Probability Estimation for Online Education System

Machine Learning and Computational Statistics

Felipe N. Ducau, Michael Higgins, Sebastián Brarda
New York University
Project Advisor, Kush Varshney

1 Introduction and Problem Definition

Online education has become one of the main education formats globally. It has many advantages over traditional education like lower costs, scalability and convenience. However it lacks the personalization of traditional education.

It is very challenging to design a system that adapts to the different student skills and weaknesses. The pace at which different students learn varies considerably across different topics and exercise types. By training a machine to understand how well a particular student has learned a particular topic, we could choose which is the most appropriate question to show next in order to optimize her rate of learning. Thus we strive to predict as accurately as possible the probability that a student answers correctly any given question given his/her past performance. If the probability of answering correctly a certain question is very high, the problem would be too easy - showing that problem to the student would be a waste of time. On the other hand, if the problem had a very low probability of success, it would be too difficult, generating frustration without assisting learning.

In this project, we focused on generating calibrated probability estimates for the mentioned task and proposing a personalization system based on these probabilities. The goal was to create

a strong probabilistic model using novel feature creation and leverage this model to create an effective recommendation system.

2 Dataset Description

The dataset comes from the Intelligent Tutoring Systems (ITS), a system used by thousands of algebra and pre-algebra high-school students. Originally, the dataset was used for the KDD cup 2010, a competition (*1*) whose objective was to achieve the minimum Root Mean Squared Error (RMSE) in predicting probability of being successful in the first attempt of the next question.

The exercises of the system are divided into a four stage hierarchy; unit, section, problem and step. For each step that a student attempts to solve there is a row in our dataset that summarizes all of the students attempts at solving the step. This includes the quantity of hints requested, the number of incorrect attempts at the step, the unit and problem to which the step belongs, the knowledge components of the step along with the number of times a student has previously encountered a step that involved each knowledge component. A knowledge component (KC) is a skill, fact or principle that is used to solve the step. There are three different types of KC with 500-600 different skill categories each.

An important thing to mention is that the original dataset was already split into training and test sets in a time series manner. The order is important for our problem, and we cannot split test and training sets randomly. In the original split, one problem was chosen at a random point in time for each student and unit. That problem was later moved to the test set, and all the succeeding problems were discarded.This strategy was used to maintain a balanced test set with problems answered at different points in the learning process.

To keep the order of the dataset and treat it as a time series, we keep the original split framework and created the test set with the last problem of each unit for each student. The process was repeated to create the validation set. The dataset had a total of 8,918,054 rows, out

of which 435,907 became the test set and 412,295 became the validation set.

For a detailed description of the dataset and features, please review Appendix A.

3 Exploratory analysis

The dataset included 1.259.272 unique steps, which belong to 188.368 unique problems. The total number of students was 3310. The number of times a given step was encountered was highly skewed - 75% of the steps occurred only once in the dataset, but these represented only 10% of the rows. The average of their accuracy in terms of students, problems and problem hierarchy are shown in the figures below. As It is evident in the third chart of Figure 1, there are proportionately few good individual problems. Averaging out over a larger sample (unit/section) gives a much more reasonable accuracy curve. For students this is the case as well.

Figure 2 suggests that the number of steps attempted by each student is highly variable. Figure 3 shows the number of times each skill component was encountered over the entire dataset. Note the log scale on the y-axis. This plot strongly influenced our decision to cluster the skill sets, which will be explained in the Feature Engineering section.

4 Baseline Model

The simplest baseline model is to predict based solely on the past performance of the student or the average performance of all the students on each particular problem. We used a more sophisticated method than this approach based on IRT (Item Response Theory) which combines both the probability of a student answering a question correctly given their past performance and the difficulty of the question. In the following derivation we show that IRT can be reduced to logistic regression with parameter λ where $\lambda \cdot \|w\|_2$ takes the place of the prior distribution.

