# CSC311 Final Project

Siqi Chen, Zuoyu Wang

## Part A

1. (a) Below is the graph for validation accuracy as a function of k:

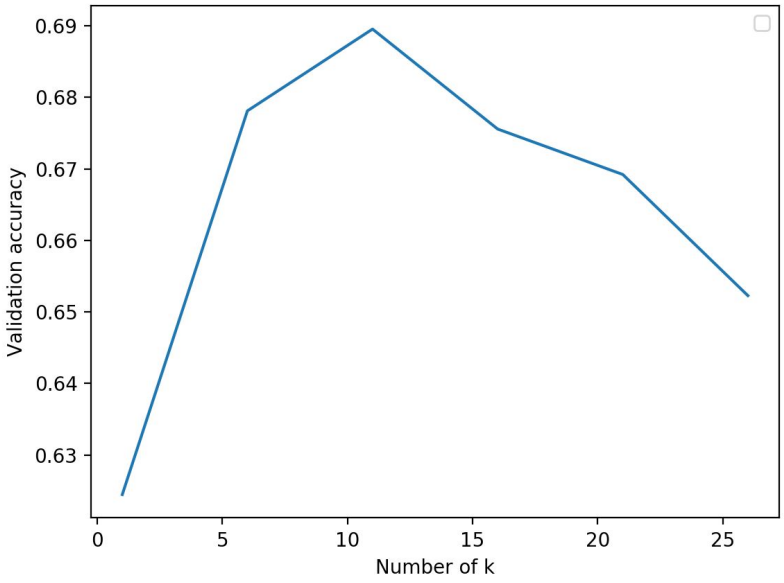


*Figure 1. Knn_impute_by_user validation accuracy vs. number of k*

(b) The k* is 11.
The final test accuracy is 0.6841659610499576.

(c) Below is the graph for validation accuracy as a function of k:

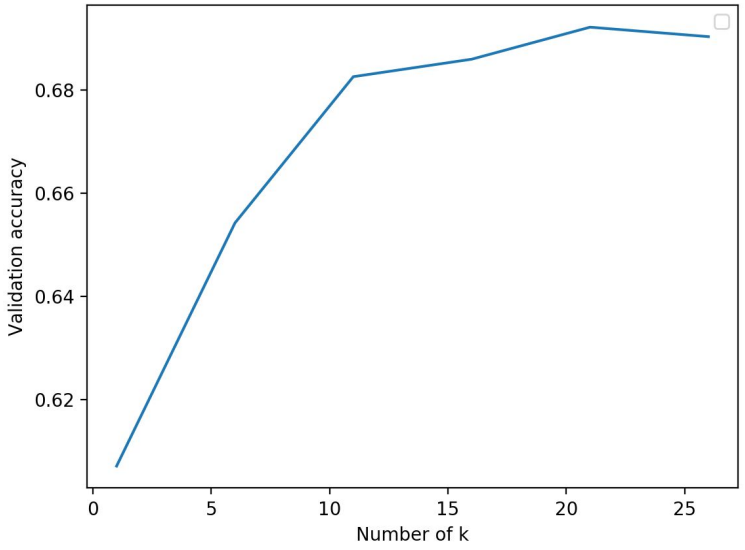


*Figure 2. Knn_impute_by_item validation accuracy vs. number of k*

The k* is 21.
The final test accuracy is 0.6816257408975445
The core underlying assumption: Question A and question B are considered similar if the correctness of them are the same for the same student. Then the correctness on A for a given student matches that of question B.

(d)
Compare the test performance (User: 0.6841659610499576. Item: 0.6816257408975445), the user collaborative filtering performs slightly better.

(e)
- The sparse matrix only considers the correctness to find the similarity, which may not be accurate enough.
- There are lots of NaN values in the sparse matrix, so the closest user or item may not be very similar.
- Both user and item have high dimensionality, which causes objects to become similar. (Curse of Dimensionality for Knn)
- The validation and test time is quite long for KNN.

