The Application of Machine Learning to Predict Correct Student Responses

Juan Rios, Computer Sciences Department, Traditional Masters Program
Shri Shruthi Shridhar, Computer Sciences Department, Traditional Masters Program

Introduction

Overview

The purpose of this project is to predict correct responses from student educational data using machine learning techniques. Recently, NeurIPS held its 2020 Educational Challenge[1]. Contestants were asked to participate in several tasks aiming to use machine learning techniques to predict student responses. This data was collected from the educational online platform Eedi. This paper discusses the algorithms used to predict whether a student answers a question correctly, using the provided data for training. The algorithms will be evaluated using a mix of the following : Prediction accuracy, confusion matrix, cross-validation, F1 score. The aim of the authors is to explore topics relevant to the course and informally participate in this challenge.

Motivation and Significance

To motivate this challenge, the concept of diagnostic questions is introduced. Diagnostic questions are used to identify misconceptions a student might hold regarding a topic. Some of these misconceptions stem from perceived difficulty of the problem, the premise of the problem, etc. By applying the predictive capabilities of machine learning, one can aim to produce sequences of questions that maximize the learning of a student[1]. Both authors have an interest in applying machine learning in the field of education.

Related Works

Most notable related works include the netflix prize, where the dataset consists of users and their movie ratings. One of the models used is an extension of singular value decomposition SVD++. These types of models perform matrix factorization, which in turn allows the prediction of new ratings based on the patterns formed from existing ratings[2].

Matrix factorization is applicable to other similar recommender systems, such as in [3], where authors incorporate implicit feedback from a user rather than explicit feedback in such matrix factorization recommender systems. Other competitions such as [4] ask participants to build recommender systems that recommend music to be added to a playlist based on the music already in the user's playlist.

Methods

Data

The data consists of student-question pairs, where an entry of '1' corresponds to a student answering the question correctly, 0 otherwise. The data is provided in a csv file whose columns take the form [*QuestionId, UserId, AnswerId, AnswerValue, CorrectAnswer, IsCorrect*]. *QuestionId* refers to a set of 27,613 unique questions, *UserId* refers to a set of 118,971 unique students, and *isCorrect* is the aforementioned binary student-answer entry. There are a total of 15,867,850 of these entries [5]. This data can be formulated as a sparse matrix, where the rows represent students, the columns each question, and the matrix values are student responses.

Here, an entry of '1' means a correct response, '-1' incorrect, and '0' as missing. The question formulated here: How can we apply machine learning techniques to predict a students' correctness to a question based on this sparse matrix?

Data Assumption

To pick the appropriate algorithms, one can discuss the underlying data assumption. The assumption is that, although there are 27,613 questions features, we assume that student responses depend on a smaller set of factors. These factors, also known as *latent variables* are able to represent the data in a compressed manner. By taking the total data, the latent variables are learned, and the data is reconstructed into a lower ranked matrix. This low-rank representation of data allows the prediction of missing values, also known as matrix imputation [2][3][6]. This data assumption drives the need for algorithms that can learn these latent variables, and predict the data. The algorithms are discussed in this section.

Baseline

The baseline is to predict a students response according to a majority-vote of whether answers from students are right or wrong. The results of this approach are discussed in the results section.

Generalized Low Rank Model

The Generalized Low Rank Model (GLRM) is an extension to Principal Component Analysis (PCA). This first approach consists of imputing missing values by first factoring the matrix, and then predicting student answer correctness based on a low-rank approximation. Here, GLRM extends PCA by adding quadratic regularization. Consider the objective function to be:

$$minimize \, ||A - XY||_F^2 \tag{1}$$

Where A is the *m* by *n* sparse matrix, XY is a low-rank *r* factorization of A. X is a *m* by *r* matrix where each row of X is a student that has a compressed features space of $r < n$, and Y is an *r* by *n* matrix, where each column is a mapping of each feature. XY is thus a reconstruction of A where Rank(XY) $\leq r$, and *F* indicates the frobenius norm. The quadratic regularization of (1) can be used to reconstruct a matrix that imputes missing values in A to achieve a good fit. The objective function is then turned into:

$$minimize \, ||A - XY||_F^2 + \gamma||X|| + \gamma||Y|| \tag{2}$$

Where $\gamma > 0$ is the regularization term. GLRM uses a truncated matrix singular value decomposition along with quadratic regularization to find a low-rank approximation of A. This model is chosen to input missing values upon the training data.

