC} = \sum_s \frac{\text{AUC(skill } s)}{\#\text{ of skills}}$$

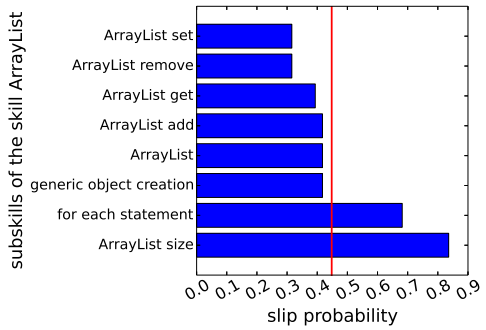


Figure 5: Subskill slip probabilities of the skill ArrayList estimated by FAST. Original Knowledge Tracing estimates the slip probability as 0.45 for the skill.

For the overall AUC, we reported the 95% confidence intervals with an implementation of the bootstrap hypothesis test method[1], a method that corrects for the non-independence of the points of the ROC. To train our classifiers, unless explicitly noted, we modeled each skill independently, and thus we have different model parameters for each of the 19 skills. In the rest of this section we discuss different feature sets we used in FAST.

4.2 Multiple Subskills

The original Knowledge Tracing formulation is designed for fitting a single skill with no subskills. A thorough survey of how prior ad hoc variants of Knowledge Tracing have accounted for multiple subskills can be found elsewhere [28].

To model subskills in FAST we just have to define binary subskill indicator features. In this way, FAST is able to estimate slip and guess probabilities as a function of the subskills present in the practice opportunity. Figure 5 compares the slip probabilities of FAST and Knowledge Tracing for the subskills that are used in the skill ArrayList. We calculate the subskill's slip probability by activating the subskill indicator and intercept in the logistic regression (using Equation 4). The original Knowledge Tracing formulation does not account for differences in subskills, and therefore estimates a single skill slip probability as 0.45. We now evaluate how this improves forecasting performance.

Table 2 compares FAST with different models previously used in the literature. For these experiments, we use a training set of a random sample of 80% of the students, and the rest of the students are used to evaluate the models. The training and testing set do not have overlapping students. We use data on all of the attempts students had to solve an item. We make predictions on all observations of the students in the test set, and evaluate them using overall AUC and mean AUC (when defined). The models we compare are:

- **FAST:** FAST using subskill binary indicator features. We allow FAST to learn different coefficients for guess and slip for each subskill.
- **PFA:** Performance Factors Analysis [22] has been shown to be effective in modeling multiple subskills [8].
- **LR-DBN:** A Knowledge Tracing Family member that uses binary subskill indicators as transition features [28].

[1] `http://www.subcortex.net/research/code/`

Table 2: Overall AUC for multiple subskills experiments.

Model	Overall AUC
FAST	$.74 \pm .01$
PFA	$.73 \pm .01$
LR-DBN	$.71 \pm .01$
KT(single skill)	$.71 \pm .01$
KT(weakest)	$.69 \pm .01$
KT(multiply)	$.62 \pm .02$

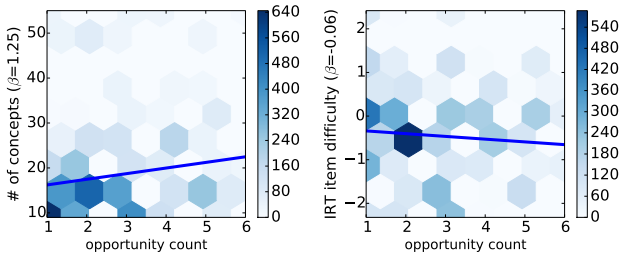
- **KT:** We evaluate different Knowledge Tracing variants:
  - **single skill:** We fit each skill independently (original formulation with no subskills).
  - **weakest:** We fit each subskill independently, and then take the minimum of each subskill's predicted probability of success as the final prediction. We update the knowledge of this weakest subskill by the actual response evidence while we update other subskills by the correct response [8].
  - **multiply:** We fit each subskill independently, and then multiply each subskill's predicted probability of success as the final prediction. We then update the knowledge of each subskill by the same actual response evidence [8].

