CSC311 Project Report

Predicting Student Performance in Diagnostic Assessments Using Machine Learning

Jiaxi Li, Jiawei Gong, Nuo Xu

April 2024

1 Part A

1.1 Collaborative filtering with k-Nearest Neighbor

The initial attempt is employing the k-Nearest Neighbor (kNN) algorithm to forecast students' correctness in responding to questions. This methodology involves two distinct approaches. The first, user-based collaborative filtering, enables kNN to predict a student's correctness by examining the performance of students who share the closest similarities. The second approach, which is item-based collaborative filtering, makes prediction on the correctness of single question based on the question that is most similar to the given one.

1.1.1 User-based Collaborative Filtering

Upon implementing `knn.py` with the provided starter code and running the program to tune the hyperparameter k for $k \in \{1, 6, 11, 16, 21, 26\}$, the results are presented in Table 1 and Figure 1. The optimal k value, determined by the highest validation accuracy, is 11, yielding a test accuracy of approximately 68.42% when $k = 11$.

1.1.2 Item-based Collaborative Filtering

The core underlying assumption on item-based collaborative filtering is the correctness of specific question can be predicted based on questions with similar patterns of responses. More precisely, given question A has been answered correctly or incorrectly by students in a similar pattern as question B, then the student's correctness in answering question A matches that in answering question B.

Then, similarly, after implementing the item-based filtering algorithm in `knn.py` and tuning the hyperparameter k within the set $\{1, 6, 11, 16, 21, 26\}$ utilizing the validation data set, the selected best k is 21, corresponding to a test accuracy of approximately 68.16%.

Table 1: Validation Accuracies for User-Based and Item-Based kNN

k	1	6	11	16	21	26
User-Based Acc	0.624	0.678	0.690	0.676	0.669	0.652
Item-Based Acc	0.607	0.654	0.683	0.686	0.692	0.690

1.1.3 Model Selection & Limitation

Overall, user-based collaborative filtering is better for several reasons. First, it generates a slightly higher test accuracy compared to item-based collaborative filtering (68.42% versus 68.16%). Second, the validation accuracy for user-based collaborative filtering increases as k grows from 1 to 11 and then starts to decrease after k=11. This decline in validation accuracy suggests that beyond k=11, the model's predictions are affected by excessive random noise from the training data, leading to diminished performance. This implies

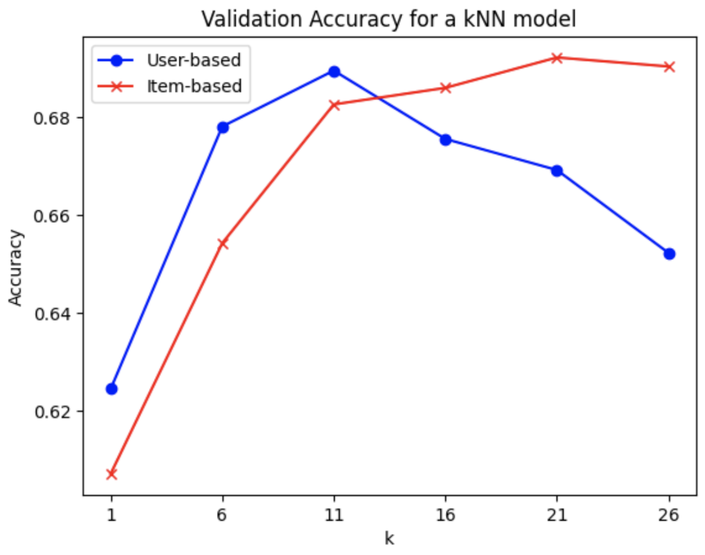

Figure 1: Accuracy Trends of kNN Model with Varying k: User-Based vs. Item-Based Collaborative Filtering

that the model with k=11 is likely to be well-generalized and not overfitted. Third, considering the provided dataset, the questions cover various subjects, which undermines the assumption on item-based filtering that the correctness of a specific question can be predicted based on questions with similar response patterns. Nevertheless, user-based collaborative filtering can still performs well because the predictions are based on the consistency of user behavior, rather than on that of questions content.

The kNN algorithm, while useful, does have certain limitations. First, kNN operates under the assumption that closeness in feature space is indicative of similarity in the output variable. Therefore, if students have inconsistent behaviors across different questions, kNN's predictions are likely to be less precise. Second, kNN's computational efficiency decreases with larger datasets, as each prediction requires a comparison with every data point. Furthermore, within the context of predicting student correctness, there may be a need to frequently retrain the kNN model to accommodate the inclusion of new students or the introduction of new questions.

1.2 The Item Response Theory Model

1.2.1 Deriving the log-likelihood

For student i, and question j

$\log P(c_{ij}|\theta_i, \beta_j)$

$= c_{ij} \cdot P(c_{ij} = 1) + (1 - c_{ij}) \cdot P(c_{ij} = 0)$

$= c_{ij} \cdot \log\left(\frac{e^{\theta_i - \beta_j}}{1+e^{\theta_i - \beta_j}}\right) + (1 - c_{ij}) \cdot \log\left(1 - \frac{e^{\theta_i - \beta_j}}{1+e^{\theta_i - \beta_j}}\right)$

$= c_{ij} \cdot \log\left(\frac{e^{\theta_i - \beta_j}}{1+e^{\theta_i - \beta_j}}\right) + (1 - c_{ij}) \cdot \log\left(\frac{1}{1+e^{\theta_i - \beta_j}}\right)$