Let $\beta_{p,i}$ represent the difficulty of the problem p in the i-th row of our data set, $\theta_{s,i}$ the skill of student s. In IRT we learn $\beta_{p,i}, \theta_{p,i}$ from $p(y_i = 1|\beta_{p,i}, \theta_{p,i})$, where y_i is the label ($+1$ or -1)

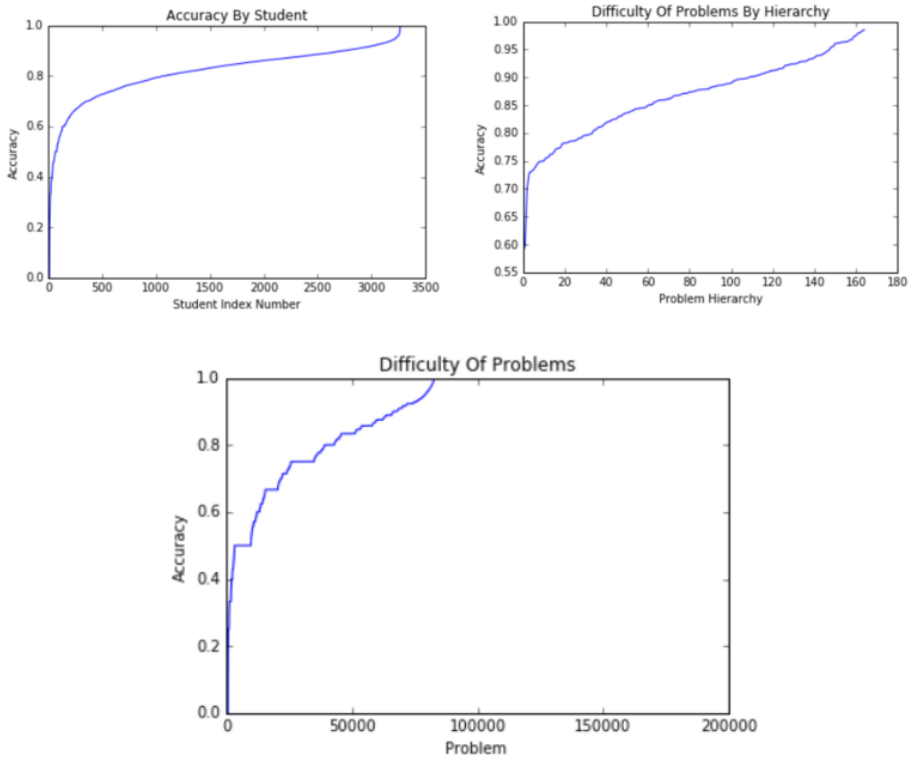

Figure 1: Problems statistics. Accuracy by student, difficulty of the problems by hierarchy and raw difficulty.

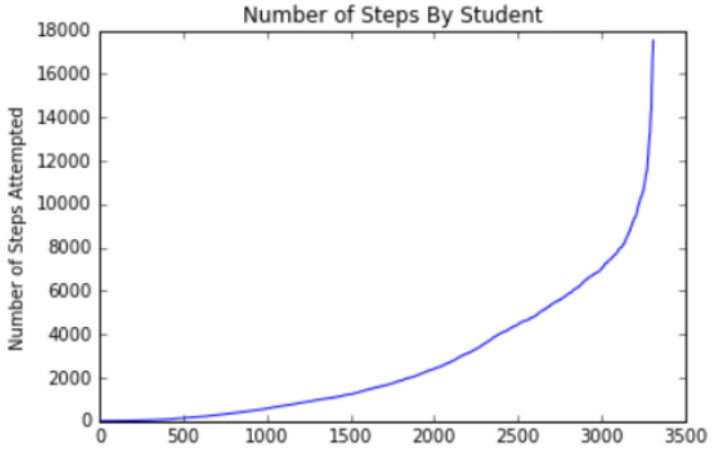

Figure 2: Number of steps attempted by each student.

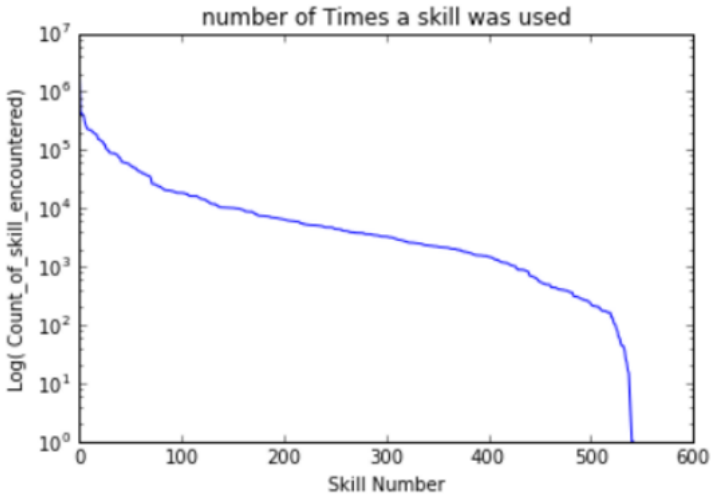

Figure 3: Distribution of the number of times each skill is used. Plot referes to the skills mapping KC Subskills.

of student s on problem p by maximizing the likelihood of the data after assuming a gaussian prior distribution with mean 0. Let P be the total number of problems in the data set and S be the total number of students. Let

$$w = [\theta_1, ..., \theta_S, \beta_1, ..., \beta_P]^T \tag{1}$$

$$x_i = [[(1)_{example\ i\ done\ by\ student\ j}]_{j=1}^{S}, [(1)_{example\ i\ is\ problem\ k}]_{k=1}^{P}]. \tag{2}$$

X_i is a sparse vector indicating which problem and student are involved in example i. X_i has the same dimension as w.