FAST significantly outperforms all the above Knowledge Tracing variants. In particular, FAST improves the overall AUC of KT(multiply) by about 19% (significant at $p \ll .01$), and outperforms KT(weakest) by over 7% (significant at $p \ll .01$). We hypothesize that FAST's better performance comes from estimating each subskill's responsibility using a logistic regression. This avoids Knowledge Tracing variants' crude assumption that each subskill accounts equally for a correct or incorrect response during the parameter learning. FAST also outperforms the LR-DBN by 4% (significant at $p < .003$), which may indicate parameterizing emission probabilities is better than parameterizing transitions in this dataset. Improving over the original Knowledge Tracing formulation (significant at $p < .002$) suggests that modeling subskills is important. The fact that LR-DBN does not improve over Knowledge Tracing questions its usefulness. We do not find statistically significant differences between FAST and PFA using this feature set in our dataset.

4.3 Temporal Item Response Theory

Knowledge Tracing and classical psychometric paradigms, such as *Item Response Theory* (IRT), treat item difficulty in different ways. The Knowledge Tracing paradigm assumes that all items for practicing a skill have the same difficulty [17]. For example, when a student struggles on some items, the paradigm explains it by assuming there is some subskill(s) that the student has yet to acquire. This paradigm requires a very careful definition of skills and subskills, which may be a very expensive requirement – skill definitions are often done manually by an expert. A cheaper alternative is to discover the skill definitions using data, but such methods are still a relatively new technology [9].

On the other hand, IRT explicitly models item difficulty and student ability. For example, the Rasch model [24], the simplest IRT model, assumes that the correctness of a student's response on an item depends on the student ability and the item difficulty. Unfortunately, IRT models are static, and



(a) Number of concepts (b) Item difficulty

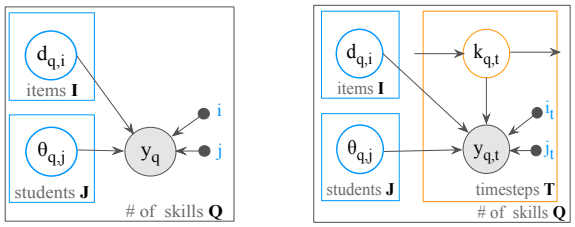
Figure 6: Although experts consider the item complexity increases in latter items, IRT estimates items become easier. Hexagon colors indicate the number of points that fall within the region. Binning is necessary to see overlapping points.

therefore, unlike Knowledge Tracing, do not account for student learning.

Prior Knowledge Tracing variants have been proposed to bridge between the Knowledge Tracing and IRT paradigm. For example, Table 1 summarizes different models that try to account for different student abilities or item difficulties in a learning environment. In addition, Latent-Factor Knowledge Tracing (LFKT) [15], a recent single-purpose specialized graphical model, bridges between both paradigms. FAST takes an alternative approach and models Item Response Theory using feature engineering. Although Rasch Model is typically formulated with latent variables for item and student differences, it can also be estimated using logistic regression with binary variables indicators (sometimes called dummy variables) for each student and each item [**?**, **?**].

In Figures 6a and 6b we show binned scatter plots of the item complexity and the estimated IRT difficulty, respectively. The complexity of an item is defined objectively by experts counting the number of Java concepts used in a question.The item difficulty is estimated by a Rasch model. A higher number of concepts and a higher value of item complexity represent harder items. We run two univariate linear regressions to fit item complexity and difficulty as a function of number of practice opportunities. Experts consider that items practiced later in the tutor are more complex (β=1.25, p<.0005), while IRT estimates that items become easier (β=-0.06, p<.0005). The mismatch between IRT difficulty and item complexity happens because learning is getting confounded with item difficulty. Even though items are becoming harder, students are learning, and thus getting better at them, resulting in the underestimation of item difficulty.