Let $\sigma(x) = \frac{e^x}{1+e^x}$

The log-likelihood in the form of sparse matrix:

$$\begin{bmatrix} \left(c_{0,0}\cdot\log\left(\frac{e^{\theta_0-\beta_0}}{1+e^{\theta_0-\beta_0}}\right)+(1-c_{0,0})\cdot\log\left(\frac{1}{1+e^{\theta_0-\beta_0}}\right)\right) & \cdots & \left(c_{0,1773}\cdot\log\left(\frac{e^{\theta_0-\beta_{1773}}}{1+e^{\theta_0-\beta_{1773}}}\right)+(1-c_{0,1773})\cdot\log\left(\frac{1}{1+e^{\theta_0-\beta_{1773}}}\right)\right) \\ \left(c_{1,0}\cdot\log\left(\frac{e^{\theta_1-\beta_0}}{1+e^{\theta_1-\beta_0}}\right)+(1-c_{1,0})\cdot\log\left(\frac{1}{1+e^{\theta_1-\beta_0}}\right)\right) & \cdots & \left(c_{1,1773}\cdot\log\left(\frac{e^{\theta_1-\beta_{1773}}}{1+e^{\theta_1-\beta_{1773}}}\right)+(1-c_{1,1773})\cdot\log\left(\frac{1}{1+e^{\theta_1-\beta_{1773}}}\right)\right) \\ \vdots & \ddots & \vdots \\ \left(c_{541,0}\cdot\log\left(\frac{e^{\theta_{541}-\beta_0}}{1+e^{\theta_{541}-\beta_0}}\right)+(1-c_{541,0})\cdot\log\left(\frac{1}{1+e^{\theta_{541}-\beta_0}}\right)\right) & \cdots & \left(c_{541,1773}\cdot\log\left(\frac{e^{\theta_{541}-\beta_{1773}}}{1+e^{\theta_{541}-\beta_{1773}}}\right)+(1-c_{541,1773})\cdot\log\left(\frac{1}{1+e^{\theta_{541}-\beta_{1773}}}\right)\right) \end{bmatrix}$$

$logP(\mathbf{C}|\boldsymbol{\theta},\boldsymbol{\beta})$

$= \sum_{i=0}^{541}\sum_{j=0}^{1773}\left((c_{ij}\cdot\log\left(\frac{e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}}\right)+(1-c_{ij})\cdot\log\left(\frac{1}{1+e^{\theta_i-\beta_j}}\right)\right)$

Let $\mathbf{S} = \frac{e^{\boldsymbol{\Theta}-\boldsymbol{B}}}{1+e^{\boldsymbol{\Theta}-\boldsymbol{B}}}$
(1) $\boldsymbol{\Theta}$ is the 542×1774 matrix expanded by θ across columns
(2) $\boldsymbol{B}$ is the 542×1774 matrix expanded by β across rows.

$L = \mathbf{C}\odot\log(\mathbf{S}) + (1-\mathbf{C})\odot\log(1-\mathbf{S})$

$logp(\mathbf{C}|\boldsymbol{\theta},\boldsymbol{\beta}) = \sum(L)$ (sums all elements in matrix)

(1) For student i, and question j, the derivative of the log-likelihood with respect to β_j is

$$\begin{aligned}
&\frac{\partial}{\partial\beta_j}\log P(c_{ij}|\theta_i,\beta_j)\\
&= \frac{\partial}{\partial\beta_j}\left(c_{ij}\cdot\log\left(\frac{e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}}\right)+(1-c_{ij})\cdot\log\left(\frac{1}{1+e^{\theta_i-\beta_j}}\right)\right)\\
&= c_{ij}\cdot\frac{\partial}{\partial\beta_j}\log\left(\frac{e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}}\right)+(1-c_{ij})\cdot\frac{\partial}{\partial\beta_j}\log\left(\frac{1}{1+e^{\theta_i-\beta_j}}\right)\\
&= c_{ij}\cdot\frac{1+e^{\theta_i-\beta_j}}{e^{\theta_i-\beta_j}}\cdot\frac{\frac{\partial}{\partial\beta_j}e^{\theta_i-\beta_j}\cdot(1+e^{\theta_i-\beta_j})-e^{\theta_i-\beta_j}\cdot\frac{\partial}{\partial\beta_j}(1+e^{\theta_i-\beta_j})}{(1+e^{\theta_i-\beta_j})^2}+(1-c_{ij})\cdot\frac{1+e^{\theta_i-\beta_j}}{1}\cdot\frac{0-1\cdot(1+e^{\theta_i-\beta_j})\cdot(-1)}{(1+e^{\theta_i-\beta_j})^2}\\
&= c_{ij}\cdot\frac{1+e^{\theta_i-\beta_j}}{e^{\theta_i-\beta_j}}\cdot\frac{e^{\theta_i-\beta_j}\cdot(-1)(1+e^{\theta_i-\beta_j})-(e^{\theta_i-\beta_j})(-1)(e^{\theta_i-\beta_j})}{(1+e^{\theta_i-\beta_j})^2}+(1-c_{ij})\cdot\frac{1+e^{\theta_i-\beta_j}}{1}\cdot\frac{0-1\cdot(1+e^{\theta_i-\beta_j})\cdot(-1)}{(1+e^{\theta_i-\beta_j})^2}\\
&= c_{ij}\cdot\frac{(-1)(e^{\theta_i-\beta_j})(1+e^{\theta_i-\beta_j})+(e^{\theta_i-\beta_j})^2}{e^{\theta_i-\beta_j}\cdot(1+e^{\theta_i-\beta_j})}+(1-c_{ij})\cdot\frac{e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}}\\
&= c_{ij}\cdot\frac{(-1)}{(1+e^{\theta_i-\beta_j})}+(1-c_{ij})\cdot\frac{e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}}\\
&= -c_{ij}+\frac{e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}}
\end{aligned}$$

