1 Part A

1.1 k-Nearest Neighbour

a) Complete a function main located at knn.py that runs kNN for different values of k ∈ 1, 6, 11, 16, 21, 26. Plot and report the accuracy of the validation data as a function of k.

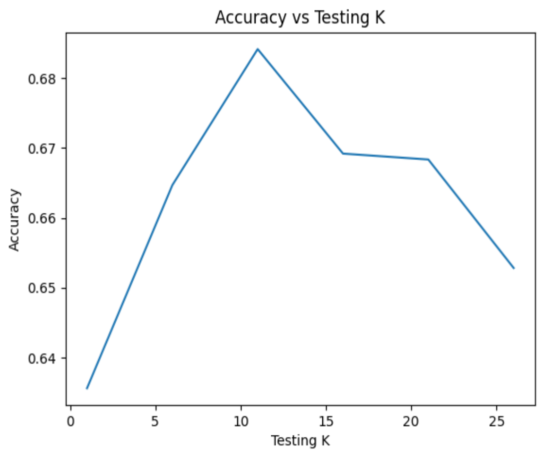

Figure 1: The accuracy of knn model with different k value

b) Choose k^* that has the highest performance on validation data. Report the chosen k^* and the final test accuracy

By observation, the KNN algorithm reaches its maximum accuracy at around k=11

c) Implement a function knn_impute_by_item on the same file that performs item-based collaborative filtering instead of user-based collaborative filtering. Given a question, kNN finds the closest question that was answered similarly, and predicts the correctness based on the closest question's correctness. State the underlying assumption on item-based collaborative filtering. Repeat parts (a) and (b) with item-based collaborative filtering.

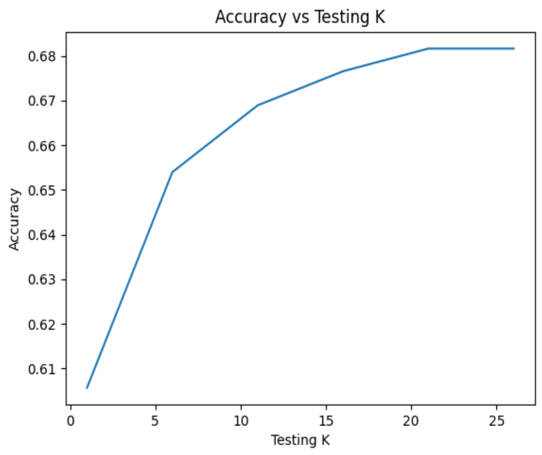

Figure 2: The accuracy of knn model with different k value

By observation, the KNN algorithm reaches its maximum accuracy at around k=26

d) Compare the test performance between user- and item- based collaborative filtering. State which method performs better.

Both algorithms have roughly the same accuracy (user- :0.684, item- :0.681). However, since user- has better running time, thus user- outperforms item-.

c) List at least two potential limitations of kNN for the task you are given.

1) Slow training time: for n = number of Users, m = number of questions. Then the KNN model for both algorithms would take O(nm) to train the model.

2) Poor accuracy: The best accuracy is only 0.684, which performs slightly better than random guessing.

1.2 Item Response Theory

a) Derive the log-likelihood $logp(C|\theta, \beta)$ for all students and questions. Here C is the sparse matrix. Also, show the derivative of the log-likelihood with respect to θ_i and β_j

$$\log(C|\theta_i, \beta_j) = \sum_j \sum_i (1 - C_{ij}) \log\left(1 - \frac{e^{\theta_i - \beta_j}}{e^{\theta_i - \beta_j} + 1}\right) + C_{ij} \log\left(\frac{e^{\theta_i - \beta_j}}{e^{\theta_i - \beta_j} + 1}\right) \tag{1}$$