We believe we are the first to show this confounding. It is unclear if prior work is able to deliver the promise of accounting for student learning in the presence of items with different difficulty, particularly because prior work only evaluates on classification accuracy. Aware of this caveat, we use FAST using IRT features. As in previous work, we estimated item difficulties using longitudinal data from students answering items in almost the same order. However, we suggest that future work should calibrate the item difficulty in a controlled experiment first. This would be an easy modification for FAST, as it would only require using a continuous feature (item difficulty) instead of discrete variables (item indicators). This would not be easy in previous work – a new ad hoc model would be required.



(a) Rasch model of IRT (b) FAST using IRT features

Figure 7: Static IRT (Rasch) and temporal IRT models.

Table 3: Overall and average AUC for IRT experiments.

features	static		temporal	
	all	avg.	all	avg
none	.65 ± .03	.50	.67 ± .03	.56
+student	.64 ± .03	.59	.67 ± .03	.60
+item	.73 ± .03	.63	.73 ± .03	.63
+IRT	.76 ± .03	.70	.76 ± .03	.70

Figures 7a and 7b show the plate diagram of the Rasch model and temporal IRT as a Knowledge Tracing Family member. We do not have to change anything in FAST: our temporal IRT definition uses student and item binary indicator features. The probability of getting a correct response y' of student j for item i on skill q, at the t^{th} time step is:

$$p(y'_{q,t}|\boldsymbol{Y}) = \sum_{\substack{l \in \{\text{mastered}, \\ \text{not mastered}\}}} p(k_{q,t} = l|\boldsymbol{Y}) \cdot \text{Rasch}(d_{q,i_t}, \theta_{q,j_t}, c_{q,l}) \quad (7)$$

where $\mathbf{Y}$ is the corresponding observed sequence. Here, the Rasch function is parametrized with item difficulty d, student ability θ and and a bias c that is specific to whether or not the student has mastered the skill. Both Knowledge Tracing and IRT can be recovered from the combined model with different choices of parameter values. For example, when abilities and difficulties are zero, the combined model is equivalent to Knowledge Tracing. When bias terms are the same (i.e., $c_{q,\text{not mastered}} = c_{q,\text{mastered}}$), we get IRT.

Table 3 evaluates FAST using IRT features. To get a better estimate of item difficulty, for these experiments we only use data on the first attempt of a student solving an item. The training set is a random sample of 50% of the students. For students in the test set, we observe the first half of their practice opportunities and make predictions in the second half of their practice opportunities. We compare static models that assume no learning using logistic regression. For the temporal models we use Knowledge Tracing (with no features) or FAST (with features). We experiment with the following feature sets:

- **none**. No features are considered for classifiers.
- **student**. We only use student indicator features.
- **item**. We only use item indicator features.
- **IRT**. We use both item and student indicator features.

FAST using IRT features outperforms the overall AUC of Knowledge Tracing by over 13% (significant at p≪.01) and the mean AUC of Knowledge Tracing by 25%. Using item features in FAST improves the overall AUC of Knowledge Tracing by about 9% (significant at p≪.01) and the mean

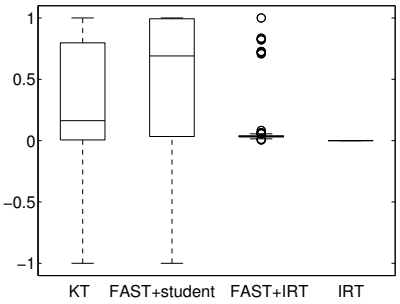


Figure 8: Boxplot of Knowledge Tracing, FAST and IRT's estimates of student learning. The whiskers indicate minimum and maximum values, the lines in the box indicate quartiles. IRT always estimates zero learning.