2. (a)
Below are the log likelihood and derivatives derivation:

$$p(c_{ij}=0\,|\,\theta_i,\beta_j) = 1 - \frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)} = \frac{1}{1+\exp(\theta_i-\beta_j)}$$

$$\ell(\theta,\beta) = \sum_{i=1}^{542}\sum_{j=1}^{1774} c_{ij}\log(\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}) + (1-c_{ij})\log\frac{1}{1+\exp(\theta_i-\beta_j)}$$

$$= \sum_{i=1}^{542}\sum_{j=1}^{1774}\left(c_{ij}(\theta_i-\beta_j) - c_{ij}\log(1+\exp(\theta_i-\beta_j)) - \log(1+\exp(\theta_i-\beta_j)) + c_{ij}\log(1+\exp(\theta_i-\beta_j))\right)$$

$$= \sum_{i=1}^{542}\sum_{j=1}^{1774}\left(c_{ij}(\theta_i-\beta_j) - \log(1+\exp(\theta_i-\beta_j))\right)$$

$$= \sum_{i=1}^{542}\sum_{j=1}^{1774} c_{ij}(\theta_i-\beta_j) - \sum_{i=1}^{542}\sum_{j=1}^{1774}\left(\log(1+\exp(\theta_i-\beta_j))\right)$$

$$\frac{\partial\ell(\theta,\beta)}{\partial\theta_i} = \sum_{j=1}^{1774}\left(c_{ij} - \frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}\right) = \sum_{j=1}^{1774} c_{ij} - \sum_{j=1}^{1774}\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}$$

$$\frac{\partial\ell(\theta,\beta)}{\partial\beta_i} = \sum_{i=1}^{542}\left(-c_{ij} + \frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}\right) = -\sum_{i=1}^{542} c_{ij} + \sum_{i=1}^{542}\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}$$

(* When using the sparse matrix, we will ignore all nan entries when doing the summations.)

(b)
The learning rate is 0.01 with 12 numbers of iteration.

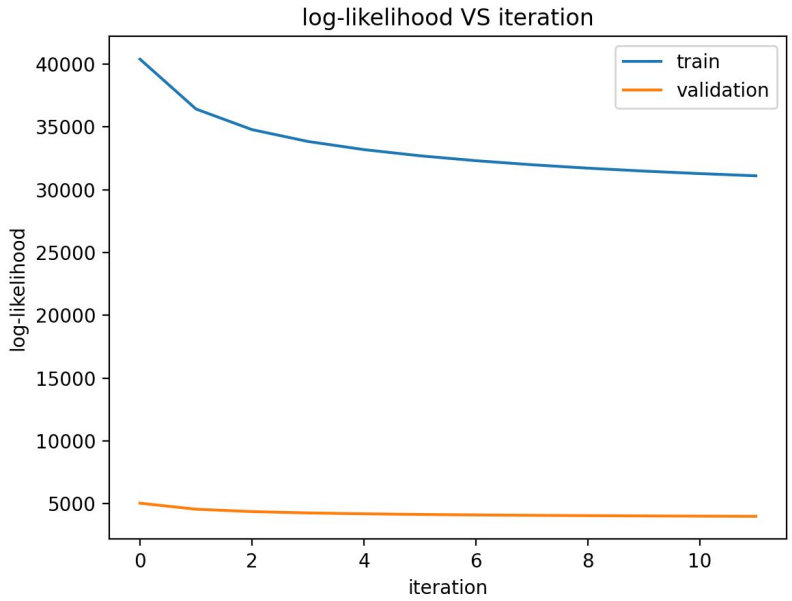


*Figure 3. Negative log-likelihood vs. number of iteration*

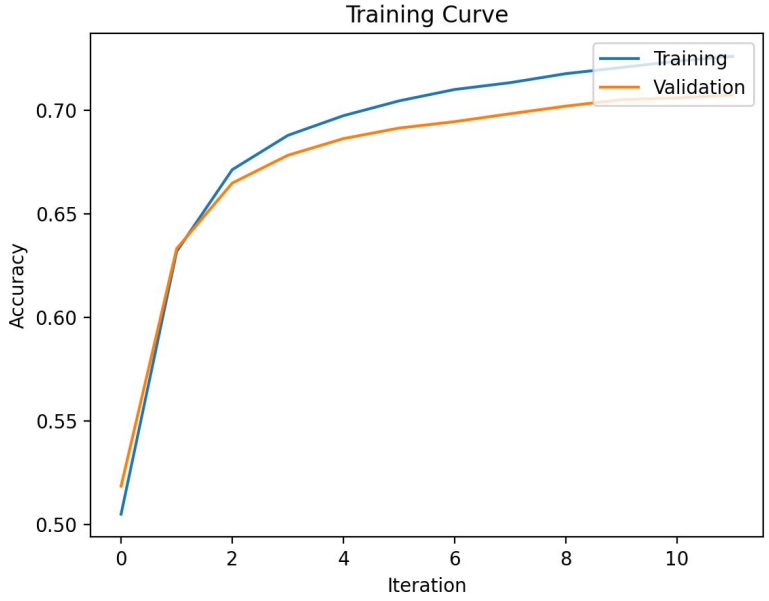


*Figure 4. Accuracy vs. number of iteration*

Both training and validation negative log-likelihood decreases, and their accuracy both increases as the number of iterations increases.