By calculations (Though calculators), we have:

$$\frac{d}{d\theta_i} = \sum_j \frac{(C_{ij} - 1)\, e^{\theta_i} + e^{\beta_j} C_{ij}}{e^{\theta_i} + e^{\beta_j}} \tag{2}$$

$$\frac{d}{d\beta_j} = -\sum_i \frac{C_{ij}e^{\beta_j} + (C_{ij} - 1)\, e^{\theta_i}}{e^{\beta_j} + e^{\theta_i}} \tag{3}$$

b) Implement missing functions in item_response.py that perform alternating gradient descent on θ and β to maximize the log-likelihood. Report the hyperparameters you selected. With your chosen hyperparameters, report the training curve that shows the training and validation log-likelihoods as a function of iteration

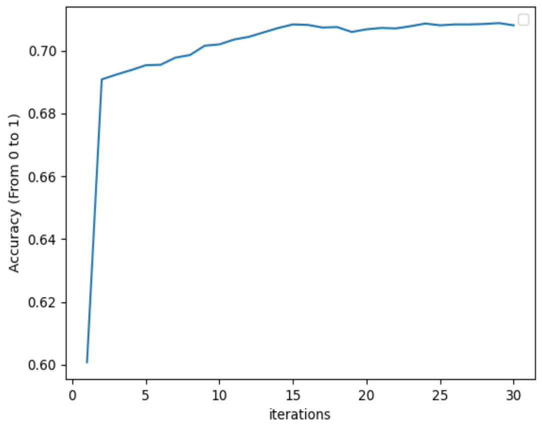

Figure 3: The accuracy of IRT on training set vs numbers of iterations

c) With the implemented code, report the final validation and test accuracies.
Final NLLK: -27278.6758692845 Final ACC: 0.7080158058142817

d) Select three questions j1, j2, and j3. Using the trained θ and β, plot three curves on the same plot that shows the probability of the correct response p(cij = 1) as a function of β given a question j. Comment on the shape of the curves and briefly describe what these curves represent.

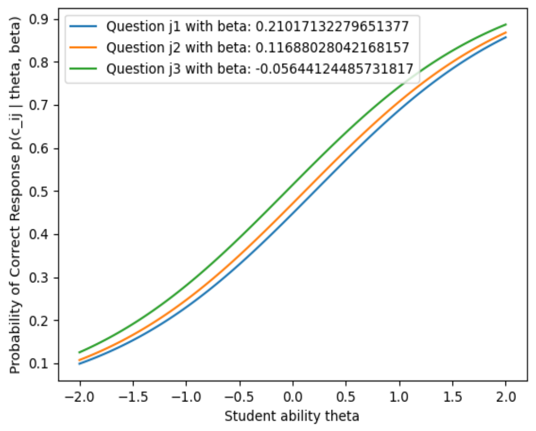

Figure 4: Correct Response Probability vs Student Ability theta

Given the given graph, we can see that for a problem with the same difficulty, the higher the students' ability, the higher the probability that a student might solve a problem. For the same student, the higher the difficulty, the lower the probability that the student solve the problem.

1.3 Matrix factorization

a) Using a function svd_reconstruct that factorizes the sparse matrix using singular value decomposition, try out at least 5 different k and select the best k using the validation set. Report the final validation and test performance with your chosen k.

By choosing k $\in$ [1, 5, 10, 20, 40, 80, 160]. We have the best k is 5, with validation accuracy being 0.659

b) State one limitation of SVD in the task you are given.

Since there are lots of missing values in the given data, the SVD algorithm is trying to fill in those missing values with a mean of each question, which is not accurate.

c) Implement functions als and update_u_z located at the same file that performs alternating updates. You need to use SGD to update un $\in R^k$ and zm $\in R^k$ as described in the docstring. To be noted, this is different from the alternating least squares (ALS) we introduced in the course slides, where we used the direct solution.

d) Learn the representations U and Z using ALS with SGD. Tune learning rate and number of iterations. Report your chosen hyperparameters. Try at least 5 different values of k and select the best k* that achieves the lowest validation accuracy.