For all data involving β_j, the sum of derivative of log-likelihood with respect to β_j is

$$\begin{aligned}
&\frac{\partial}{\partial\beta_j}logP(\mathbf{C}|\boldsymbol{\theta},\boldsymbol{\beta})\\
&= \sum_{i=0}^{541}\frac{\partial}{\partial\beta_j}\log P(c_{ij}|\theta_i,\beta_j)\\
&= \sum_{i=0}^{541}(-c_{ij}+\frac{e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}})
\end{aligned}$$

Then, $\nabla_{\boldsymbol{\beta}}logP(\mathbf{C}|\boldsymbol{\theta},\boldsymbol{\beta})$

$$= \begin{bmatrix} \frac{\partial}{\partial\beta_0}logP(\mathbf{C}|\boldsymbol{\theta},\boldsymbol{\beta}) \\ \frac{\partial}{\partial\beta_1}logP(\mathbf{C}|\boldsymbol{\theta},\boldsymbol{\beta}) \\ \vdots \\ \frac{\partial}{\partial\beta_{1773}}logP(\mathbf{C}|\boldsymbol{\theta},\boldsymbol{\beta}) \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{i=0}^{541}(-c_{i,0} + \frac{e^{\theta_i - \beta_0}}{1+e^{\theta_i-\beta_0}}) \\ \sum_{i=0}^{541}(-c_{i,1} + \frac{e^{\theta_i - \beta_1}}{1+e^{\theta_i-\beta_1}}) \\ \vdots \\ \sum_{i=0}^{541}(-c_{i,1773} + \frac{e^{\theta_i - \beta_{1773}}}{1+e^{\theta_i-\beta_{1773}}}) \end{bmatrix}$$
$= \mathbf{1}_{542}^{\mathbf{T}} \cdot (-\mathbf{C} + \sigma(\mathbf{\Theta} - \boldsymbol{B})) \leftarrow$ sum by column

(1) $\mathbf{1}_{542}$ is 542-dimensional column vector with all elements equal to 1.
(2) $\mathbf{\Theta}$ is the 542 $\times$ 1774 matrix expanded by θ across columns
(3) B is the 542 $\times$ 1774 matrix expanded by β across rows

Our goal is to maximize $logP(\mathbf{C}|\boldsymbol{\theta}, \boldsymbol{\beta})$ using gradient descent and find β_j (j $\in \mathbb{N}$ and $0 \le j \le 1773$).

(2) For student i, and question j, the derivative of the log-likelihood with respect to θ_i is

$$\frac{\partial}{\partial\theta_i} \log P(c_{ij}|\theta_i, \beta_j)$$
$$= \frac{\partial}{\partial\theta_i}\left(c_{ij} \cdot \log\left(\frac{e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}}\right) + (1-c_{ij}) \cdot \log\left(\frac{1}{1+e^{\theta_i-\beta_j}}\right)\right)$$
$$= c_{ij} \cdot \frac{\partial}{\partial\theta_i} \log\left(\frac{e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}}\right) + (1-c_{ij}) \cdot \frac{\partial}{\partial\theta_i} \log\left(\frac{1}{1+e^{\theta_i-\beta_j}}\right)$$
$$= c_{ij} \cdot \frac{1+e^{\theta_i-\beta_j}}{e^{\theta_i-\beta_j}} \cdot \frac{\frac{\partial}{\partial\theta_i} e^{\theta_i-\beta_j} \cdot (1+e^{\theta_i-\beta_j}) - e^{\theta_i-\beta_j} \cdot \frac{\partial}{\partial\theta_i}(1+e^{\theta_i-\beta_j})}{(1+e^{\theta_i-\beta_j})^2} + (1-c_{ij}) \cdot \frac{1+e^{\theta_i-\beta_j}}{1} \cdot \frac{0-(1)\cdot(1+e^{\theta_i-\beta_j})\cdot(1)}{(1+e^{\theta_i-\beta_j})^2}$$
$$= c_{ij} \cdot \frac{1+e^{\theta_i-\beta_j}}{e^{\theta_i-\beta_j}} \cdot \frac{e^{\theta_i-\beta_j}\cdot(1)(1+e^{\theta_i-\beta_j}) - (e^{\theta_i-\beta_j})(1)(e^{\theta_i-\beta_j})}{(1+e^{\theta_i-\beta_j})^2} + (1-c_{ij}) \cdot \frac{1+e^{\theta_i-\beta_j}}{1} \cdot \frac{0-(1)\cdot(1+e^{\theta_i-\beta_j})\cdot(1)}{(1+e^{\theta_i-\beta_j})^2}$$
$$= c_{ij} \cdot \frac{(1)(e^{\theta_i-\beta_j})(1+e^{\theta_i-\beta_j}) - (e^{\theta_i-\beta_j})^2}{e^{\theta_i-\beta_j}\cdot(1+e^{\theta_i-\beta_j})} + (1-c_{ij}) \cdot \frac{(-1)e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}}$$
$$= c_{ij} \cdot \frac{1}{(1+e^{\theta_i-\beta_j})} + (1-c_{ij}) \cdot \frac{-e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}}$$
$$= c_{ij} - \frac{e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}}$$

For all data involving θ_i, the sum is

$$\frac{\partial}{\partial\theta_i} logP(\mathbf{C}|\boldsymbol{\theta}, \boldsymbol{\beta})$$
$$= \sum_{j=0}^{1773} \frac{\partial}{\partial\theta_i} \log P(c_{ij}|\theta_i, \beta_j)$$
$$= \sum_{j=0}^{1773} (c_{ij} - \frac{e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}})$$