(c)
The final validation accuracy = 0.7029353655094552.
The final test accuracy = 0.7013830087496472.

(d) Below are the five curves for five selected questions:

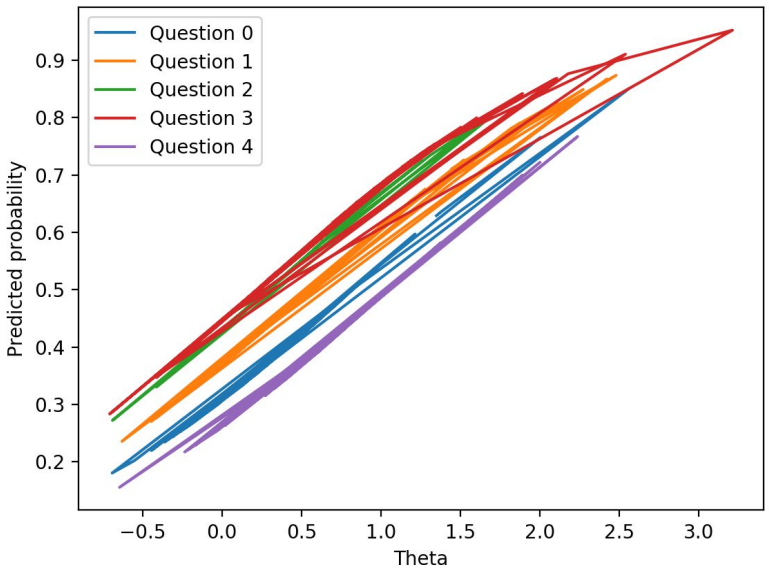


*Figure 5. Probability vs theta for five questions*

For different question j, the probability of the correct response increases and approaches one as theta increases. This demonstrates that the probability for correctly answering one question j will be increased and close to 100% as the ability of the student increases.

3.ii Neural Networks

(a)

1. Neural network is more powerful and expressive since it can handle nonlinearities, while ALS could not.
2. Neural network is hard to train when having multiple hidden layers, while ALS computes very fast (matrix computation).
3. ALS fits two parameters alternatively, while the Neural network only fits one parameter once for each layer.

(c) When number of epochs = 8, learning rate = 0.05, $\lambda = 0$,
chosen k* = 50 and the highest validation accuracy = 0.6844482077335591

(d) Chosen k* = 50, Test Accuracy = 0.6830369743155518

Below is the plot which reports how the training and validation objectives changes as a function of epoch:

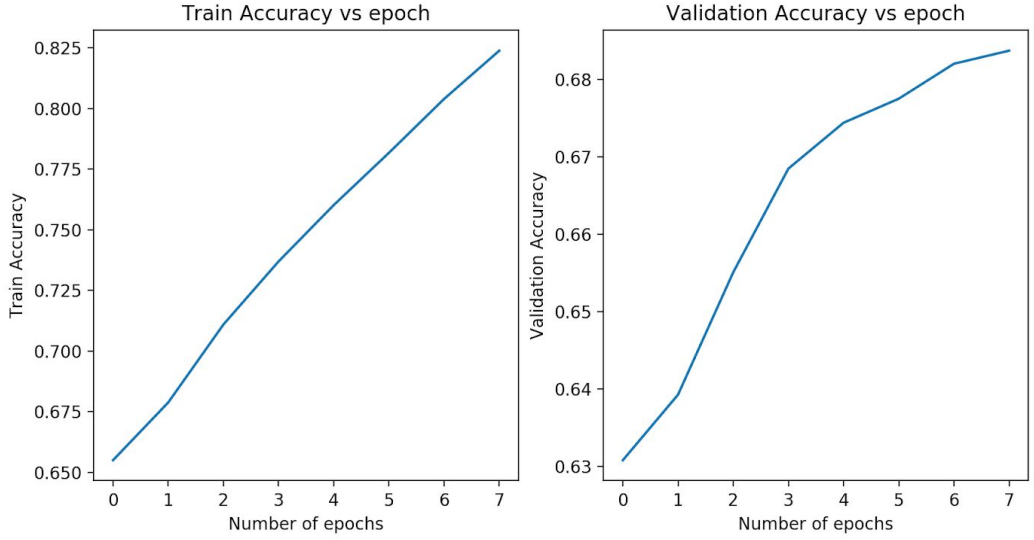


*Figure 6. Accuracy vs epochs*

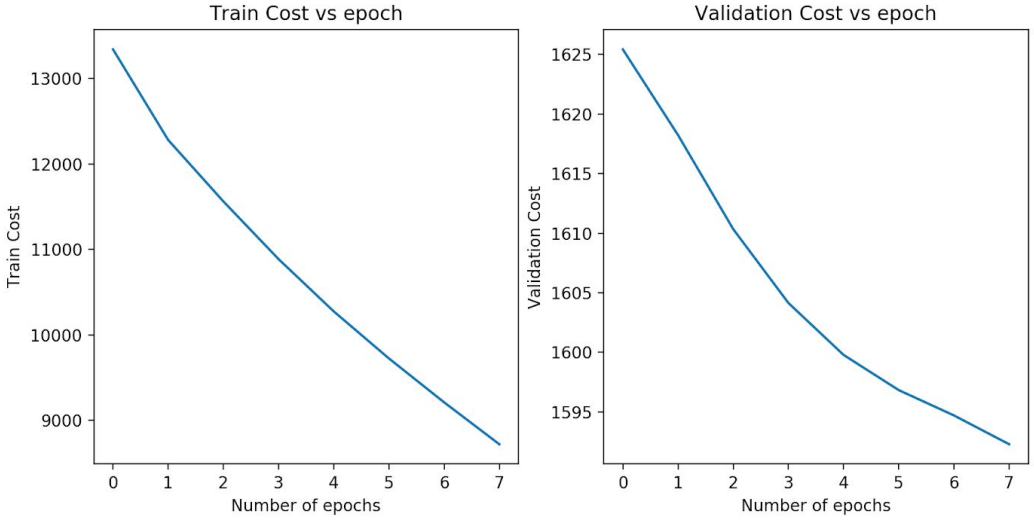


*Figure 7. Cost vs epochs*

Both the train and validation accuracy increases as the number of epochs increases. Also, both the train and validation cost decreases as the number of epochs increases.

(e) Chosen $\lambda = 0.01$, the final validation accuracy = 0.6913632514817951 and the final test accuracy = 0.6867061812023709.

The model with the regularization penalty perfroms a slight better than the model without the regularization penalty.

4. Selected 3 different base models:

Knn : k = 11

Neural networks : number of iterations = 8, learning rate = 0.05, $\lambda = 0$, k = 50 (AutoEncoder)

Item response theory : number of iteration = 12, learning rate = 0.01

*The bagging ensemble process:*

For knn and neural networks, resample the sparse matrix first, while for item response theory, resample the training data. Then training each model, recording each prediction and compute the final prediction by averaging 3 predictions. Finally evaluating the accuracy for the final averaged prediction.

*Calculated accuracy:*

final validation accuracy = 0.6157211402766017

final test accuracy = 0.6217894439740334

Selected 3 identical base models:

Knn : k = 11

*The bagging ensemble process:*

Run three Knn models with k = 11, each time resample the sparse matrix first. To compute the final prediction, take the average of the three predicted values performed by these three Knn models. Then compute the validation and test accuracy using the final prediction.

*Calculated accuracy:*

final validation accuracy: 0.6007620660457239

final test accuracy: 0.6082416031611628

Do you obtain better performance using the ensemble? Why or why not?

Both of the final validation accuracy and the final test accuracy show that the ensemble doesn't obtain better performance because bagging would not affect the bias, thus the accuracy of each model's prediction would not improve.

This might be caused by too little number of models taken into consideration. Also, we compute the final prediction only by averaging the results of different models. Maybe they should not be treated equally since different algorithms might perform differently.

## Part B

**Formal Description:**

To improve the optimation, we take consideration of the ensemble algorithm and neural networks algorithm in part A. The ensemble algorithm mixes each base model only by taking average prediction, which may not optimize the results. We want to apply weighted ensembling such as adaboosting to enhance the accuracy of our prediction by using the neural networks algorithm as our base models. To be specific, below is our training procedure.