Choose k = 50, lr = 0.05, iteration = 128000, the validation accuracy is 0.704

e) With your chosen k, plot and report how the training and validation squared-error losses change as a function of iteration. Also report final validation accuracy and test accuracy

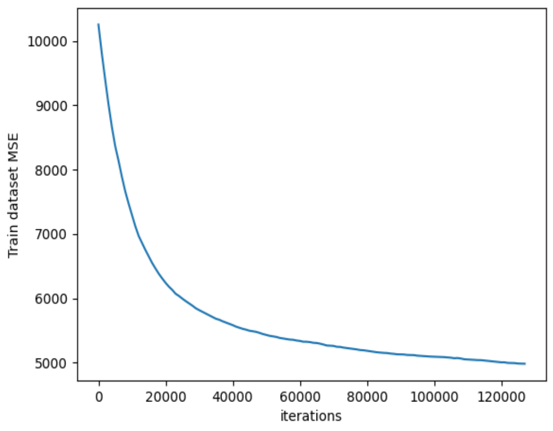

Figure 5: Matrix Factorization Train MSE vs numbers of iterations

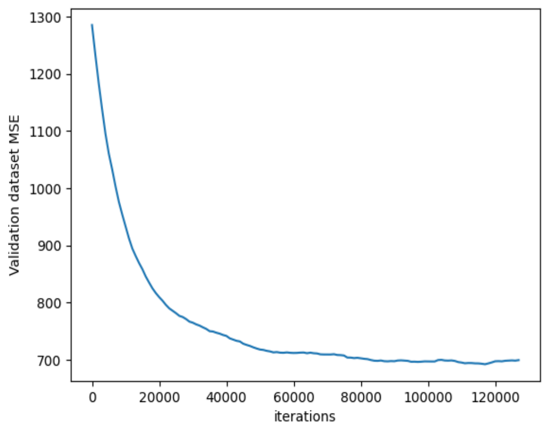

Figure 6: Matrix Factorization Train MSE vs numbers of iterations

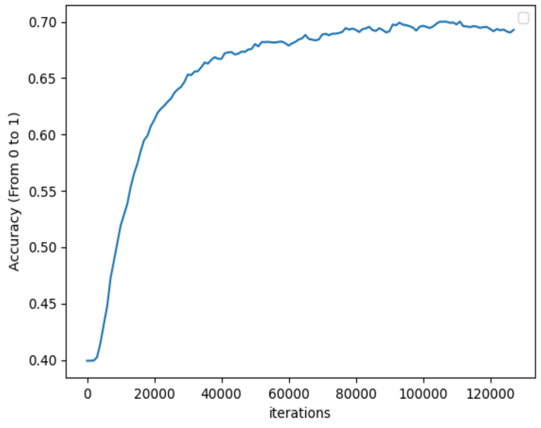

Figure 7: Matrix Factorization Acc vs numbers of iterations

Test Accuracy is 0.703

1.4 Bagging Ensemble

In this problem, you will be implementing a bagging ensemble to improve the stability and accuracy of your base models. Select and train 3 base models with bootstrapping the training set. You may use the same or different base models. Your implementation should be completed in part_a/ensemble.py. To predict the correctness, generate 3 predictions by using the base model and average the predicted correctness. Report the final validation and test accuracy. Explain the ensemble process you implemented. Do you obtain better performance using the ensemble? Why or why not?

a) Base Model: SGD matrix factorization

b) bootstrapping number: 3

c) Ensemble Process: First bootstrap 3 datasets from the original training data, then train SGD matrix factorization on each bootstrapped dataset, lastly take a majority vote of those models, and evaluate their final accuracy.

d) Final validation accuracy: 0.683

e) Final test accuracy: 0.684

f) Hyperparameters: k=50, lr=0.05, iteration=128000

g) Conclusion: Notice that both the Final Validation accuracy and test accuracy (˜0.68) are lower compared to what we obtained before the bagging ensemble (˜0.70). This is due to the correlations between the datasets, by Bagging Ensemble, we are assuming there is each entry of the training data is independent, however, this is not true for our dataset. If a student can't do a Calculus problem, there is a greater probability that he also can't solve other Calculus problems. This correlation causes the Bagging ensemble to be less effective.

h) Solution: We can use Random Forst to reduce the correlation between datasets by introducing additional variability.