Then:
$\nabla_{\boldsymbol{\theta}} logP(\mathbf{C}|\boldsymbol{\theta}, \boldsymbol{\beta})$

$$= \begin{bmatrix} \frac{\partial}{\partial\theta_0} logP(\mathbf{C}|\boldsymbol{\theta}, \boldsymbol{\beta}) \\ \frac{\partial}{\partial\theta_1} logP(\mathbf{C}|\boldsymbol{\theta}, \boldsymbol{\beta}) \\ \vdots \\ \frac{\partial}{\partial\theta_{541}} logP(\mathbf{C}|\boldsymbol{\theta}, \boldsymbol{\beta}) \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{j=0}^{1773}(c_{0,j} - \frac{e^{\theta_0-\beta_j}}{1+e^{\theta_0-\beta_j}}) \\ \sum_{j=0}^{1773}(c_{1,j} - \frac{e^{\theta_1-\beta_j}}{1+e^{\theta_1-\beta_j}}) \\ \vdots \\ \sum_{j=0}^{1773}(c_{541,j} - \frac{e^{\theta_{541}-\beta_j}}{1+e^{\theta_{541}-\beta_j}}) \end{bmatrix}$$

$= (\mathbf{C} - \sigma(\mathbf{\Theta} - \boldsymbol{B})) \cdot \mathbf{1}_{1772} \leftarrow$ sum by row

(1)$\mathbf{1}_{1772}$ is 1772-dimensional column vector with all elements equal to 1.
(2) $\mathbf{\Theta}$ is the 542 $\times$ 1774 matrix expanded by θ across columns
(3) $\boldsymbol{B}$ is the 542 $\times$ 1774 matrix expanded by β across rows

Our goal is to maximize this sum using gradient descent to find θ_i (i $\in \mathbb{N}$ where $0 \leq i \leq 541$)

1.2.2 Implementation & Tuning the Hyperparameter

The function is implemented within the provided `item_response.py`. The potential choices of the two hyperparameters we tuned are shown in below.

- learning rate = [0.001, 0.01, 0.05, 0.1, 0.2, 0.3]
- the number of iterations = [350,450,550]

Within all the possible combinations, the best learning rate of 0.3 and number of iterations of 550 generate the highest validation accuracy. The validation accuracy with selected hyperparameters is approximately 70.69% and the test accuracy is approximately 70.84%. The training and validation log-likelihoods as a function of iteration are shown in Figure 2 and Figure 3 respectively.

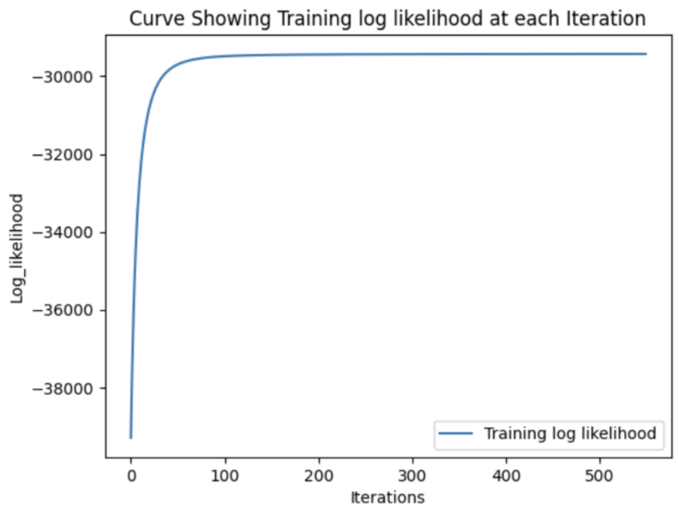

Figure 2: Training Log-likelihood at each iteration

1.2.3 Trends in probability of the correct response given three questions

According to Figure 4, we found that for each question, the probability of correct response increases as the student's ability increases. This accords with our common sense: the greater the ability a student has, the more likely this student will the answer the question correctly.

Also, we can see the curve for question 1030 is always below the curve of question 1525. This indicate the probability of answer question 1030 correctly is lower than the probability of answering question 1525 when two student have the same ability. This also indicates the difficulty of question 1030 is bigger than the difficulty of question 1525.

Similarly, the curve for question 1525 is always below the curve of question 1574. This indicate the probability of answer question 1525 correctly is lower than the probability of answering question 1574 when two

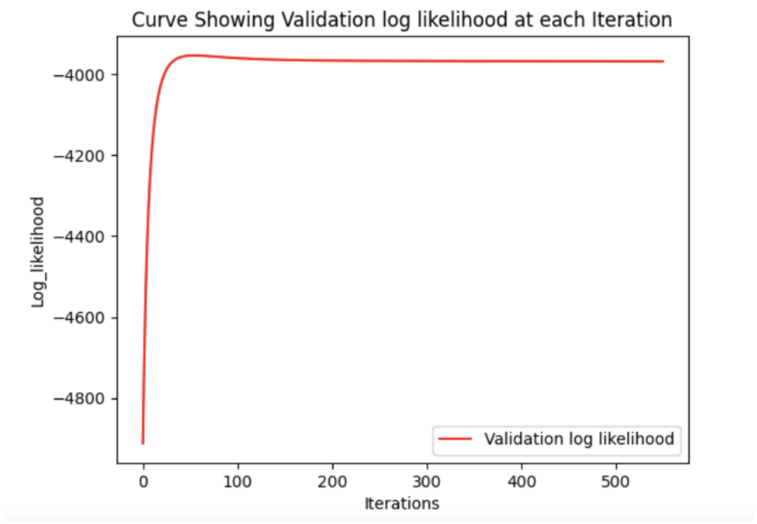

Figure 3: Validation Log-likelihood at each iteration

student have the same ability. This also indicates the difficulty of question 1525 is bigger than the difficulty of question 1574.

