CSC311 - Final Assignment

Zelong Liu, Fizzah Mansoor, Harrison Deng

December 3, 2021

Contents

Part I
Predicting Student Correctness

1 K-Nearest Neighbor

Following parts will refer to the following information:

```
import knn as knn
k_vals, val_user_acc, val_item_acc = knn.main(data_path="./data")
```

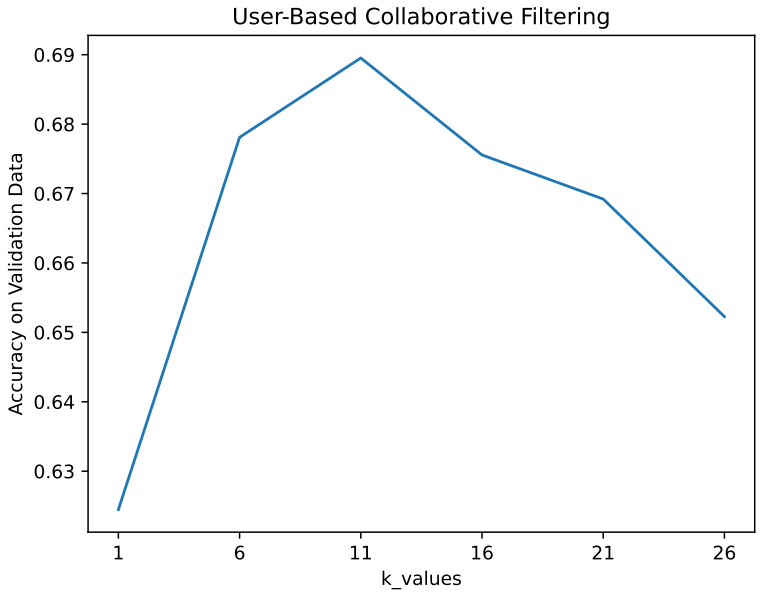
Output:

```
k*: 11 val. acc.: 0.6895286480383855
k*: 21 val. acc.: 0.6922099915325995
User-Based Algorithm Accuracy on Test Data:  0.6841659610499576
Item-Based Algorithm Accuracy on Test Data:  0.6816257408975445
User-based collaborative filtering works better
```

a Complete Main kNN, Plot and Report Accuracy

The implementation of all code is in the `part_a/knn.py` file. Following are plots of accuracy on validation data as a function of $k \in \{1, 6, 11, 16, 21, 26\}$:

```
knn.accuracy_plot(k_vals, val_user_acc, "User-Based Collaborative
↪  Filtering")
```



See accuracies in the data output near the beginning of the question.

b Selecting k*

We selected $k = 11$ for user-based collaborative filtering as this resulted in the highest validation accuracy (refer to data output of `main` function near beginning of question for report on final test accuracy).

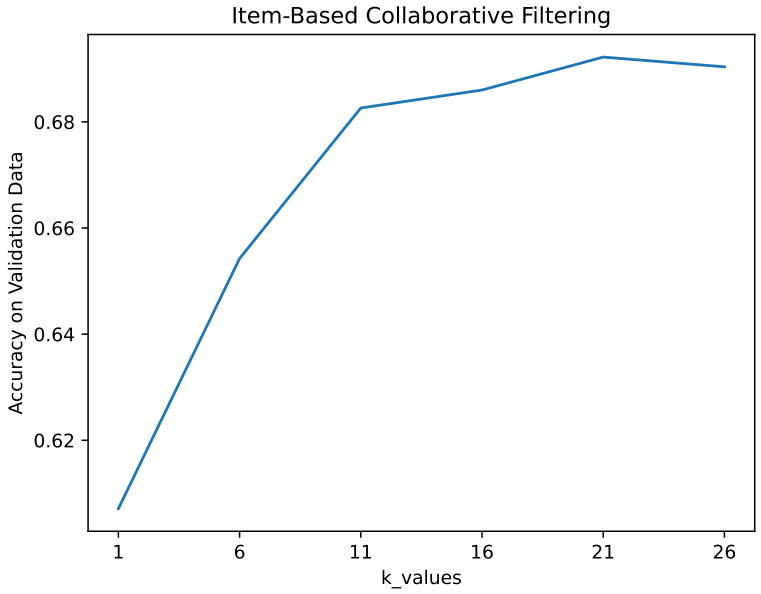
c Implementing Impute by Item

The implementation is in the same file as the user-based version.