2 Part B

2.1 L2 regularization

a) Formal Description: In our previous implementation, we are trying to find the minimum point of

$$L = \frac{1}{2}\sum_{n,m}(C_{nm} - u^T z)^2 \tag{4}$$

However, this function is prone to overfitting, so we decided to add L2 regularization to solve the overfitting problem. By adding an L2 regularizer, we got the equation:

$$L = \frac{1}{2}\sum_{n,m}(C_{nm} - u^T z)^2 + \frac{\lambda}{2}(\|u\|^2 + \|z\|^2) \tag{5}$$

Hyperparameters: k=50, lr=0.05, iterations=400,000, $\lambda = 0.025$

b) Figure or Diagram: We are comparing the difference in accuracy, Train Data Square Error through iterations, and Validation Data Square Error through iterations. And compare between two algorithms.

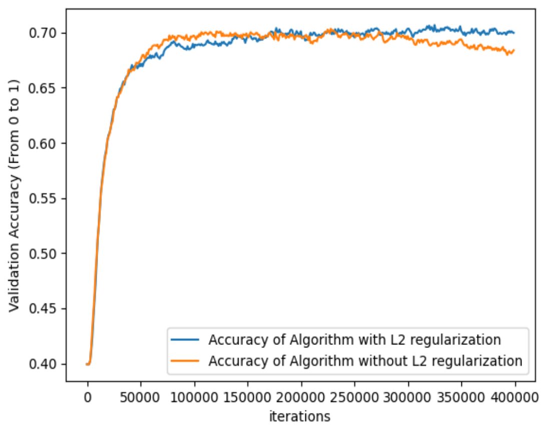

Figure 8: Matrix Factorization Acc vs numbers of iterations

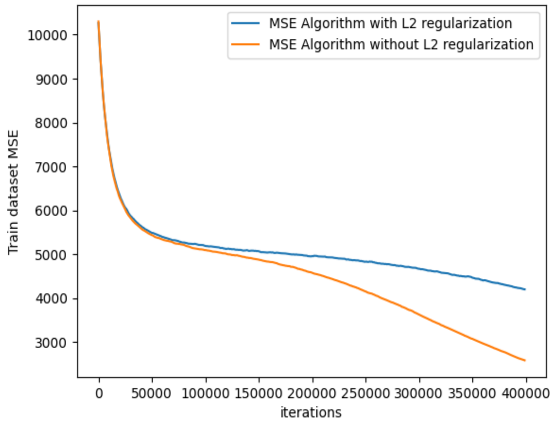

Figure 9: Matrix Factorization MSE vs numbers of iterations on training data

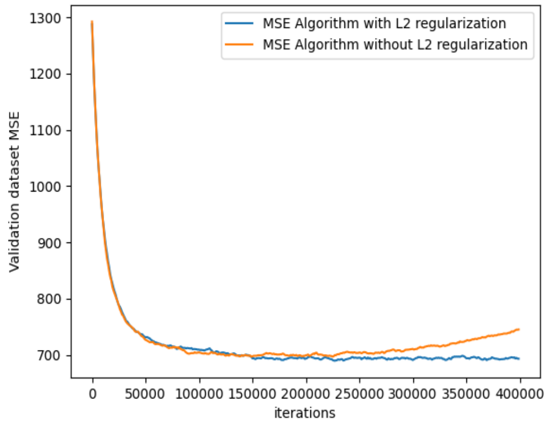

Figure 10: Matrix Factorization MSE vs numbers of iterations on testing data

From Figure 8, we can see that SGD_L2 outperforms SGD at around 200,000 iterations. Figures 9 and 10 show a significant drop in MSE for the Training set and an increase in MSE for the Validation set, indicating a sign of overfitting. However, by implementing early stopping at 100,000 to 150,000 iterations, both algorithms perform identically in terms of both MSE and accuracy

C) Limitations: The incorporation of an L2 regularization term does not yield a significant increase in accuracy as anticipated. In comparison, implementing early stopping achieves comparable accuracy and MSE to that of L2 regularization, while requiring fewer iterations.