Then our task transforms to

$$\underset{w}{\text{argmax}}\, L(y_i|x_i, w) = \underset{w}{\text{argmax}} \prod_{i=1}^{n} p(w|y_i) \tag{3}$$

$$= \underset{w}{\text{argmax}} \sum_{i=1}^{n} \log \frac{p(y_i|w)p(w)}{p(y_i)} \tag{4}$$

$$= \arg\max_{w} \sum_{i=1}^{n} \log p(y_i|w) + \log p(w) \tag{5}$$

since y_i does not depend on w.

If we assume our prior distribution to be normal and centered around zero, then

$$p(w_k) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-w_k^2}{2\sigma^2}} \tag{6}$$

Which leads to

$$\begin{aligned}
&\underset{w}{\operatorname{argmax}}\, L(y_i|x_i, w) && (7)\\
&= \arg\max_w \sum_{i=1}^{n} \log p(y_i|w) + \sum_{k=1}^{S+P} (-\frac{1}{2}\log 2\pi\sigma^2 - \frac{w_k^2}{2\sigma^2}) && (8)\\
&= \arg\max_w \sum_{i=1}^{n} \log p(y_i|w) - \frac{1}{2\sigma^2}\sum_{k=1}^{S+P} w_k^2 && (9)
\end{aligned}$$

Taking the negative of this expression and using the IRT conditional probality, separating into the cases in which the student got the question correct or incorrect,

$$\begin{aligned}
p(y_i = 1|\theta_{s,i}, \beta_{p,i}) &= g(w \cdot x_i) && (10)\\
&= \frac{1}{1 + e^{-(\theta_{s,i} - \beta_{p,i})}} && (11)\\
&= \frac{1}{1 + e^{-(w \cdot x_i)}} && (12)
\end{aligned}$$

Then,

$$\begin{aligned}
&\underset{w}{\operatorname{argmax}}\, L(y_i|x_i, w)\\
&= \arg\min_w \left\{ -\log\Big(\prod_{correct} \frac{1}{1 + exp(-(w \cdot x_i))} \prod_{incorrect} \Big(1 - \frac{1}{1 + exp(-(w \cdot x_i))}\Big)\Big) + \frac{\lambda}{2}\|w\|_2 \right\} && (13)
\end{aligned}$$

Where $\lambda = \frac{1}{\sigma^2}$. We see that that this expression is equivalent to the regularized logistic regression objective function.

For our baseline model, we performed a machine learning approach, where instead of assuming a variance for the prior distribution, we performed a grid search over λ to find the best fit.

5 Performance Evaluation

Since the main goal of the project to get well calibrated probabilities, we had to restrict our loss functions to the set of Proper Scoring Rules, or any loss function that allowed transformation into a Proper Scoring Rule. We decided to use Log-Loss and RMSE as performance evaluations. Both loss functions are minimized when the probability estimation is optimal. RMSE was chosen as the secondary performance metric, mainly because that was the original evaluation metric for the KDD cup competition.

Finally, each model was compared not only on these metrics but also taking into account their calibration plots.

6 Feature Engineering

Data Preprocessing

Different features received different treatment regarding Null values. There were several instances with no KC skills assigned. For these cases, we replaced the Null values with new features indicating the Unit in which they appeared. The idea behind this was that these problems might be testing a new skill set that was not properly tracked.

The time stamps were very messy - very few instances had valid data in terms of time spent by step. We decided not to use these columns for our project. Since the step names were sometimes repeated across problems, but were different steps, we created a new unique step identifier which was the combination of problem and `step_id`. We also created `unique_problem_id` which were a combination of problems and units.

Another crucial step was to decode the KC skills information that were in the form of strings separated by $\sim\sim$ in the same column. These strings are a sparse representation of the skills in the sense that in each row there only appeared the strings of the skills associated with that

particular step. The number of cumulative previous times that those skills had been shown to a particular student were separated in the same manner in another column. We split those values and created sparse matrices for each of the three KC skills types which each contained between 500 and 600 skills.