GLRM Implementation

The training data is first partitioned into an 80/20 training/validation set. The GLRM is available through the h2o package [7] and the implementation in python. An exploratory use of the model tests different rank approximations, varying from 10 to 50. Each column of the training

data is normalized as recommended by [6]. The model is trained for 5 - 400 iterations, which is part of the input, with a regularization term γ = 0.5 as recommended by [6]. One downside of GLRM is that the model does not handle more than 5,000 features. Thus considering the original data with ~27,000 question features proved challenging. One approach taken was to split the original dataset into partitions, meaning each partition would retain the total number of students (rows), but would only train on a fraction of the total questions, approximately ⅕. The assumption is the partitions are large enough to allow latent variable learning. The model is able to run and provide results, at the cost of losing underlying information across the 27,000 questions.
The best results came from training and predicting on a single partition with 150 iterations, and 50 approximation rank. The running time of 13 hours made it impractical to train and predict on the remaining 4 partitions. This single partition is taken as a representative result, assuming the other partitions give similar results. The time instead of was used to implement the Restricted Boltzmann Machine.

Singular Value Decomposition
Singular Value Decomposition algorithm is used to decompose a high dimensional matrix, R into approximate low-dimensional matrices:

$$R = U \Sigma V^T \quad (3)$$

We can use such matrix factorization algorithms to decompose the matrix, pick top features and perform matrix completion to “fill in the blanks” in the original matrix. We created a prediction model using this approach.

Singular Value Decomposition Implementation
Similar to GLRM, the training data is partitioned into 80:20 train/test sets. As mentioned above, there are 27,613 unique questions and 118,971 students. For building the model of SVD based factorization and matrix completion, we first convert the training set into a matrix where the rows represent *UserId* and the columns represent *QuestionId*. Once we have the matrix, we feed a normalized matrix to the SVD algorithm and mention the number of features preferred, k. The algorithm returns U, V^T and Σ. Here, U represents how many users like each question, V^T represents how much each *QuestionId* affects the users and Σ is the diagonal matrix containing weights connecting these 2 matrices. Now, to get a k-ranked low dimensional approximation of R, we multiply the returned matrices as mentioned in (3).

We used the scipy package [8] and implemented the model in python. The results of this approach are evaluated by computing the classification rate on the validation set, and the confusion matrix is calculated. The results are detailed in the results section.

Restricted Boltzmann Machine
A restricted Boltzmann Machine (RBM) is a two-layer undirected graphical model, which can be used on table data [9]. Suppose there are *Q* questions, *S* students. The first layer of the RBM are the visible units, and there are Q of them. Each visible unit takes as input a single student row,

where the entries are {-1, 0 ,1}. The second layer holds *F* hidden (latent) variables, where *F* << Q. The second layer is modeled using a conditional Bernoulli distribution. W is a set of weights that parametrize the interaction between the visible ratings, and the hidden features. W is learned through gradient ascent. Given the observed student responses, the RBM can predict the value for a new question by having the latent features reconstruct the input. The mathematical equations are described in detail in [9].

Restricted Boltzmann Machine Implementation

The implementation of the RBM was guided by the implementation found in [10]. Here, the code is implemented to predict movie ratings from users. The code is adapted to predict on questions answers from students. The code is implemented in python and MATLAB. First, the data is split into 5 sets. Each set contains 80/20 split between training and validation. The first layer is 27,613 units, and the hidden layer sizes explored range from 40 to 150. The training is performed in epochs, and the weights are updated after each batch of 1000 training samples. The recommended earning rate is 0.01 divided by the batch size. The recommended hidden layer size is 100, the recommended number of epochs is 40-50, and the recommended batch size is 1000. These values come from [9]. It was found that the number of layers did not have a significant impact, thus 50 hidden layers are used. Reducing the number of epochs increased the classification rate, and the learning rate was maintained. The chosen parameters to undergo 5-fold cross validation are : 5 epochs, 1000 batch size, 50 hidden layers, and 0.01/batch size learning rate. The results are discussed in the results section.