First we initialize the weights of all the training data to 1 (Dimension NxM for N students and M questions). In order for the weights to work properly for all training samples. We convert 0 to -1 for samples indicating incorrect answers. (Otherwise weight * 0 will always yield zero and produce no effect.)

Assume we have $T$ number of iterations for training, then for each iteration $t$, we will first fit a neural network model to the weighted data. The neural networks algorithm is similar from Part A. However, since the input sample is now -1 or 1 instead of 0 or 1, slight modification of the algorithm is required to make sure the prediction is between [-1,1]. In particular, for function:

$$f(v;\theta) = h(W^{(2)}g(W^{(1)}v + b^{(1)}) + b^{(2)})$$

The activation function $h$ and $g$ will both be Tanh activation functions so that our prediction will lie on the region -1 to 1. Define the trained neural networks model to be nn_t. Then we

$$\text{err}_t = \frac{\sum_{n=1}^{N}\sum_{m=1}^{M} w_m^{(n)}\, \mathbb{I}\{nn_t(w_m^{(n)}) \neq t_m^{(n)}\}}{\sum_{n=1}^{N}\sum_{m=1}^{M} w_m^{(n)}}$$

will compute the weighted error as following:

The weighted error computes the total weights of misclassified samples divided by the total weight. Using the weighted error to compute the coefficient for nn_t as following:

$$\alpha_t = \frac{1}{2}\log\frac{1-\text{err}_t}{\text{err}_t}$$

Combine the weighted error and coefficient we will update the weights for the next iteration as following:

$$w_m^{(n)} = w_m^{(n)}\exp\left(-\alpha_t t_m^{(n)} nn_t(x_m^{(n)})\right)$$

As a result, misclassified samples will gain more attention for the next iteration. The final prediction will combine all the models with calculated coefficient:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T}\alpha_t nn_t(x)\right)$$

It is clear that the model with lower weighted error will have a larger coefficient since its prediction is more accurate. Also, during the training procedure, we update the weights to let the next model focus more on the misclassified samples. Therefore we should expect the adaboosting algorithm performs better on optimization.

We also tune the hyperparameter to yield the best validation accuracy for our model, the chosen hyperparameters are
Neural networks: k = 10, lr = 0.05, num_epoch = 1, lamb = 0.05
AdaBoost: number of iterations = 10
*Note that when evaluating the accuracy, the threshold is now 0 since our prediction is now either positive or negative.