Hence, we can conclude the question ordered from difficult to easy is 1030,1525,1574.

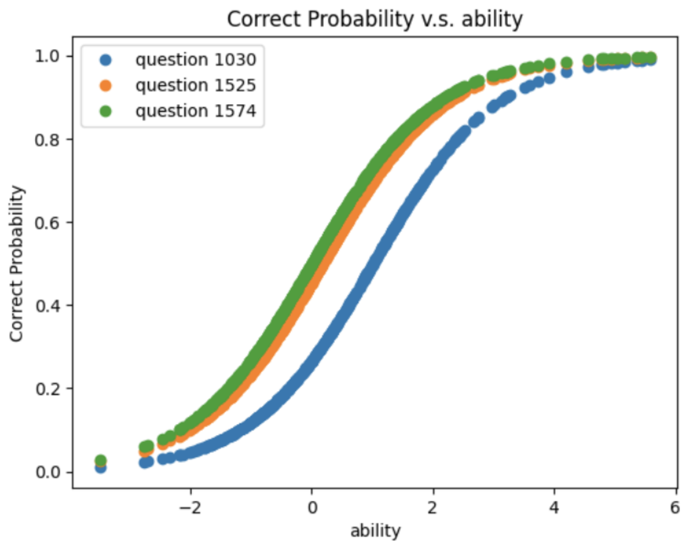

Figure 4: Student Ability v.s. Correct Probability for question 1030,1525,1574

1.3 Matrix Factorization

1.3.1 Singular-Value Decomposition

We selected different k values for $k \in \{1, 3, 5, 7, 8, 9, 11, 13\}$. Among these values, the model with k=9 achieved the highest accuracy, so it was chosen as the final model. The validation accuracy of the final model is 66.13%, and the test accuracy of the final model is 65.88%.

1.3.2 Limitation of Single-value Decomposition (SVD)

For the missing entries, we simply fill in the missing values using the average on the current items. This approach makes a strong assumption that the student would performs as well as the average for that question. This may not be accurate as it introduces bias to the data, especially for those students' ability significantly higher or lower than the average. It might not be representative of the general population.

1.3.3 Implementation of Alternating Least Squares Using Stochastic Gradient Descent

This part is shown in the `matrix_factorization.py`.

1.3.4 Tuning the Hyperparameters of ALS with SGD & Report

The selected potential hypterparameters for ALS include:

- k=[1, 3, 5, 7, 9, 11, 13]
- learning rate=[0.0005, 0.001, 0.003, 0.005, 0.01, 0.03, 0.05]
- number of iterations=[100, 1000, 10000, 50000, 100000, 200000]

The final chosen hyperparameters are k=9, learning rate=0.03, iterations=200000. And the validation accuracy for the final ALS model is 70.07%, and the test accuracy is 69.77%. The training squared-error loss and validation squared-error loss for the final model is shown in Figure 5 and Figure 6.

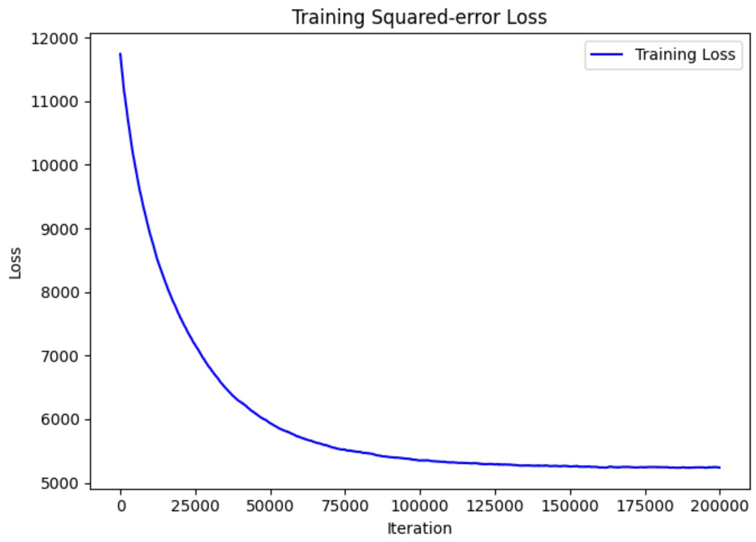

Figure 5: Training Squared-error Losses for ALS model

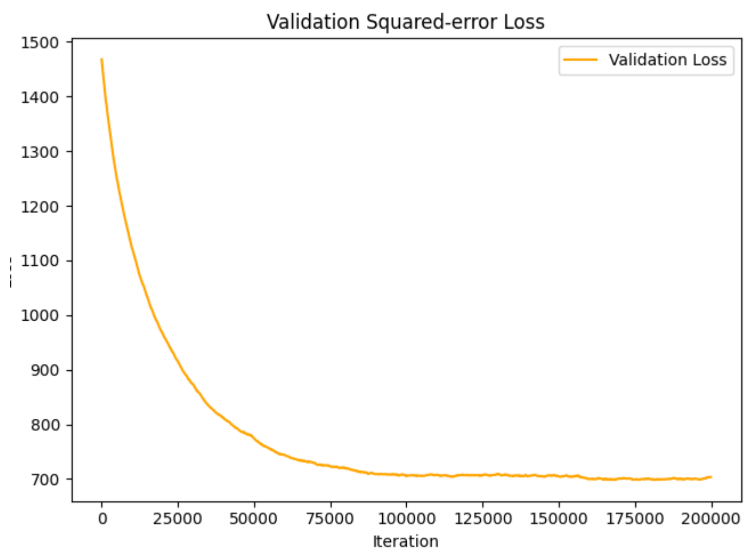

Figure 6: Valid Squared-error Losses for ALS model

2 Part B

2.1 The Modified Algorithm

The refined model is based on our Alternating Least Squares (ALS) algorithm using Stochastic Gradient Descent (SGD) as outlined in section 1.3.3. To enhance performance, we implemented two key modifications, which are shown in Figure 7 and detailed subsequently.