Results

Table 1 summarizes select results for each of the models described, and the results are discussed in the following subsections

Table 1 Summary of each model implemented, the best results for each model are shown. Indicates the results underwent cross-validation.

Description	Approximation rank	Iterations	Prediction accuracy (%)	F1-Score (%)	Run Time (Hours)
Baseline	-	-	67.2	-	0
Subject Based	-	-	69.47	-	10
GLRM single partition	50	150	67.8	-	13
SVD	50	-	72.5		3
SVD	50 (CV)	-	72.54	79.56	15

SVD	60 (CV)	-	72.63	79.6	15
SVD	75 (CV)	-	72.734	79.66	15
SVD	100 (CV)	-	72.9	79.75	15
SVD	200 (CV)		73.17	80	15
RBM	50 (CV)	5	68.7	-	0.5

Baseline
The baseline predicts a student answer based on the majority of answers for this question. Meaning for a question answered correctly by the majority of students, the baseline will predict '1'. The results of the baseline are shown in Table 1

GLRM
Table 1 shows results for a number of tests cases, where each case varies in approximation rank and iterations. All test runs use a regularization term γ = 0.5 with quadratic regularization. Table 2 shows the confusion matrix best performing case for GLRM.

SVD Based Model
Table 3 shows the confusion matrix for the SVD based cross-validated model with a rank of 200.

RBM
The restricted boltzmann machine predicted values in the range of 0 to 1, but the inputs only included 3 values : -1, 0, and 1. A threshold needs to be determined to give the final prediction. For example if the threshold is 0.5, the predicted values < 0.5 would be mapped to -1, and values $\geq$ 0.5 would map to 1. To determine the threshold, the validation data, which is 20%, is split further in 10% threshold training, and 10% validation. The threshold training data is used to determine the best threshold. This data is visualized in Figure 1.

The mean and median of the training data were considered. However, the best results came from the threshold corresponding to the peaks found in Figure 1. Figure 1 shows two superimposed histograms. The x-axis corresponds to a prediction bin, and the y-axis the number of predictions in that bin. One histogram corresponds to predictions whose true label is '1', and the second histogram corresponds to predictions whose true value is '-1'. The best threshold is the prediction that corresponds to the center peak of the second histogram. The results shown in Table 1 are the average results of 5-fold cross validation, where each fold includes a unique threshold. The averaged confusion matrix is shown in Table 4.

Table 2 The confusion matrix for the best performing test case of GLRM. Each value is divided by the total number of predictions performed.

GLRM	Predicted '1'	Predicted '-1'
Actual '1'	0.56	0.24
Actual '-1'	0.08	0.12

Table 3 The confusion matrix for SVD based model with rank 200. Each value is divided by the total number of predictions performed.

SVD	Predicted '1'	Predicted '-1'
Actual '1'	0.534	0.16
Actual '-1'	0.109	0.197

Table 4 the average confusion matrix results for 5-fold cross validation for the RBM.