Feature Creation

Creating features was one the most challenging part of the project due to the complexity of the calculations and computing requirements. Since the dataset is very high dimensional not only in terms of instances but in terms of features, we could not work with dense representations of the data. The number of distinct steps in the dataset is 1.2 M, so we would not have even been able to run the baseline model with a dense representation. We treated the dataset as a time series, which required taking into consideration the order of the data and the previous observations in our feature creation for each row. We used the Scipy sparse matrices representations for the whole process of creating features and training models.

After creating the sparse matrices of the KC skills, we started using them to create additional features. First we created a sparse matrix (same dimension as KC skills) that counted the cumulative number of correctly answered steps that corresponded to this skill. Second, we created another set of matrices that considered only recent performance for each skill at a certain time-window (i.e. proportion of correctly answered questions with that skill out of the last 5 times that student were exposed to a question with that skill component). The next step was to create new features based on the corrects and incorrects columns. Once again we considered a time window and summed the corrects and incorrects attempts at each `step_id` for the last X attempts that a particular student had. All these different windows would ultimately became hyper parameters of our model.

Another sparse feature created for each instance reflected whether a student answered correctly or incorrectly the previous attempt of that particular `step_id`. This feature would take

value -1 if the previous attempt was incorrect, 1 if it was correct, and 0 if that was the first time that the student was exposed to that question.

For the hints column, we had to think carefully about how to encode it in the training, test and validation datasets. If a student uses a hint for a particular step, then the target variable automatically becomes 0 (it is considered that the first attempt was incorrect). In the original test set of the competition, that variable was not present for obvious reasons (it would be leakage for the instances were it appeared). However, we had the information on how prone each student was to asking for hints. We decided to create a feature that would reflect for each student the average number of hints requested over the training set. By clustering the skill sets (K-means using TFIDF) we created new nonlinear features and reduced our feature dimensionality. We decided to do that after manually inspecting the names/descriptions of each skill since many of them looked very similar. Many skills were encountered a very small amount of times - for these we added relevant information: the previous performance of that student on problems with skill requirements that fell in the same cluster. For example: " Find X - positive slope, Write expression - positive one slope, Write expression - positive slope ," "Find Y- positive slope" were all considered separate skills. After applying clustering they became one skill set. The number of clusters built for each type of Skill became another hyper-parameter of our model.

Finally, since the performance of our baseline model was relatively high, we decided to encode that information as new features. Each parameter of the resulting coefficients vector of the baseline represented the "theta" or "beta" of the student and problem of one particular instance. We encoded these coefficients as two new columns for each instance of our datasets.

7 Models & Training

We ran into substantial computational challenges estimating the models and creating features. Not only training the algorithms took a long time due to the size of the data, but we had to

create several versions of our datasets based on different feature parameters (such as the window sizes), which resulted in very expensive processes. Because of that we decided to start creating our datasets and training our models in the CIMS Courant Compute Servers, which allowed as to leverage the highly parallelizable characteristics of some of the algorithms that we trained. This allowed us to train several models at the same time at a considerably faster pace.

Our first approach after running the baseline model was to train a logistic regression, as it was a natural extension to it. In order to benefit from possible non-linearity of the data, we also decided to try Random Forest Regressor from Scikit learn and XGBoost [1] using log-loss as the loss function to optimize.

We first defined the set of features that are not in a sparse representation (such as `previous_incorrect_step` or `hints_rate`) and one set of sparse features, consisting of the first KC mapping, with a window of 10 for the cumulative features and a clustering of KCs of 75 clusters. With this set of features we conducted a grid search to look for the best hyperparameters for each of the models based on the performance on the validation set. Note that it was not possible to perform cross-validation in our setup because of the temporal structure of the dataset.

As a second step we decided to embed the information generated by our baseline model and collaborative filtering approach (see next section) into the mentioned models. The matrix decomposition procedure used for the collaborative filtering approach gives us a dense n-dimensional representation (latent variables) for each student and each problem along with each term of their cross-product. The baseline encoding approach was to include the estimation of theθ (estimation for the difficulty of a problem) and β (estimation of the proficiency of a student) in our dataset as well. These were the estimated parameters of the baseline model, which we added as two additional columns.

We tried adding these features one at a time and re-training our models. The first thing to

[1] `xgboost.readthedocs.io`

note is that if we used the previous computed hyperparameters, the two expanded datasets (in both versions), led to high overfitting. After recomputing the hyperparameters and regularizing we noted that adding the coefficients from the baseline model yielded an overall better performance than using the latent representation computed from the collaborative filtering. Except for XGBoost, the other models improved performance in both their Log-loss and RMSE.