AUC by 13%. The Follow-up work [**?**] confirms in other datasets that FAST with IRT features is able to significantly outperform Knowledge Tracing.The overall AUC of Knowledge Tracing is not significantly different from that of logistic regression with no features (p>.2). The difference between the mean and overall AUC of these models with no features should be accounted by the expert knowledge introduced by the expert's skill definition. These results coincide with previous work: adding item difficulty and student ability features can greatly increase the performance of Knowledge Tracing. However we do not find any significant differences between the temporal and static models. This may be because the item difficulty is confounded with learning.

Even if Knowledge Tracing with IRT features does not outperform IRT, such a model could be preferable to IRT if it allows modeling student learning. We estimate student learning by subtracting the probability of mastery at the first practice opportunity from the probability of mastery at the last practice opportunity for each student-skill sequence:

$$\text{learning} \equiv p(k_{q,T} = \text{mastered}|\boldsymbol{Y}) - p(k_{q,0} = \text{mastered}|\boldsymbol{Y})$$

Figure 8 shows the box plot of average learning of students across all skills for the models. Both Knowledge Tracing and FAST+student are able to capture student learning from data much better than models that account for item difficulty. In our literature review from Table 1, none of the methods reported both a comparison with IRT and this analysis of learning. We do not know if our results are typical, but they suggest that item difficulty is confounded with learning. Future work should study using FAST with item difficulties calibrated in a controlled experiment to avoid confounding with item difficulties.

4.4 Expert Knowledge

In this section we perform feature engineering to improve student performance prediction. We use features that previous work has found to be useful to predict student performance [10, 20, 22, 26]. More concretely, we demonstrate FAST with three types of features: continuous features that indicate the number of prior practice opportunities where the student answered (i) correctly, and (ii) incorrectly the item template, and (iii) item indicator features. The item indicator features correspond to a binary indicator per item (95 indicator features in total). We use the same experimental setup of Section 4.2, which is a non-overlapping set of students for training and testing, and data from all attempts. Table 4 compares the following models:

Table 4: Overall and average AUC for item practice feature experiments

Model	all	avg.
FAST+item+practice	.77 ± .01	.73
FAST+item	.75 ± .01	.68
FAST+practice	.72 ± .01	.67
PFA	.70 ± .01	.60
KT	.71 ± .01	.58

- **FAST+item+practice** uses item indicator features and the numbers of prior correct and incorrect practices features for each item. The coefficients of item practice performance features can be interpreted as learning rates from correct and incorrect practices of an item.
- **FAST+item** uses item indicator features.
- **FAST+practice** uses only the numbers of prior correct and incorrect practices as features.
- **PFA** (Perfomance Factors Analysis) model uses skill indicator, the numbers of prior correct and incorrect practices feature of the skill as features.
- **KT** is the original Knowledge Tracing without features.

The most predictive model is FAST using item difficulty and prior practice features, which outperforms the overall AUC of KT over 8% (significant at p≪.01) and outperforms PFA by 10% (p≪.01). Its mean AUC also beats KT by 26% and PFA by 22%. The best model outperforms FAST with only item indicator features with an improvement of 3% of overall AUC (significant at p<.005) and 7% of mean AUC, indicating that adding item practice features can provide significant gain over just using item difficulty parameters. When FAST uses only practice features we do not find a significant difference from PFA and Knowledge Tracing in terms of overall AUC (p>.1), but it shows improvement using the mean AUC metric – improving PFA by 12% and Knowledge Tracing by 16%. Future research should investigate whether this discrepancy between the overall AUC and mean AUC is because of the distribution of common and rare skills or because of the quality of the item to skill mapping. Our results suggest that feature engineering can improve the forecasting quality in Knowledge Tracing.