RBM	Predicted '1'	Predicted '-1'
Actual '1'	0.52	0.19
Actual '-1'	0.12	0.17

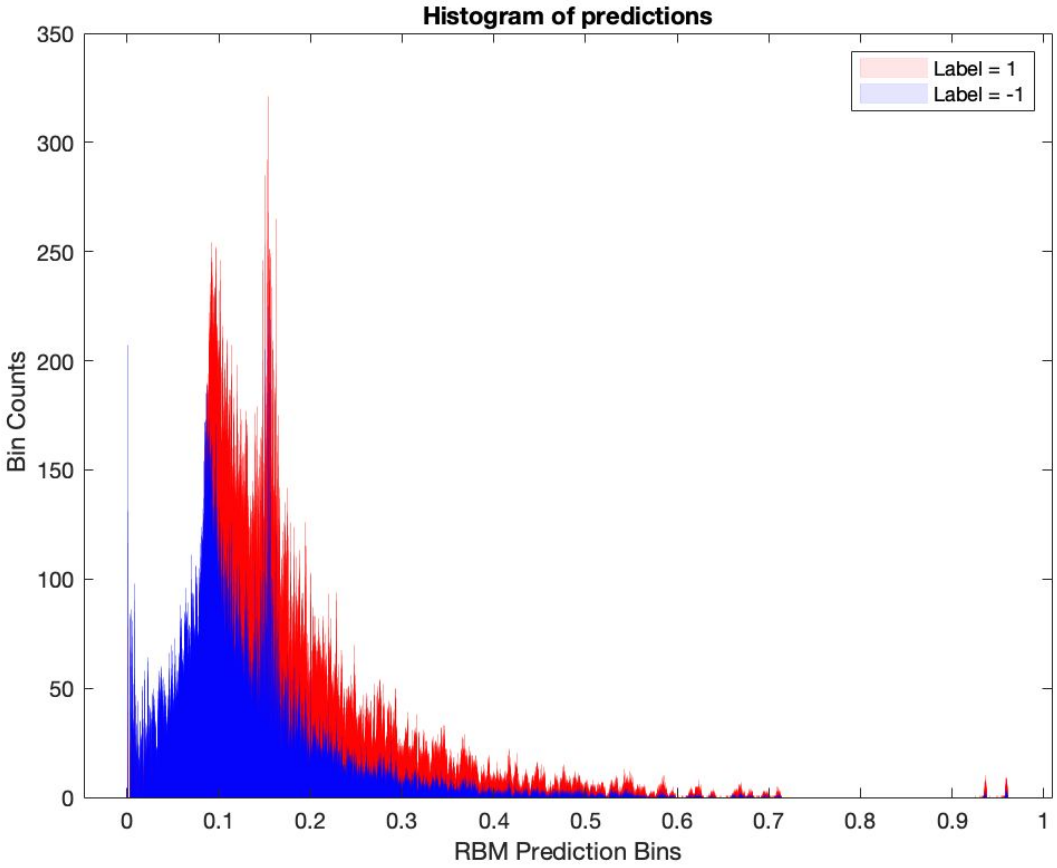

Figure 1 shows the distribution of predictions by the RBM for the first of five folds. The data shown is the 10% validation values that were predicted on, and used to find the threshold. The best threshold occurs at the x-axis bin pertaining to the center peak of the blue histogram. The blue histogram represents predictions whose true value is '-1', and the red histogram predictions whose true value is '1'.

Discussion

GLRM

The results of the GLRM so far prove not much better than the baseline. One likely reason is the inability of the GLRM package to accommodate the full training data matrix, thus information across the 27,000 question-features is lost using the partition approach. Although increasing iterations give better results, there is a diminishing return, and 150 iterations give a good performance/ computational time balance.

From the confusion matrix, the model does poorly predicting when a student misses a question, and predicts too many false negatives. One likely explanation is that the data contains more

data labeled ‘1’ for isCorrect than ‘-1’. A better data to train on would include equal amounts of classes. A second explanation is the handicap imposed by the partitioning method.

Potential improvements include experimenting with other regularization. For example, in this project, Quadratic was solely used, but the framework has other options, such as L2, L1, and sparse. Additionally, the regularization parameters, γ = 0.5 was the sole value. Experimenting with different combinations of this parameter could improve performance. The biggest obstacle is the running time, and better computational resources need be allocated.

SVD-Based Model

The results from the SVD factorization + completion method is better than the baseline. As mentioned in [11], the parameter k (rank of lower dimensional matrix) can be varied to optimize the model in SVD based models. Hence, We evaluated the performance of the SVD based model on different ranks such as 50, 60, 75, 100, 200. The current best model achieves an accuracy of 73.17% leading to an improvement of over 5% compared to the baseline model. The F1-score of this model is 80%. Currently, this model uses 200 features as the preference. We also performed 5-fold cross-validation for this model and other models with different ranks.In the future, we also plan to explore other SVD based factorization methods owing to the better performance of the current model.