The last step was to try different window sizes and number of clusters for the knowledge components with little adjustment to the model hyperparameters until finding the best fit by looking at performance on validation set.

The final settings used to train our models were the followings:

- Logistic Regression: Cumulative window of 50, number of clusters for KC's 150, the three KC representation were used plus the θ and β parameters from the baseline model.
- Random Forest: Cumulative window of 100, number of clusters for KC's 100, the three KC representation were plus the θ and β parameters from the baseline model.
- XGBoost: Cumulative window of 50, number of clusters for KC's 75, the three KC representation were used.

8 An alternate approach: Collaborative Filtering

In some sense the education problem can be thought of as the Netflix problem in the sense that different groups of students might be similar in terms of which kind of problems or skills they are good or bad at. We trained a matrix-factorization collaborative filtering model with a binary target with Dato. We used the information of the problems, students and the performance on each of them (without any additional feature). The only hyper parameter of this model was the number of latent variables. The results were surprising: using 8 latent variables, we achieved

a Log-Loss of 0.3254 and a RMSE of 0.3112 . This performance is relatively close to the performance achieved by the other models with the complete set of features.

9 Results

Algorithm	**RMSE on Test**	**Log-loss on Test**
Collaborative Filtering	Train:0.32 Test:0.32	Train:0.35 Test:0.32
Collaborative Filtering - Calibrated	Train: 0.2924 Test:0.3059	Train: 0.2805 Test: 0.3154
Baseline Model - Calibrated	Test: 0.3032	Test: 0.3095
Baseline Model	Train: 0.3104 Test: 0.3025	Train: 0.3130 Test: 0.30765
Logistic Regression	Train: 0.2994 Test: 0.2983	Train: 0.29188 Test: 0.29961
XG Boost	Train: 0.2841 Test: 0.2956	Train: 0.2706 Test: 0.2983
Logistic Regression - Calibrated	**Train: 0.3014** **Test: 0.2955**	**Train: 0.2962** **Test: 0.2946**
Random Forest	**Train: 0.2793** **Test: 0.29**	**Train: 0.2538** **Test: 0.28493**

Table 1. Final results. The best parameters definded through a grid search were chosen for each algorithm.

Despite the fact that our best Random Forest model has slightly better performance, the calibrated Logistic Regression has almost the same performance and it is better calibrated. Additionally, it takes roughly around one tenth of the time to train, so we think that for this particular case it is the most suitable model.

The results do not exhibit the typical train/validation/test performance relationship. For some of our algorithms, performance on test is better than on train. This could be related to the fact that our dataset has an order, and the test set reflects information about students when they have already tried several steps (and hopefully learned). In other words, validation, test and

Dataset	**RMSE**	**Log-Loss**
Train (set1)	*0.32961*	*0.3715702*
Validation(set2) - Original Validation set	**0.32724**	**0.367552**
Train (set2)	**0.327281**	**0.367435**
Validation(set1)	*0.32972*	*0.37168*

train sets might have been generated by a different distribution because they were generated in order.

To confirm this we conducted a small experiment. We created a smaller training set, comparable in size to the validation set. We trained a logistic regression model with the train set as usual, and measured performance both in train and validation. Then we trained the model with the original validation set and evaluated the model from the training set . The results show that performance is quite similar for the same dataset, no matter which is used as the train and which is used as validation, which supports the idea of different generating distributions.

Figure 4 shows the calibration plots for the top performing models of each family.

10 Recommendation system Proposal

After training our model and confirming that probabilities were well calibrated, we wanted to outline how they could be used in a real production system. The chart from Figure 5 chart describes the proposal. For some of the models we used Isotonic Regression for re-calibration of the predicted probabilities.