Underlying assumption: if answers by certain users to Question A match those of Question B, then As answer correctness corresponding to a specific user matches that of question Y.

Repetition of a) and b) where the data is in the same output box as for user-based collaborative filtering and plot as follows:

```
knn.accuracy_plot(k_vals, val_item_acc, "Item-Based Collaborative
↪  Filtering")
```



d Comparing user and item based Collaborative Filtering

User-Based collaborative filtering performs better on test data. 68.416% accuracy on user-based filtering and 68.162% accuracy on item-based filtering.

e Potential Limitations of kNN in this Context

We can safely assume that there is a high correlation between both question difficulty and student ability on whether or not the question was answered correctly. But, feature importance is not possible for the KNN algorithm (there is no way to define the features which are responsible for the classification), so it will not be able to make accurate inferences based on these two parameters. In the algorithm used in this question, either one of the two parameters (user ability or question difficulty) is focused on, so it has lower validation and test accuracy scores than other algorithms in Part A of this project.

KNN runs slowly. Finding the optimal k-value from the given list of possible k values (1, 6, 11, 21, 26) takes several minutes for each function.

2 Item Response Theory

a Mathematical Derivations for IRT

We are given that $p(c_{ij} = 1|\boldsymbol{\theta}, \boldsymbol{\beta})$. We will assume c_{ij} is a value in $\boldsymbol{C}$ where i and j as coordinates are in set O as defined:

$$O = \{(i, j) : \text{Entry } (i, j) \text{ of matrix } \boldsymbol{C} \text{ and is observed}\}$$

.

Since this c_{ij} is a binary value, we can describe $P(\boldsymbol{C}|\boldsymbol{\theta}, \boldsymbol{\beta})$ with a bernoulli distribution:

$$p(C|\boldsymbol{\theta}, \boldsymbol{\beta}) = \prod_{ij} [\frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)}]^{c_{ij}} [\frac{1}{1 + exp(\theta_i - \beta_j)}]^{(1-c_{ij})}$$

Therefore, our Likelihood function is:

$$L(\boldsymbol{\theta}, \boldsymbol{\beta}) = \prod_{ij} [\frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)}]^{c_{ij}} [\frac{1}{1 + exp(\theta_i - \beta_j)}]^{(1-c_{ij})}$$

Then, apply log to obtain the log-likelihood where N and M are the number of users and questions respectively:

$$\begin{aligned}
L(\boldsymbol{\theta}, \boldsymbol{\beta}) &= \prod_{ij} [\frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)}]^{c_{ij}} [\frac{1}{1 + exp(\theta_i - \beta_j)}]^{(1-c_{ij})} \\
log(L(\boldsymbol{\theta}, \boldsymbol{\beta})) &= \log(\prod_{ij} [\frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)}^{c_{ij}}] [\frac{1}{1 + exp(\theta_i - \beta_j)}^{1-c_{ij}}] \\
&= \sum_{i=1}^{N} \sum_{j=1}^{M} \log([\frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)}^{c_{ij}}] [\frac{1}{1 + exp(\theta_i - \beta_j)}^{1-c_{ij}}]) \\
&= \sum_{i=1}^{N} \sum_{j=1}^{M} c_{ij}((log(exp(\theta_i - \beta_j)) - log(1 + exp(\theta_i - \beta_j))) \\
&+ (1 - c_{ij})(\log(1) - \log(1 + exp(\theta_i - \beta_j))) \\
&= \sum_{i=1}^{N} \sum_{j=1}^{M} [c_{ij}(\theta_i - \beta_j) - \log(\frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)})]
\end{aligned}$$

Then, we solve for the partial derivative with respect to θ_i and β_j respectively:

$$\frac{\delta}{\delta \theta_i} = \sum_{j=1}^{M} [c_{ij} - \frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)}]$$

$$\frac{\delta}{\delta \beta_j} = \sum_{i=1}^{N} [-c_{ij} + \frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)}]$$