- **Over-fitting** is a common issue with matrix factorization techniques. In the given tasks, most of the entries in the sparse matrices being factorized are unknown, forcing the algorithm to infer great amount of missing interactions from a limited set of known interactions in the training data. This increases the risk of overfitting.

  To address this issue, we proposed two strategies. The first approach involves a **more thorough tuning of the hyperparameter** k, as the optimal k identified in section 1.3.4 is 9, which is comparatively high. A higher k indicates a greater number of latent factors, leading the model to potentially learn from the noise in the training data. However, further tuning of k across a broader range of values consistently resulted in a k value that is greater than or equal to 9.

  Recognizing that adjusting k might not effectively prevent overfitting, we decided to include **regularization terms** into the stochastic gradient descent method to control the model's complexity. Specifically, the gradients for the user matrix u is added with the regularization gradient $\lambda_{reg} \cdot u[n]$, and the gradients for the item matrix z is added with the regularization gradient $\lambda_{reg} \cdot z[q]$. The modification in coding is:

  `grad_u = -error * z[q] + lambda_reg * u[n]` and

  `grad_z = -error * u[n] + lambda_reg * z[q]`

  We compute the gradients for u and z respectively including regularization, then update u and z using the gradients with regularizer. Ideally, the regularization prevents the model from overfitting, which may capture too much noise from the training data. It allows the model to generalize better to unseen data and balance the issues of under-fitting and over-fitting.

  The regularizer λ_{reg} is also a hyperparameter we need to tune. We choose several values including 0.005, 0.01, 0.03, 0.05, 0.1, and 1, and modify the original `main` function to include a list of possible

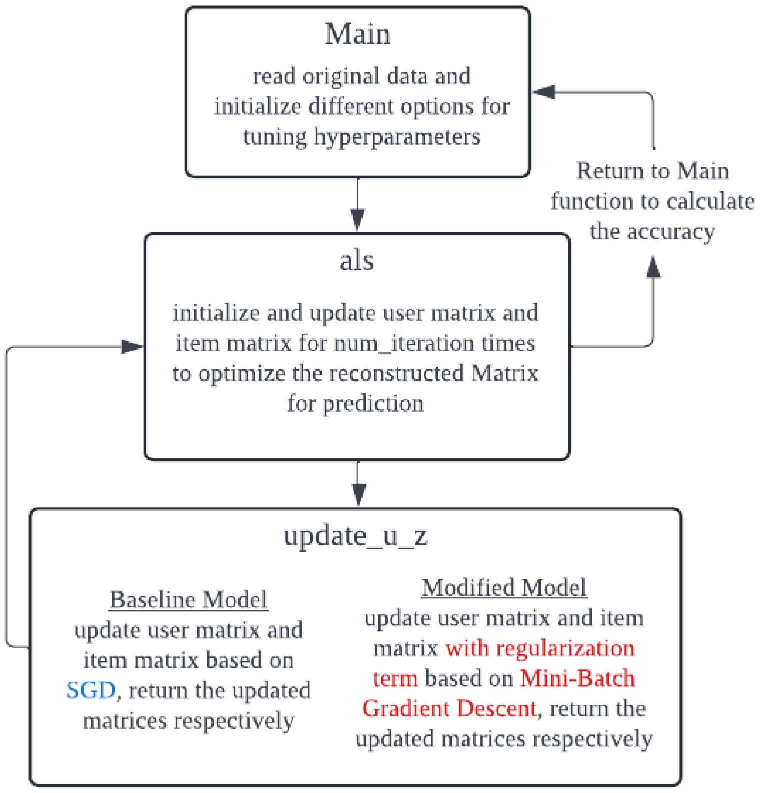
Main
read original data and initialize different options for tuning hyperparameters
Return to Main function to calculate the accuracy
als
initialize and update user matrix and item matrix for num_iteration times to optimize the reconstructed Matrix for prediction
update_u_z
Baseline Model
update user matrix and item matrix based on SGD, return the updated matrices respectively
Modified Model
update user matrix and item matrix with regularization term based on Mini-Batch Gradient Descent, return the updated matrices respectively

Figure 7: flowchart from Baseline Model to Modified Model

values for the regularizer. The best value for regularizer will be selected based on the model which have highest validation accuracy.

- The original ALS algorithm employs SGD to identify the optimal user and item matrices. As shown in Figure 5, the squared-error losses seem to converge at around the 100,000 iteration, which requires a significant time for convergence. This prolonged convergence may be attributed to the unstable updates caused by SGD method as each update is based on an individual random example from training set only. To ensure more stable updates, we revised the original function to utilize **mini-batch gradient descent** instead of SGD. Compared to SGD, this approach has the ability to reduce the variance in the updates of the user and item matrices by averaging the gradient across a mini-batch.

  When we implement the mini-batch gradient descent, we use a random subset of examples. In each epoch, we group the data set in to mini-batches of size in 8, 16, 32. For each mini-batch, we estimate the gradient using the mini-batch and update the user and item matrices based on the gradient. We repeat it until convergence.