5. EXECUTION TIME

We now study the execution time of learning the parameters and making predictions. As a comparison, we use a popular tool that facilitates the creation of ad hoc Knowledge Tracing variants called BNT-SM [6]. We conduct the experiments on a contemporary laptop (1.8 GHz Intel Core i5 CPU and 4GB RAM). We compare FAST and BNT-SM under two settings: (i) tracing single skills as the standard Knowledge Tracing and (ii) tracing multiple subskills. For the Knowledge Tracing experiment, Figure 9 shows the execution time of both algorithms varying the dataset size, and FAST is about 300 times faster. For the multiple subskill experiment, we compare with LR-DBN, a recent method [28] implemented on BNT-SM. We use the authors' implementation of LR-DBN. LR-DBN takes about 250 minutes while FAST only takes about 44 seconds on 15,500 datapoints. We didn't report results varying dataset since LR-DBN requires much time. The execution time of LR-DBN is comparable to the one reported by LR-DBN authors.

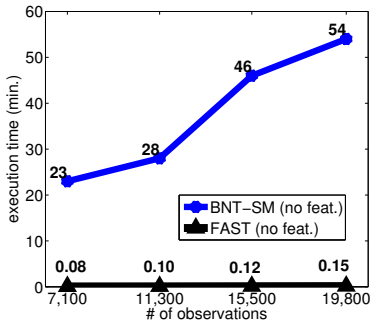


Figure 9: Execution time (in minutes) of FAST and BNT-SM with different sizes of dataset on single skill experiment

In both experiments, FAST's parameter fitting time can be up to 300 times faster — while keeping the same or better performance in terms of overall AUC (significant at $p<.05$). Our implementation of FAST is in Java, while BNT-SM is implemented in Matlab. It is contentious to measure the effect of the programming language in the experiment. However, some informal benchmarks [2] suggest that Matlab is in fact faster for scientific computations.

6. RELATION TO PRIOR WORK

The likelihood functions of FAST and Knowledge Tracing are non-convex. Therefore, parameters discovered might only be local optima, not a global solution. In this paper we only experiment with the Expectation Maximization algorithm, but future work may compare multiple fitting procedures to avoid local solutions [8]. Prior work [2] has also used regression as a post-processing step after Knowledge Tracing. This is different from FAST's approach of jointly training (logistic) regression and Knowledge Tracing. We leave for future work the analysis and comparison of FAST and the post-hoc analysis.

Table 5 compares FAST with models from the literature whether they allow general features, slip and guess probabilities, time, and multiple subskills. For the time dimension we consider whether the models consider recency (R), ordering (O) and learning (L) or none. The Rasch model [24] and Performance Factor Analysis (PFA) [22] use logistic regression and may model arbitrary features. However, Rasch can not account for student learning or multiple subskills. Although PFA is able to fulfill these two cases, it does not consider recency (a correct response following an incorrect response is modeled in the same way as an incorrect response following a correct response), or ordering (a question that was answered incorrectly recently is modeled in the same way as if it were answered incorrectly two weeks ago). Furthermore, PFA does not model slip and guess probabilities.

Knowledge Tracing [7] has a robust mechanism to model time, but lacks the ability to allow arbitrary features and multiple subskills. LR-DBN is a variant of Knowledge Tracing that uses logistic regression [28], yet it is proposed for modeling subskills but not general features. Moreover, in Section 4 we report experiments in which FAST has better predictive performance than LR-DBN. Unlike prior work, FAST is a general method that is able to fulfill a wide range of use cases.

[2] `https://modelingguru.nasa.gov/docs/DOC-1762`

Table 5: Model Comparison

Model	features	slip / guess	time	multiple subskills
FAST	✓	✓	R,O,L	✓
LR-DBN		✓	R,O,L	✓
KT		✓	R,O,L	
PFA	✓		L	✓
Rasch	✓			

7. CONCLUSION

In this paper we identified a family of models, the Knowledge Tracing Family, that have similar graphical model structures. The graphical model structures of the Knowledge Tracing Family have been reinvented multiple times for different applications. We presented FAST as a flexible and efficient method that allows representing all of the models in the Knowledge Tracing Family. FAST uses logistic regression to model general features in Knowledge Tracing. Previous student modeling frameworks [6] for Knowledge Tracing are inefficient because their time and space complexity is exponential in the number of features. FAST is very efficient, and its complexity only grows linearly to the number of features.