For a given question of the test set, we estimate the probability of success for that particular student. If the probability of success is too high (i.e. if it is higher than a threshold of 0.8) we move to the next problem of the test set and estimate the probability again. If the probability of success is to low (i.e. lower than a threshold of 0.6) we look for a new problem but only within the subset of problems that require that particular skill. The idea behind this is that if a

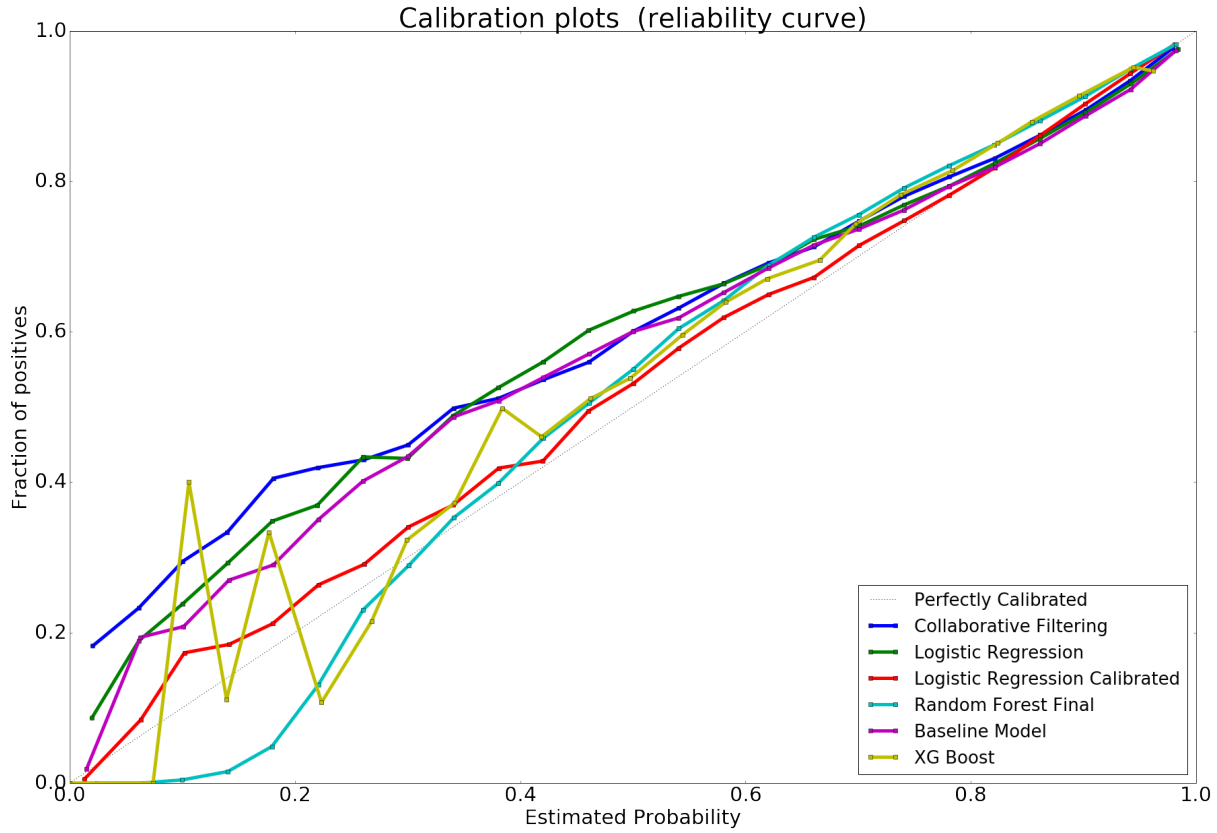

Figure 4: Callibration plots for the models presented in Table 1.