b Implementation of IRT

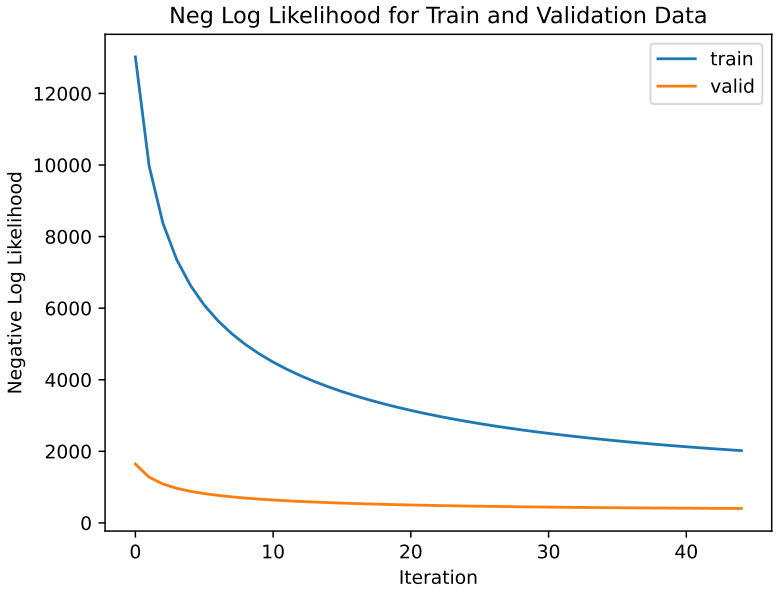
The implementation of IRT is in `part_a/item_response.py`. We chose the hyperparameters α and iterations number by performing multiple combinations of them and seeing which one had the highest validation score (automated, see mentioned code file for this automation). We then manually adjusted the set of tested values and repeated. Doing this a few times resulted in:

```
import item_response as irt
print()
irt_results = irt.main("./data")
```

```
Training with lr of 0.005 and 45 number of iterations.
Final accuracy: 0.7058989556872707
lr*: 0.005 iterations*: 45 val_acc*: 0.7058989556872707
test accuracy: 0.7061812023708721
```

Which is the best result out of the combinations we tried.

The following is the training curve showing training and validation negative log likelihoods as a function of number of iterations:



c Reporting Validation and Test Accuracies

Validation and test accuracies have been calculated in the previous call to the main function. Implementation is in `part_a/item_response.py`.

Our validation accuracy:

```
print(irt_results["val_acc"])
```

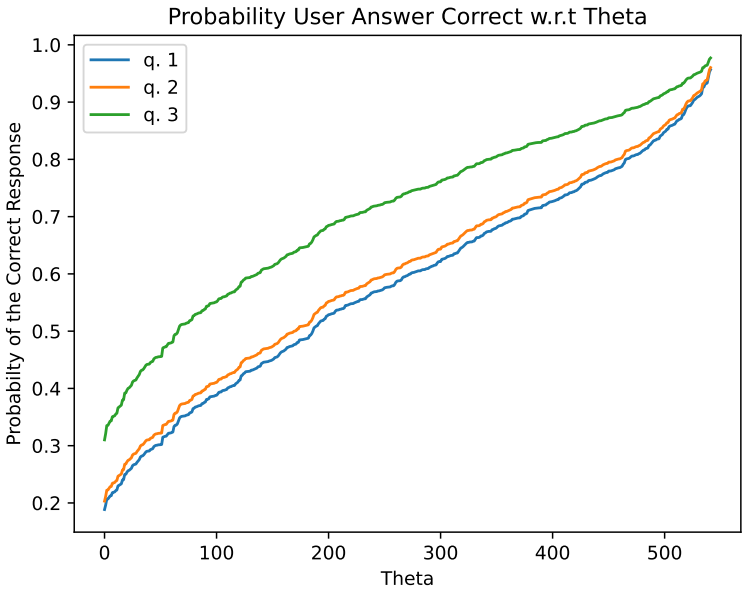
```
0.7058989556872707
```

Test accuracy:

```
print(irt_results["test_acc"])
```

```
0.7061812023708721
```

d Plots of Questions With Respect to θ and β



From this figure, we can see that there seems to be a sigmoidal shape to all three curves. Since the question difficulty, i.e, β_j, doesn't change, it can be considered a constant for one curve. θ_i, the student's ability, is on the x axis and changing. Note that the probability being calculated is the sigmoid of the difference between the θ_i and β_j. Since the curve is not the sigmoid curve without transformations, and β_j is constant, this must mean that θ when sorted, do not increase linearly. We can thus interpret the curve to mean: For a given question that has a theoretical constant difficulty level, as a user's ability increases, their probability of solving a problem correctly also increases, much more drastically at the lower ability levels (steeper slope), and slowing down near the middle (indicated by the decrease in slope) and then increases dramatically again (steep slope again).