From the confusion matrix, we can see that the model does not perform well when a student answers a question wrong and instead predicts it as True (35% of true negatives - .109/.306). We can also notice that the model predicts 22% of the correct answers as wrong (.536/4). Again, one likely reason is the imbalance of different classes in the data. A potential solution is to use stratified sampling or oversample the class with lesser values before training. Better evaluation metrics such as AUROC might help us better understand the performance of the model.

RBM

It was expected the RBM to perform better than SVD, but performed worse than SVD but better than GLRM in terms of predictive accuracy. The confusion matrix is more balanced than GLRM. The true positive rate is on par with SVD, and the true negative rate is better than GLRM. However, RBM does not excel in these aspects. Using the GLRM to predict ‘1’ and SVD to predict ‘-1’. In theory, this would achieve 75% classification rate. The NeurIPS 2020 challenge highest contestant classification rate is 77% [1].

One reason the model did not perform near expectations is the code adapted was a relatively simple implementation of RBM. [9] provides a more custom approach relevant to our data, but the implementation is not as straightforward, and more time is needed to carefully study and implement. A second reason is the need for further experimenting with the number of epochs, the number of hidden layers, and the learning rate to increase performance. Additionally, there is room for a more sophisticated approach towards mapping the RBM predictions to the binary labels, rather than simply mapping based on a threshold. Finally, it might be worth-while to look into multi-layered RBMs.

Conclusion and Future Work:

Through this project, we have tried to predict whether a student has answered a particular question right or wrong based on their previous records and the records of students with similar performances using collaborative filtering methods. We have explored different algorithms such as GLRM, SVD based models, RBM other than baseline models. Currently, SVD based cross-validated model with a rank of 200 performance the best achieving an accuracy of 73.17% and a F1-score of 80%. It underperforms in predicting wrong answers. Based on the performances of the models currently explored, we plan to do the following as part of our future work:

1. Balance the imbalance in training data using stratified sampling or over sampling.
2. Explore other matrix factorization based methods such as SVD++ owing to the success of SVD.
3. Better implement and tune RBM models.
4. Train GLRM models with regularization.
5. Though we implemented the Nearest Neighbours model, we could not get the results due to high testing time (~1 day for 10000 entries). In the future, we plan to better allocate resources and evaluate this model and explore ways to test Nearest Neighbours methods faster.
6. Implement other evaluation metrics such as AUROC for better understanding of model performance.

References

[1] 2020 Conference. [Online]. Available: https://nips.cc/. [Accessed: 11-Oct-2020]. NeuRips https://nips.cc/

[2] R. Bell, Y. Koren and C. Volinsky. The BellKor Solution to the Netflix Prize. 2007.

[3] Xiangnan He, Hanwang Zhang, Min-Yen Kan, Tat-Seng Chua. Fast Matrix Factorization for Online Recommendation with Implicit Feedback. Association for Computing Machinery, 10.1145/2911451.2911489.

[4] https://www.kaggle.com/c/recommender-system-2018-challenge-polimi

[5] Z. Wang, “Diagnostic Questions: The NeurIPS 2020 Education Challenge,” CodaLab, 2020. [Online]. Available: https://competitions.codalab.org/competitions/25449 [Accessed: 11-Oct-2020].

[6] M. Udell, C. Horn, R. Zadeh, and S. Boyd, “Generalized Low Rank Models.” Foundations and Trends in Machine Learning, 2016.

[7]http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/glrm.html

[8] https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.svds.html

[9] R. Salakhutdinov, A. Mnih, and G. Hinton, “Restricted Boltzmann Machines for Collaborative Filtering.” Proceedings of the 24 th International Conference on Machine Learning, Corvalllis, 2007.

[10] https://github.com/srp98/Movie-Recommender-using-RBM/blob/master/Recommender_System.py

[11] https://beckernick.github.io/matrix-factorization-recommender/