  Similar to the newly introduced regularizer, we also tune the mini-batch size as a hyperparameter. However, during the tuning process, we found large mini-batch size always lead to more time-consuming update, we choose small fixed mini-batch size 8 to improve our algorithm. The detail is explained in the following sections.

2.2 Revised Model versus Baseline Models

2.2.1 Baseline Model

- Hyperparameters
  - k (# of latent factor): [1, 3, 5, 7, 9, 11, 13]
  - Learning Rate: [0.0005, 0.001, 0.003, 0.005, 0.01, 0.03, 0.05]
  - # of Iterations: [100, 1000, 10000, 50000, 100000, 200000]
- The selected best hyperparameters for the baseline model are k=9, learning rate= 0.03, and # of iterations=200000 with the final validation accuracy of 70.07% and test accuracy of 69.77%.
- The training validation square-error losses are shown in Figure 5 and 6.

2.2.2 Revised Model with Regularization Only

- Hyperparameters
  - k (# of latent factor): [3, 5, 7, 9, 11]
  - Learning Rate: [0.005, 0.01, 0.03, 0.05, 0.1]
  - # of Iterations: [10000, 50000, 100000, 200000]
  - Mini-Batch Size
  - Regularizer: [0.01, 0.05, 0.1, 1]
- The selected best hyperparameters for the revised model are k=7, learning rate= 0.03, # of iterations=200000, and regularizer=0.01 with the final validation accuracy of 70.42% and test accuracy of 69.91%.
- The training validation square-error losses are shown in Figure 10.

2.2.3 Baseline Model with 1000 Iterations

- Hyperparameters
  - k (# of latent factor): [3, 5, 7, 9, 11]
  - Learning Rate: [0.0005, 0.001, 0.003, 0.005, 0.01, 0.03, 0.05, 0.1]
  - # of Iterations: [1000]
- The selected best hyperparameters for the baseline model are k=3, learning rate= 0.05, and # of iterations=1000 with the final validation accuracy of 42.17% and test accuracy of 42.14%.
- Due to prolonged running time, we only plot the training and validation squared-error losses for the baseline model with 1000 iteration (shown in Figure 8).

2.2.4 Revised Model with Regularization and Mini-batch

- Hyperparameters
  - k (# of latent factor): [3, 5, 7, 9]
  - Learning Rate: [0.01, 0.03, 0.05]
  - # of Iterations: [1000]
  - Mini-Batch Size: 8
  - Regularizer: [0.01]
- The selected best hyperparameters for the revised model are k=3, learning rate= 0.03, regularizer=0.01, , and # of iterations=1000 with the final validation accuracy of 69.38% and test accuracy of 68.16%.
- Due to prolonged running time, we only plot the training and validation squared-error losses for the revised model with 1000 iteration and mini-batch size=8 (shown in Figure 9). We only test regularizer=0.01 here based on the result of tuning hyperparameters from model 2.2.2, due to the expensive update.

Comparing Figure 8 and Figure 9, there is a more dramatic initial drop in both training and validation losses for the revised model while the training and validation squared-error losses for the baseline model consistently decrease as the number of iterations increases. The steeper decline in squared-error losses for the revised model implies that incorporating mini-batch and regularization potentially helps the model to learn more quickly and effectively.

However, the test accuracy did not show a significant improvement, with only a slight increase to 69.91% in section 2.2.2 from 68.16% in section 2.2.1 when applying regularization alone. Moreover, incorporating mini-batch gradient descent significantly extended the runtime, allowing us to only complete the tuning of other hyperparameters with the number of iterations set to 1000 and a mini-batch size of 8. Fortunately, after employing mini-batch gradient descent, the test accuracy enhanced markedly from 42.14% (section 2.2.3) to 68.16% (section 2.2.4). This enhancement in test accuracy, as well as the validation accuracy, suggests that the introduction of mini-batch gradient descent facilitates faster convergence.

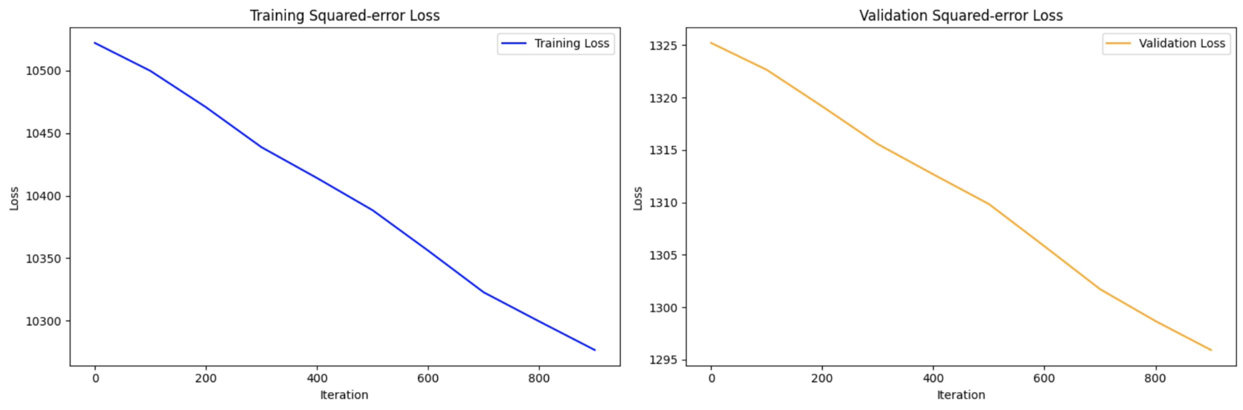

Figure 8: Training and Validation Squared-error Losses for Baseline Model with 1000 iterations