3 Neural Networks

a Differences Between ALS and Neural Networks

1. ALS optimizes 2 variable U and Z, neural net optimize one variable W(with gradient descent),

2. ALS is an optimization algorithm that is incorporated as a part of a machine learning algorithm, while the neural network is a machine learning algorithm that that uses optimization algorithms to achieve learning.

3. In neural net, W is used to manipulate x, while in ALS, W,X is being optimized as one variable U .

4. ALS is essentially measuring the difference between target and product of two value(s), the obtain the two value, train matrix need to be pre-processed with SVD, neural network don't need to do this.

5. neural net don't optimize latent Z directly but achieve Z's optimization with W_1 using $g(W_1x)$, where as ALS directly optimize Z.

b Implementing AutoEncoder

This part can be found in `part_a/neural_network.py`.

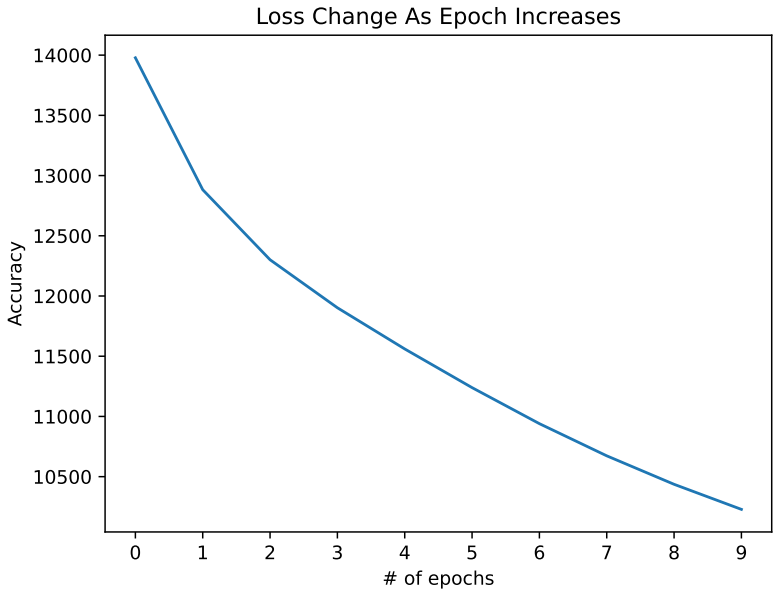
c Tuning and Training NN

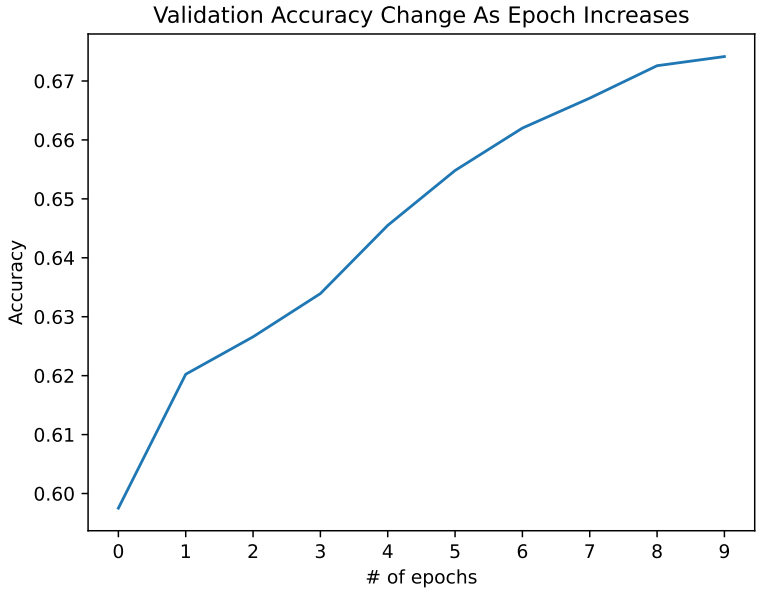
Implementation is in the previously mentioned code file.

From trying various combinations of k where $k \in \{10, 50, 100, 200, 500\}$ and learning rates (α), we found the highest accuracy to be 0.6858594411515665 or 68.5% at $k* = 10$ $\alpha* = 0.05$ and $epoch = 22$.

d Plotting and Reporting

Our final test accuracy was 0.69348010160880611 or around 69.3%. From the learning for that result, the following are the plots generated:





e Implementing L_2 Regularization

L_2 regularization has been implemented in the same code file as the other parts of this question (`part_a/neural_network.py`).

```
lamb=0
Final Validation Accuracy*    0.6858594411515665
Final Test Accuracy*    0.6836014676827548
lamb=0.001
Final Validation Accuracy*    0.6869884278859724
Final Test Accuracy*    0.6785210273779283


(optional extra finding):
but with lamb=0.00025
Final accuracy    0.6848715777589613
Final Test Accuracy*    0.6861416878351679
```

There are improvements on the validation accuracy, but not on the test accuracy.