2.2 Adding Biases

a) Formal Description: In our previous implementation, we did not consider effects associated with either users or items, known as biases or intercepts, which is independent of any interactions.[1] So in this section, we are going to add a bias term to the Algorithm and see how it works.

$$L = \frac{1}{2}\sum_{n,m}(C_{nm} - (\mu + b_n + b_m + u_n^T z_m))^2 + \frac{\lambda}{2}(\|u_n\|^2 + \|z_m\|^2) + \frac{\lambda_b}{2}(b_n^2 + b_m^2) \tag{6}$$

In this equation, we introduce new bias terms:
1) μ: Global bias term
2) b_n: Bias term for Student n
3) b_m: Bias term for Student m
4) λ_b: Multiplier Coefficient for Bias L2 regularization
In this equation, our estimation of the C_{nm}, $\bar{r}$ is given by

$$\bar{r} = \mu + b_n + b_m + u_n^T z_m \tag{7}$$

Hyperparameters: k=50, lr=0.05, iterations=400,000, $\lambda = 0.025$, $\lambda_b = 0.025$,

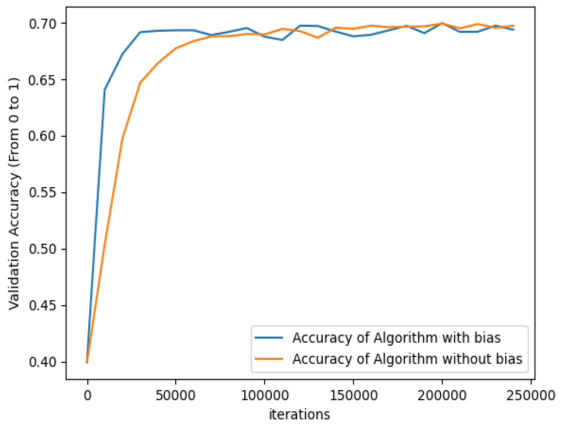

Figure 11: Matrix Factorization Acc vs numbers of iterations

Figure 11 illustrates a notable difference in the convergence rate between the SGD_L2_Biased and SGD_L2 models. Specifically, with the same lr value, the SGD_L2_Biased model requires approximately 30,000 iterations to achieve an accuracy of 0.7, whereas the SGD_L2 model takes around 100,000 iterations to reach the same level of accuracy. However, it's important to note that beyond 100,000 iterations, there is no significant difference in accuracy observed between the two models

2.3 Additional Input Sources

In our previous model, we relied primarily on students' past answer histories. However, the dataset's sparsity limits its effectiveness: some students rarely answer certain questions, providing minimal data. This sparsity likely contributes to our model's stagnation at 70%

accuracy. To improve, we're modifying our Stochastic Gradient Descent (SGD) equation to include new parameters. The updated SGD_M now integrates L2 regularization, a bias factor, and additional inputs, offering a more robust predictive framework. And we expect this change can improve accuracy beyond 70%

$$\begin{aligned} L = \frac{1}{2}\sum_{n,m}&(C_{nm} - (\mu + b_n + b_m + [u_n + y_{\text{age}}]^T[z_m + y_{\text{category}}]))^2 \\ &+ \frac{\lambda}{2}(\|u_n\|^2 + \|z_m\|^2) \\ &+ \frac{\lambda_b}{2}(b_n^2 + b_m^2) \\ &+ \frac{\lambda_{\text{age}}}{2}\|y_{\text{age}}\|^2 \\ &+ \frac{\lambda_{\text{category}}}{2}\|y_{\text{category}}\|^2 \end{aligned} \tag{8}$$

In this equation, we introduce new terms:
1) y_{age}: Trained Matrix for additional input (age) for student
2) $y_{category}$: Trained Matrix for additional input (category) for question

From here, we add an extra parameter y_{category} and y_{age} into the equation. We are grouping categories and ages into J and I categories, and then each y_{category} and y_{age} would be Jxk and IxK matrix. Where each entry represents the bias of each category toward each latent feature. For example, a kindergarten student(age 0-8) would expect to have a very small bias for solving a calculus problem. We didn't use data like gender and premium_pupil due to two reasons. 1) Weak correlation to correctness. 2) Sparse in data