Although theoretically FAST should be very similar to the conventional Knowledge Tracing Family implementations, future work may run a more detailed comparison. A limitation of this study is that we did not compare against all of the previous implementations of the Knowledge Tracing Family.

A secondary contribution of this paper is that we identified a problem of prior published work of learning item difficulty from within Knowledge Tracing. The resulting item difficulty estimates are confounded with learning. FAST is also susceptible to this problem, and in future work we will use FAST with item difficulty estimates calibrated with a controlled study. Additionally, future work may study alternative ways of training FAST [4] and discovering an item to skill mapping.

We demonstrated FAST's generality with three use cases that have been shown to be important in prior work: (i) modelling subskills, (ii) incorporating IRT features in Knowledge Tracing, and (iii) using features designed by experts. In our experiments we see improvements of FAST over Knowledge Tracing by up to 13% in mean AUC of skills, and 25% in the overall AUC. When and how feature engineering can assist in student modeling depends on the characteristics of the data and the experience of domain experts. FAST provides high flexibility in utilizing features, and as our studies show, even with simple general features, FAST presents much improvement over Knowledge Tracing. We expect more thorough feature engineering for FAST in the future should provide greater improvement. Moreover, FAST is efficient for model fitting and inferencing — FAST can be 300 times faster than models created in other general purpose student modeling toolkits while keeping the same or better classifier performance.

Acknowledgements

This research is supported by Pearson[3] and the Advanced Distributed Learning Initiative[4]. We also thank the 2013 LearnLab Summer School for enabling the cooperation.

8. REFERENCES

[1] A. Agresti. Computing conditional maximum likelihood estimates for generalized rasch models using simple loglinear models with diagonals parameters. *Scandinavian Journal of Statistics*, pages 63–71, 1993.

[2] R. Baker, A. Corbett, and V. Aleven. More accurate student modeling through contextual estimation of slip and guess probabilities in bayesian knowledge tracing. In B. Woolf, E. Aïmeur, R. Nkambou, and S. Lajoie, editors, *ITS*, volume 5091 of *Lecture Notes in Computer Science*, pages 406–415. Springer, 2008.

[3] J. E. Beck, K.-m. Chang, J. Mostow, and A. Corbett. Does help help? introducing the bayesian evaluation and assessment methodology. In *Intelligent Tutoring Systems*, pages 383–394. Springer, 2008.

[4] T. Berg-Kirkpatrick, A. Bouchard-Côté, J. DeNero, and D. Klein. Painless unsupervised learning with features. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 582–590, 2010.

[5] R. D. Bock and M. Aitkin. Marginal maximum likelihood estimation of item parameters: Application of an em algorithm. *Psychometrika*, 46(4):443–459, 1981.

[6] K.-m. Chang, J. Beck, J. Mostow, and A. Corbett. A bayes net toolkit for student modeling in intelligent tutoring systems. In *Intelligent Tutoring Systems*, pages 104–113. Springer, 2006.

[7] A. Corbett and J. Anderson. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, 4(4):253–278, 1995.

[8] Y. Gong, J. E. Beck, and N. T. Heffernan. Comparing knowledge tracing and performance factor analysis by using multiple model fitting procedures. In *Intelligent Tutoring Systems*, pages 35–44. Springer, 2010.

[9] J. P. González-Brenes and J. Mostow. What and When do Students Learn? Fully Data-Driven Joint Estimation of Cognitive and Student Models. In A. Olney, P. Pavlik, and A. Graesser, editors, *EDM*, pages 236–240, Memphis, TN, 2013.