4 Ensemble

Code for the ensemble is implemented in `part_a/ensembly.py`. We bagged our base neural network, k-Nearest-Neighbor, and Item-Response models with their previously discussed optimal hyper parameters.

```
import ensemble as ensemble
ensemble.evaluate_ensemble(verbosity=1, data_path="./data")
```

Output:

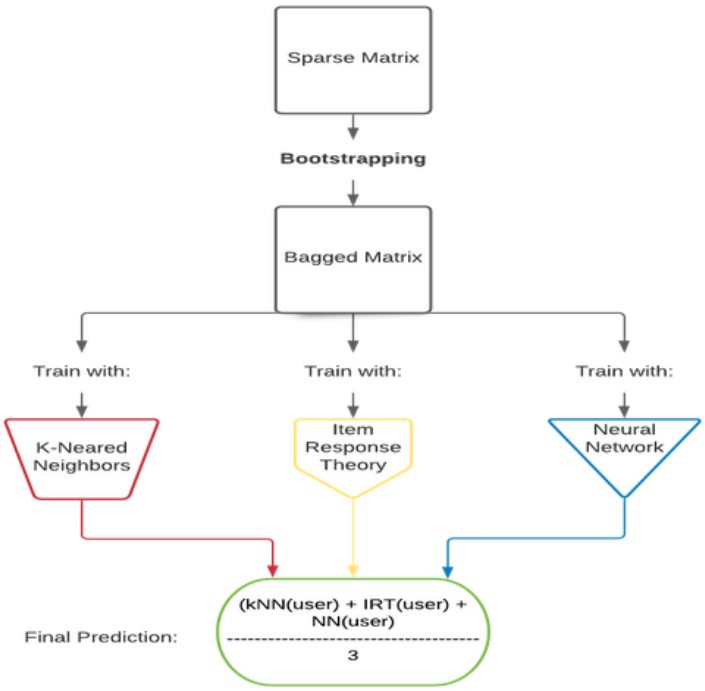
```
Final training results:

Epoch: 40/40, Loss: 3753.432684, Valid acc.: 0.5901778154106689

Final ensemble val. acc: 0.6926333615580017

Final ensemble test acc: 0.6943268416596104
```

We selected 3 base model: kNN, item response model and autoencoder, the bootstrapping phase is done with `numpy.choice`, which samples n user v from the training sparse matrix uniformly, with replacement. To accomadate at least one entree per user and to maintain user order that is required in current version of evaluation functions, we concatenated all of our samples under the original `train_matrix`. We generate 3 versions of bagged train matrix then train each basemodel respectively. For final prediction, we take the average of the predictions given by the 3 trained models, if the average of the value is greater than (or equal to) 0.5, the prediction will be 1, otherwise, it will be 0. Our final result on test data showed the ensemble algorithm performed better than kNN, had very close performance to neural network, and performed a bit less than item response model. To elaborate, we see that the test accuracy for the ensemble is around 69.4% while the neural network's testing accuracy is around 68.6%, 68.4% and IRT is around 70.6%. The average (without weighing) of the latter three is around 69.2% which aligns with the first, i.e, the ensemble of the three. We believe this is because bagging may reduce variance, but will not affect the bias. It would seem that the two under-performing models, kNN and NN, dragged the accuracy down, while IRT kept it up.

Part II
Modifying for Higher Accuracy

5 Formal Description

We are extending the ensemble algorithm (Part A, Question 4). The original ensemble algorithm has 3 phases: bagging (bootstrap aggregating sparse matrix) $\Longrightarrow$ train (training models using the bagged matrix) $\Longrightarrow$ prediction, as demonstrated in the figure below:



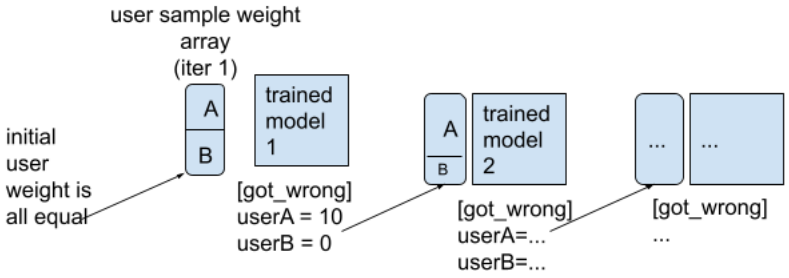
For this project, we gathered inspiration from the AdaBoost algorithm (Freund & Schapire, 1999). AdaBoost is a short form of Adaptive Boosting; The algorithm is adaptive in the sense that subsequent "weak learners" are tweaked in favor of those samples misclassified by previous classifiers. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing, the final model can be proven to converge to a strong learner.