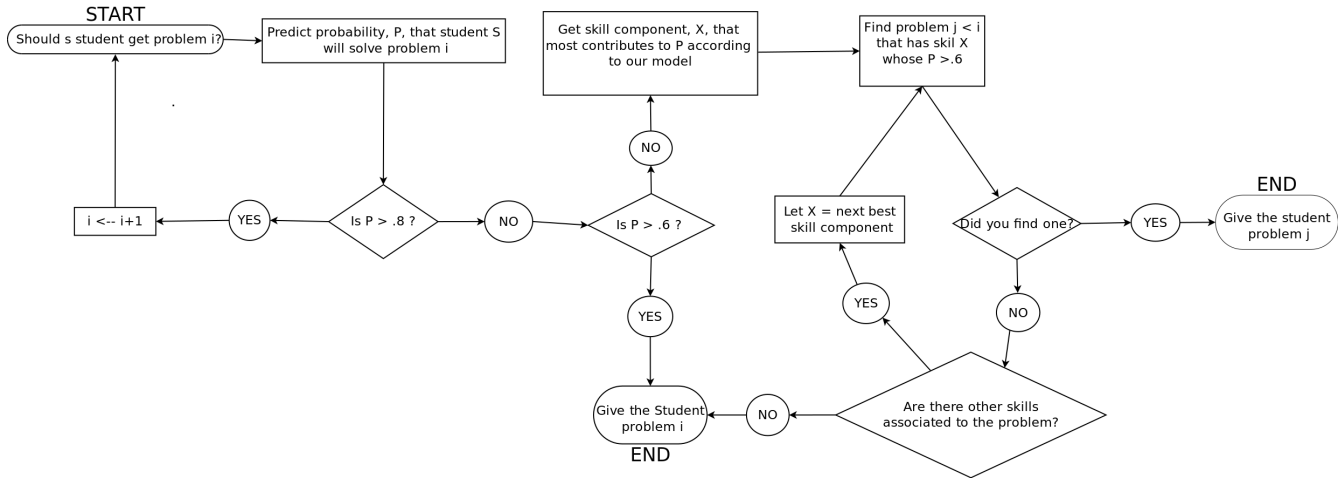

Figure 5: Recomendation system diagram.

student has low probability of succeeding in a problem, he/she needs to develop that skill further. Finally, if the problem is not too difficult and not too easy (within the thresholds window) we show that problem to the user. To provide a rough estimation of the impact that a system like this could provide, lets choose a very conservative upper threshold of 90% of probability in order to consider a problem as "too easy". In our test set 264028 steps out of 453907 which represents a 58.16% are predicted as having probability of success higher than 90% by our best performing Random Forest model. If we are very conservative in the amount of time that each of these steps require, and set it as 15 seconds on average, we would prevent each student from losing $264028 \cdot 15/3310 = 1196.5$ seconds or 20 minutes in total - this time could be invested in solving more challenging problems. If we set the threshold probability in 80% and the average step time to 30 seconds, we would be saving around 52 minutes for each student.

11 Next Steps

- It would be interesting to keep tuning the model and create new features to try to reach the performance achieved by the participants of the KDD cup 2010.
- Conduct a more extensive grid search. Because of the high-dimensionality of our dataset, the amount of hyper parameters and limited time, we could not to search over all possible combinations.
- Try to create our recent performance features based on temporal decaying weights, instead of fixed windows.
- Also, with the knowledge acquired by doing this project, it would be challenging to replicate the same analysis for other education systems datasets.
- Putting into production a recommendation system based on our model is clearly out of the

scope of this project, but would be an excellent way to understand the real performance of our work. It would also allow us gather new data in order to re-train our models.

References and Notes

1. https://pslcdatashop.web.cmu.edu/KDDCup/

2. Feature Engineering and Classifier Ensemble for KDD Cup 2010, http://pslcdatashop.org/KDDCup/workshop/papers/kdd2010ntu.pdf

3. Collaborative Filtering Applied to Educational Data Mining, Journal of Machine Learning Research (2010). Andreas Toscher, Michael Jahrer. http://pslcdatashop.org/KDDCup/workshop/papers/KDDCup2010_Toescher_Jahrer.pdf

4. Using HMMs and bagged decision trees to leverage rich features of user and skill from an intelligent tutoring system dataset. Zachary A.Pardos, Neil T. Heffernan. http://pslcdatashop.org/KDDCup/workshop/papers/pardos_heffernan_KDD_Cup_2010_article.pdf

5. On L2-norm Regularization and the Gaussian Prior, Jason Rennie. http://qwone.com/~jason/writing/l2gaussian.pdf

6. Loss Functions for Binary Class Probability Estimation and Classification: Structure and Applications. Loss Functions for Binary Class Probability Estimation and Classification: Structure and Applications. Andreas Buja, Werner Stuetzle, Yi Shen. http://www-stat.wharton.upenn.edu/~buja/PAPERS/paper-proper-scoring.pdf

7. Obtaining Calibrated probabilities from Boosting. Alexandru Niculescu-Mizil, Richard A. Caruana. 2012. https://arxiv.org/abs/1207.1403

Appendix A: Dataset

Attributes

- Row: the row number
- Anon Student Id: unique, anonymous identifier for a student
- Problem Hierarchy: the hierarchy of curriculum levels containing the problem.
- Problem Name: unique identifier for a problem
- Problem View: the total number of times the student encountered the problem so far.
- Step Name: each problem consists of one or more steps (e.g., "find the area of rectangle ABCD" or "divide both sides of the equation by x"). The step name is unique within each problem, but there may be collisions between different problems, so the only unique identifier for a step is the pair of problem_name and step_name.
- Step Start Time: the starting time of the step. Can be null.
- First Transaction Time: the time of the first transaction toward the step.
- Correct Transaction Time: the time of the correct attempt toward the step, if there was one.
- Step End Time: the time of the last transaction toward the step.
- Step Duration (sec): the elapsed time of the step in seconds, calculated by adding all of the durations for transactions that were attributed to the step. Can be null (if step start time is null).
- Correct Step Duration (sec): the step duration if the first attempt for the step was correct.