[10] S. M. Gowda, J. P. Rowe, R. S. J. de Baker, M. Chi, and K. R. Koedinger. Improving models of slipping, guessing, and moment-by-moment learning with estimates of skill difficulty. In *EDM*, pages 199–208, 2011.

[11] R. Hosseini and P. Brusilovsky. Javaparser: A fine-grain concept indexing tool for java problems. In *The First Workshop on AI-supported Education for Computer Science*, page 60, 2013.

[12] I.-H. Hsiao, S. Sosnovsky, and P. Brusilovsky. Guiding students to the right questions: adaptive navigation support in an e-learning system for java programming. *Journal of Computer Assisted Learning*, 26(4):270–283, 2010.

[13] Y. Huang, Y. Xu, and P. Brusilovsky. Doing more with less: Student modeling and performance prediction with reduced content models. In *the 22nd Conference on User Modeling, Adaptation and Personalization*, 2014.

[14] M. Khajah, Y. Huang, J. P. González-Brenes, M. C. Mozer, and P. Brusilovsky. Integrating knowledge tracing and item response theory: A tale of two frameworks. In *In submission*, 2014.

[15] M. Khajah, R. M. Wing, R. V. Lindsey, and M. C. Mozer. Integrating latent-factor and knowledge-tracing models to predict individual differences in learning. In *In submission*, 2014.

[16] K. R. Koedinger, R. S. J. Baker, K. Cunningham, A. Skogsholm, B. Leber, and J. Stamper. In *Handbook of Educational Data Mining*, pages 43–55.

[17] K. R. Koedinger, A. T. Corbett, and C. Perfetti. The knowledge-learning-instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cognitive Science*, 36(5):757–798, 2012.

[18] J. I. Lee and E. Brunskill. The impact on individualizing student models on necessary practice opportunities. In *Proceedings of the 5th International Conference on Educational Data Mining*, pages 118–125, 2012.

[19] K. Murphy et al. The bayes net toolbox for matlab. *Computing science and statistics*, 33(2):1024–1034, 2001.

[20] Z. Pardos and N. Heffernan. $kt - idem$: Introducing item difficulty to the knowledge tracing model. In *User Modeling, Adaption and Personalization*, volume 6787 of *Lecture Notes in Computer Science*. 2011.

[21] Z. A. Pardos and N. T. Heffernan. Modeling individualization in a bayesian networks implementation of knowledge tracing. In *Proceedings of the 18th international conference on User Modeling, Adaptation, and Personalization*.

[22] P. Pavlik, H. Cen, and K. Koedinger. Performance Factors Analysis–A New Alternative to Knowledge Tracing. In *Proceeding of the 2009 conference on Artificial Intelligence in Education: Building Learning Systems that Care: From Knowledge Representation to Affective Modelling*, pages 531–538. IOS Press, 2009.

[23] L. Rabiner and B. Juang. An introduction to Hidden Markov Models. *ASSP Magazine, IEEE*, 3(1):4–16, 1986.

[24] G. Rasch. Probabilistic models for some intelligence and attainment tests. In *Paedagogike Institut, Copenhagen.*, 1960.

[25] M. A. Sao Pedro, R. S. Baker, and J. D. Gobert. Incorporating scaffolding and tutor context into bayesian knowledge tracing to predict inquiry skill acquisition. In *EDM*, pages 185–192, 2013.

[26] S. Schultz and T. Tabor. Revisiting and extending the item difficulty effect model. In *In Proceedings of the 1st Workshop on Massive Open Online Courses at the 16th Annual Conference on AIED*, pages 33–40, Memphis, TN., 2013.

[27] B. D. Wright and G. A. Douglas. Best procedures for sample-free item analysis. *Applied Psychological Measurement*, 1(2):281–295, 1977.

[28] Y. Xu and J. Mostow. Comparison of methods to trace multiple subskills: Is LR-DBN best? In *EDM*, pages 41–48, 2012.

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

[3] `http://researchnetwork.pearson.com/`

[4] `http://www.adlnet.gov/`