As a departure from the Ensemble algorithm demonstrated in part (A), are removing kNN, modifying the bootstrapping phase (that generates a bagged matrix from the sparse matrix), and prediction phase (within the green box in figure above).

First, our Adaboost Ensemble model does not include k-nn as a weak base

model. We have omitted KNN model from our final adaboost ensemble, and instead replaced it with our previously implemented neural network model, with a differently-bagged matrix from the original neural network model. We chose this implementation because the decision surfaces of k-nn classifiers are typically too stable and any multiples of data points in the bootstrap sample do not shift the 'weight' like in many other models. [insert citation]

The second change is instead of bootstrapping with a uniform probability distribution, we are assigning weights to each user; the weight of each user is determined by the number of questions that the previous model got wrong on this user. The initial bagging is still performed with uniform probability distribution.



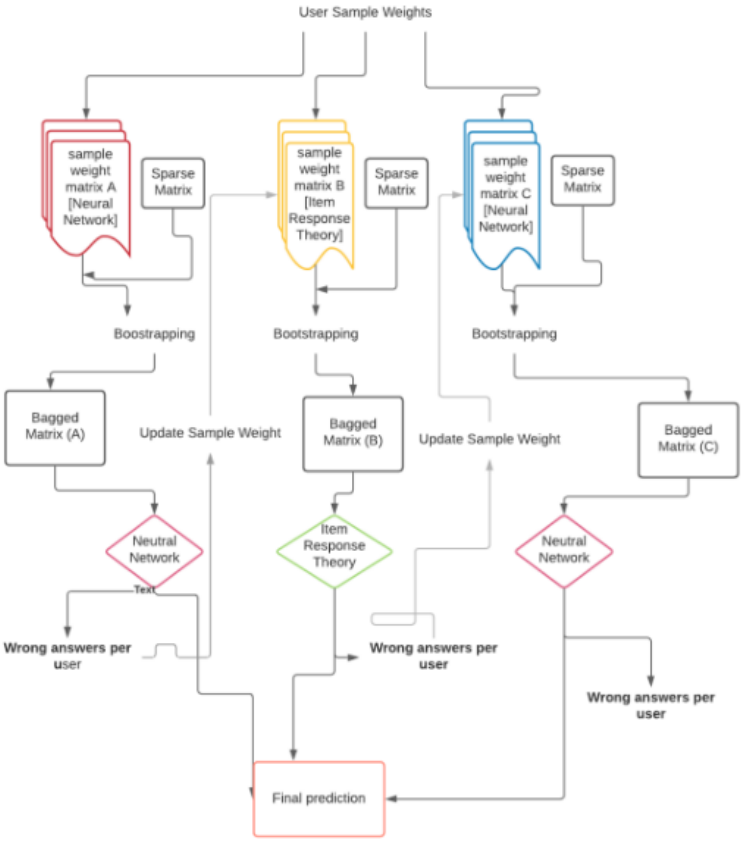
[Insert formula here] In the general case, After the training of a model, we obtain the number of total training entries; number of training entries that the model scored correctly and wrong , per user . We determine a training accuracy

By focusing on the training errors the models made, we expect our model to perform better because this will reduce overfit on users that can be easily predicted, and improve underfit on users that the models struggle to predict.

The third change is during the prediction phase. The prediction phase, instead of generating 3 predictions and taking the average to be the final prediction, we are assigning specific user weights to each model. These weights are for the sampled users for a bagged training set and determined by the performance of the model on said bagged training set. The model with the highest weighting of that specific user will be used for all predictions regarding that user.

By picking a single most-suited model to predict based on which user is doing the question, we expect our model to perform better because this will eliminate the risk of bad performing models having a say on a user that they cant predict well on.

The original adaboost uses a general weight to predict which weight is determined by the accuracy of training data, but we believe predicting weight per specific user can further show us which model is good at predicting which user.



6 Figure or Diagram

7 Comparison or Demonstration