- Error Step Duration (sec): the step duration if the first attempt for the step was an error (incorrect attempt or hint request).
- Correct First Attempt: the tutor's evaluation of the student's first attempt on the step1 if correct, 0 if an error.
- Incorrects: total number of incorrect attempts by the student on the step.
- Hints: total number of hints requested by the student for the step.
- Corrects: total correct attempts by the student for the step. (Only increases if the step is encountered more than once.)
- KC (KC Model Name): the identified skills that are used in a problem, where available. A step can have multiple KCs assigned to it. Multiple KCs for a step are separated by $\sim\sim$ (two tildes). Since opportunity describes practice by knowledge component, the corresponding opportunities are similarly separated by $\sim\sim$.
- Opportunity(KC Model Name): a count that increases by one each time the student encounters a step with the listed knowledge component. Steps with multiple KCs will have multiple opportunity numbers separated by $\sim\sim$. Additional KC models, which exist for the challenge data sets, will appear as additional pairs of columns (KC and Opportunity columns for each model).

Training and Test Split

Size of the dataset and missing values

We will use the original training set as our full dataset. Each observation/row corresponds to one interaction (step) of a student with a certain student.

- Number of observations = 8.918.054

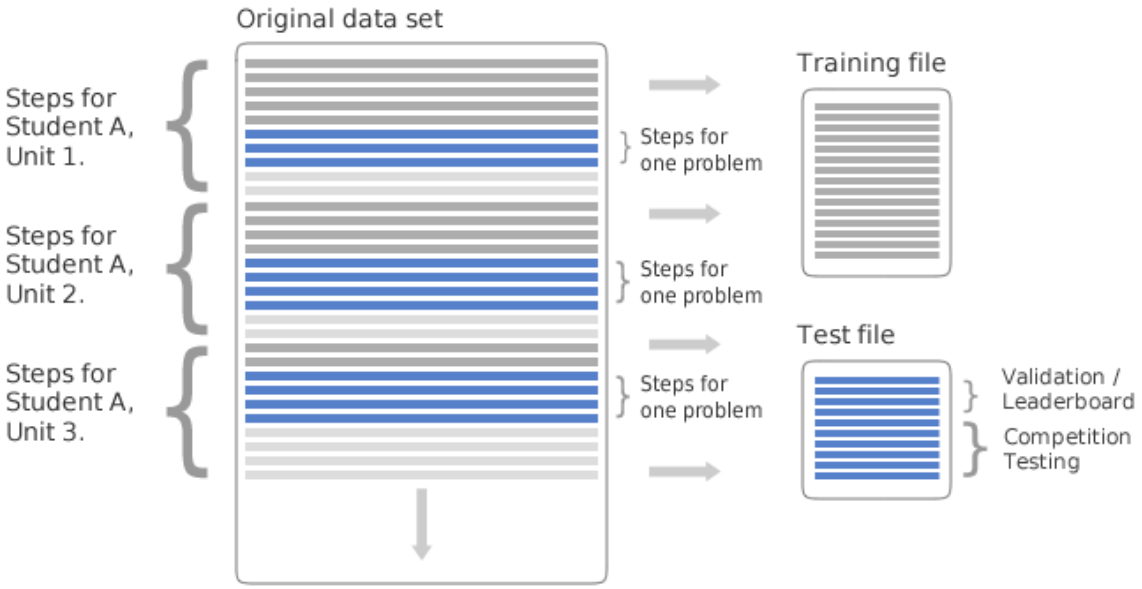

Figure 6: Train/Test split diagram.

Figure 7: Source: `https://pslcdatashop.web.cmu.edu/KDDCup/rules_data_format.jsp`

- Number of features = 23

The following features have Null values. We list the feature, the number of Nulls and the percentage that the Nulls represent in the dataset.

- Step Start Time: 265516, 2.98%
- Correct Transaction Time: 238090, 2.67%
- Step Duration (sec): 442921, 4.97%
- Correct Step Duration (sec): 1641028, 18.4%
- Error Step Duration (sec): 7719947, 86.56%
- KC(SubSkills): 2475917, 26.76%
- Opportunity(SubSkills): 2475917, 26.76%
- KC(KTracedSkills): 4498349, 50.44%
- Opportunity(KTracedSkills): 4498349, 50.44%
- KC(Rules): 322051, 3.6%
- Opportunity(Rules): 322051, 3.6%