**Figure or Diagram:**
The chart below demonstrates the general training procedure of our model.

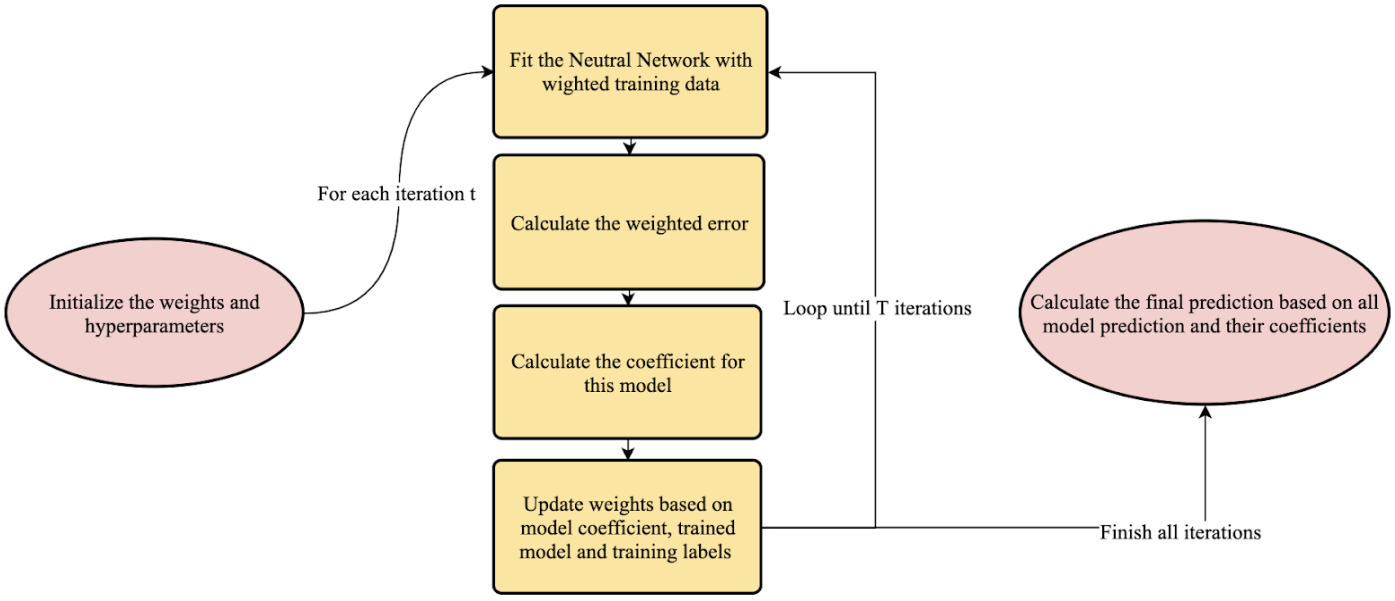


*Figure 8. Training process for adaboost using neural network algorithm*

As mentioned above, the training procedure starts by initializing weights and hyperparameters. Each iteration involves fitting a model, calculating weighted error and model coefficient and updating weights. Finally, the prediction will be calculated based on all trained models and their coefficients.

**Comparison or Demonstration:**
<u>Hyperparameters:</u>
Our Model: neural networks: k = 10 (AutoEncoder), lr = 0.05, num_epoch = 1, lamb = 0.05
adaBoost: number of iterations = 10
Neural Networks: k=10 (AutoEncoder), num_epoch = 10, lr = 0.05, lamb = 0.05
Item Response Theory: number of iterations = 10, lr = 0.05
KNN: k = 11

Validation Accuracies:

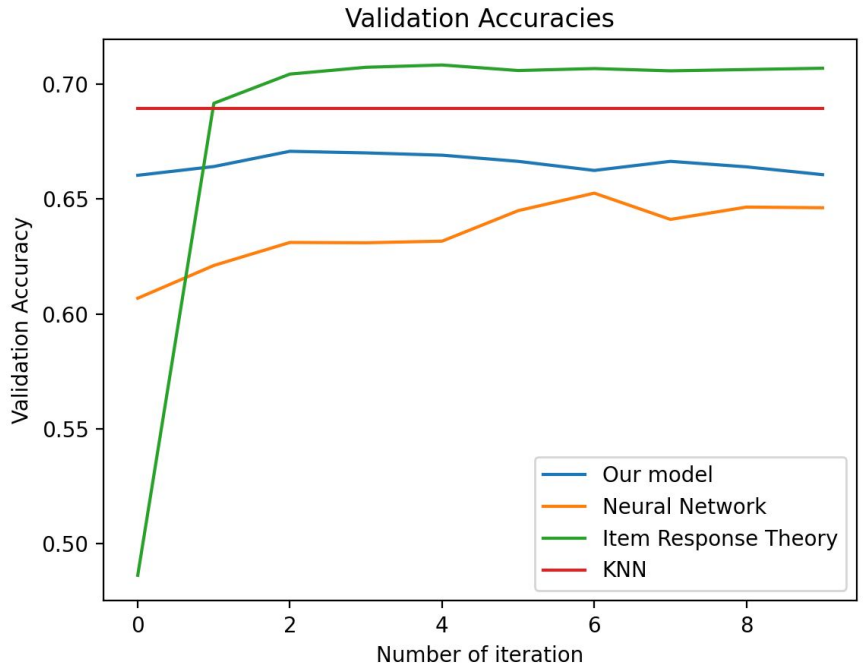


*Figure 9. Validation Accuracy vs iteration for each model*

Test Accuracies:

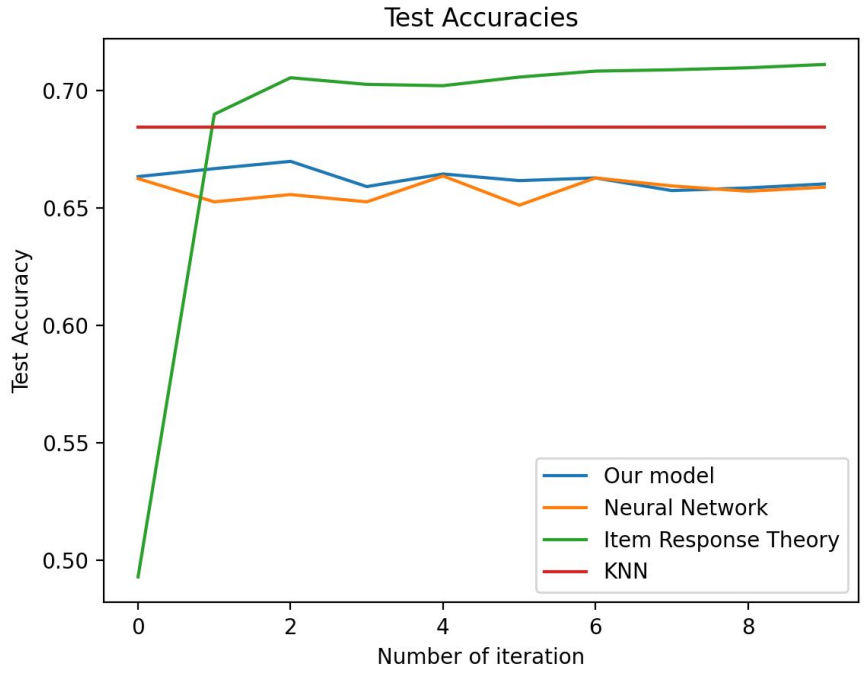


*Figure 10. Test Accuracy vs iteration for each model*

Our model obtains final validation accuracy = 0.6725938470222975
final test accuracy = 0.6697713801862828

Compared with other 3 baseline models, it doesn't obtain the highest accuracy, but it gets higher validation and test accuracies than base neural networks models.

To test if our algorithm actually works, we can inspect the weighted error and coefficient for the trained neural network model at each iteration. The fitted model and coefficient illustrates how the weights are updated during the training process, and we can track how the updated weights for a misclassified example influence its prediction. Ideally, the misclassified examples should be emphasized and get better estimation at the next iteration.

**Limitations:**
If we increase the number of iterations, the training accuracy will converge to 1, but the validation accuracy will decrease. As shown in the following plot:

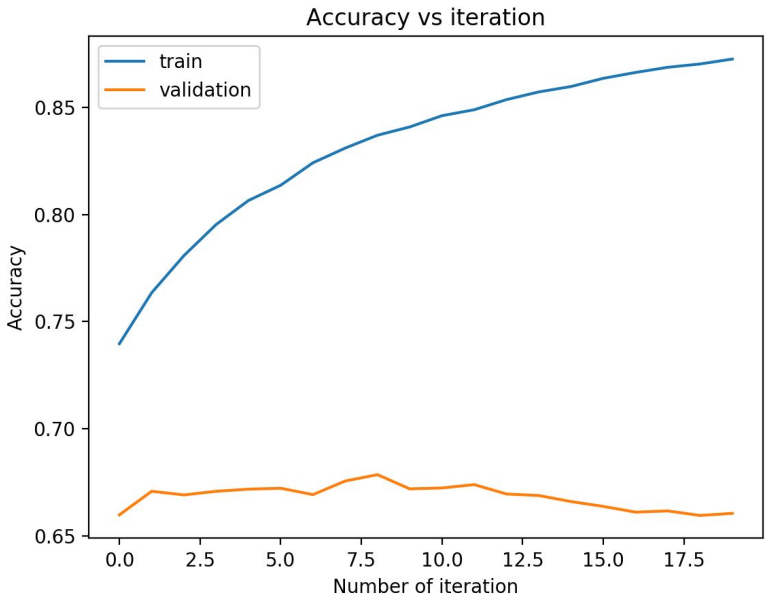


*Figure 11. Accuracy vs iteration for our model*

1. When the weak classifier (neural network) in our model gets more complex, our model would perform poorly on the validation data and test data.To be more precise, more layers in the neural network model, the training accuracy would converge to 1, however due to the overfitting, the validation accuracy and the test accuracy would be worse. To handle the overfitting problem, we might adjust the weak classifier to become less complex as long as it can achieve higher than 50% accuracy and train reasonable iterations by inspecting validation accuracy.

2. When there are many noisy datasets , the AdaBoost model would focus on those noise to get a higher accuracy on the training dataset. When evaluating the model on the validation or test datasets, the model would overfit on those noisy sets which makes the performance poorly. To handle the overfitting problem caused by the noisy datasets, we might set a regularizer on our model to punish those noisy data.

**Contributions:**

Part A

Zuoyu Wang: 1, 2 Siqi Chen: 3, 4

Part B

Zuoyu Wang: 1, 2 Siqi Chen: 3, 4