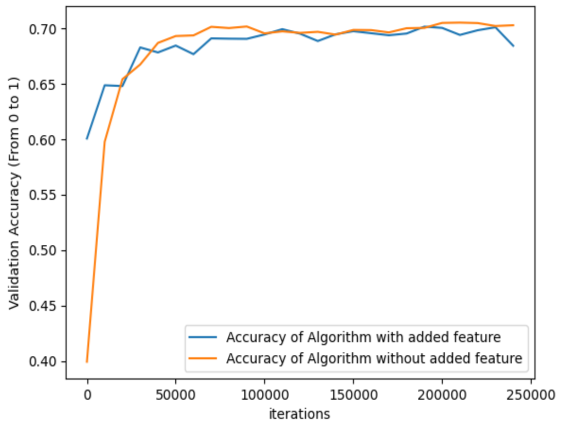

Figure 12: Matrix Factorization Acc vs numbers of iterations

From this graph, we can see no evidence of accuracy improvement with the newly introduced additional input source. Both SGD with additional input source and Base SGD converge into the same accuracy after 100,000 iterations.

2.4 Hyperparameters Togging

In this section, we are trying to toggle the weight of Bias, Age, and Question Category participating in Prediction and Training. And trying to find the balance between those models.

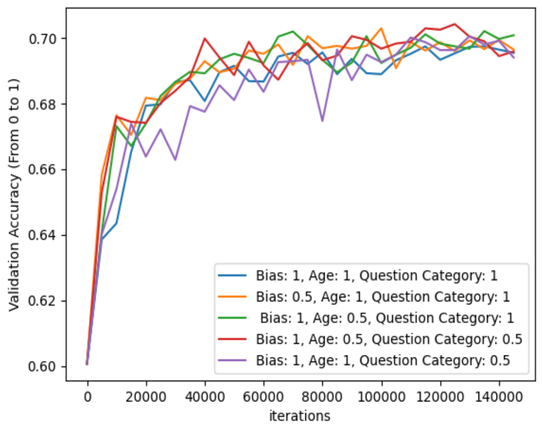

Figure 13: Matrix Factorization Acc vs numbers of iterations

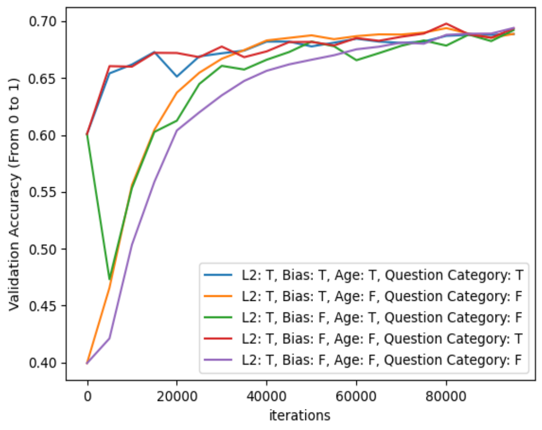

Figure 14: Matrix Factorization Acc vs numbers of iterations

In Figure 13, we experiment with varying the weights of Bias, Age, and Question Category. Figure 14 shows a comparison of accuracy when some optimizations are disabled, revealing that all curves converge at an accuracy of 0.7. Additionally, we attempted to introduce a displacement to the matrix, but the local maximum of the final accuracy was approximately 0 for displacement, suggesting that the inaccuracy is likely due to high variance, instead of high Bias.

2.5 Conclusion

In conclusion, the application of various techniques to Stochastic Gradient Descent (SGD) algorithms has resulted in a faster convergence rate and a more stable accuracy curve. However, these techniques do not enhance the final accuracy of the output matrix. This persistent inaccuracy is likely attributed to the high variance of the output. Addressing this issue may require the adoption of more advanced models, such as Factorization Machines, to achieve any significant improvements.

References

[1] Yehuda Koren, Robert Bell and Chris Volinsky *Matrix Factorization technique for recommender system.* IEEE Computer Society, 2009, 0018-9162/09/