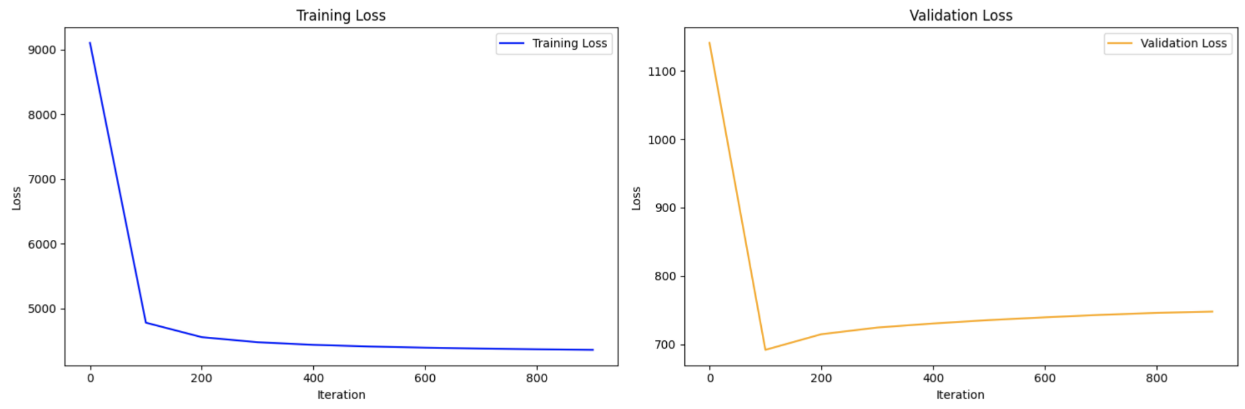

Figure 9: Training and Validation Squared-error Losses for Revised Model with Mini-batch and Regularization with 1000 iterations

2.3 Performance of Revised Model

The final chosen hyperparameters for the revised model with regularization (see section 2.2.2 and Figure 10) are k=3, learning rate=0.03, iterations=200000, $\lambda_{reg} = 0.01$. And the validation accuracy for the final ALS model is 70.42%, and the test accuracy is 69.91%. Both of them are slightly higher than the baseline model in section 2.2.1.

Our model's performance did not significantly improve after the introduction of regularization, and tuning the fully revised model with mini-batch gradient descent proved to be too time-consuming. Although the model described in section 2.2.4 achieved a higher test accuracy than that in section 2.2.3, the test accuracy did not exceed 70%, implying that while convergence was reached more efficiently, the impact on test accuracy was minimal.

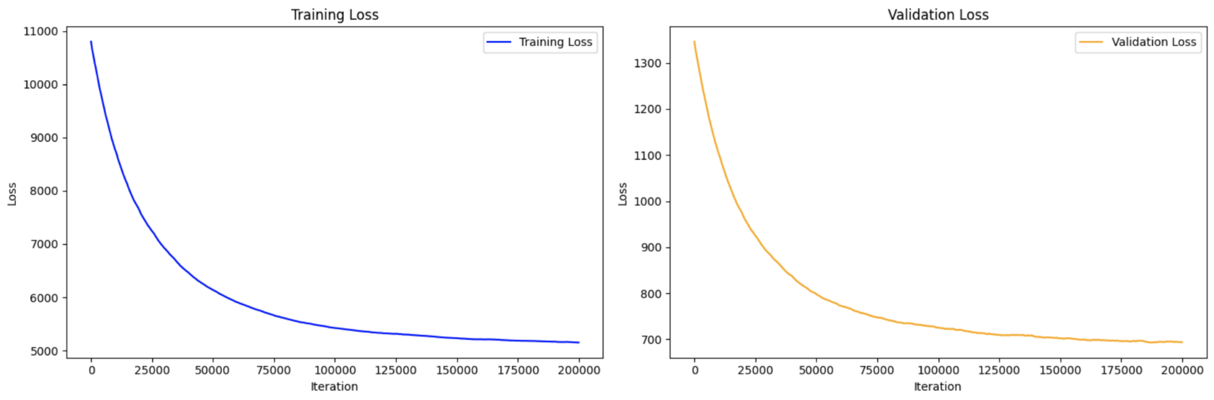

Figure 10: Training and Validation Squared-error Losses for Improved ALS Model with Regularization

One reason is that the regularization is an effective tool to avoid overfitting that may or may not increase test accuracy. The help of regularization in our revised model is limited might be attributed to the fact that the complexity of original coefficients of baseline model is not high. Therefore, there is only a slight enhancement in the test accuracy.

Another reason is that we did not include any additional unseen features that may reveal more important pattern of the relationship between these features and the answer of student.

2.4 Model Limitation

One potential limitation is that the model is not good at handling the sparsity. Since the matrix factorization assumes there are enough data to learn accurate representations of users and items. In sparse data sets, where users have interacted with very few items (or vise versa), the model may struggle to learn meaningful factors, leading to poor performance. This issue arises since the matrix factorization model relies on historical interaction data to learn latent feature for users and items. The model tends to perform poorly if we want to predict some data without or with very small interaction with our training data.

To address this limitation, one possible modification is to incorporate content-based features into the model. For instance, we could use the subject in question metadata to compute initial factor vectors for item matrices, allowing the model to make more accurate decisions. The algorithm may be improved by adding a content-based filtering method, leveraging both interaction data and feature data for prediction. Introducing the content-based features could help us improve the interaction between data to be predicted and training data.

2.5 Contribution

Nuo Xu finished 'Collaborative filtering with k-Nearest Neighbor in Part A. Jiawei Gong finished 'Item response model' in Part A. Jiaxi Li finished 'Matrix Factorization' in Part A. Nuo Xu and Jiaxi Li also finished the implement of Part B.