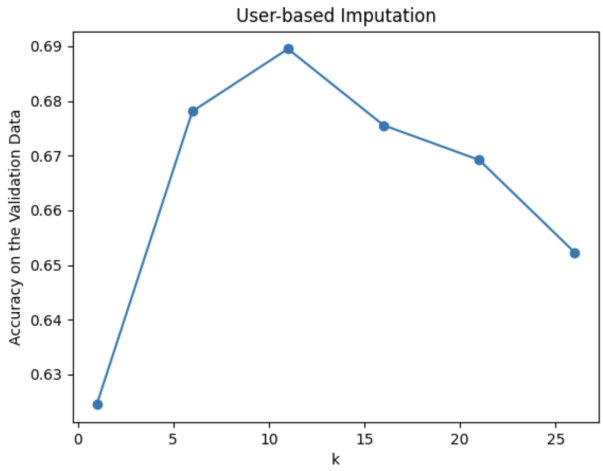
CSC311 Project Part A

Luke Zhang, Ximing Shen, Zhijing Zhang

July 2024

1 k-nearest neighbors

a. The plot and validation accuracies for user-based imputation are reported below.

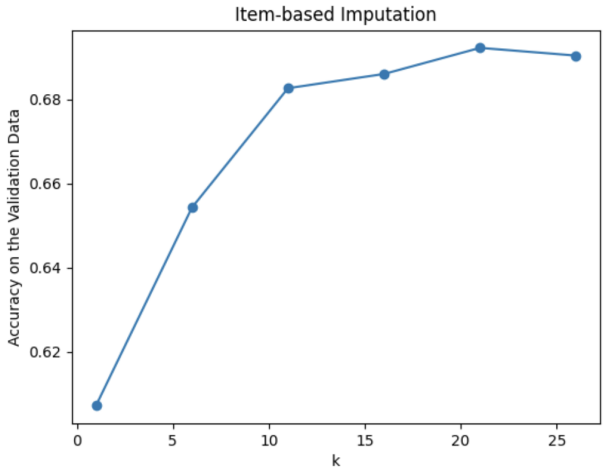



```
Validation Accuracy for 1 is: 0.6244707874682472
Validation Accuracy for 6 is: 0.6780976573525261
Validation Accuracy for 11 is: 0.6895286480383855
Validation Accuracy for 16 is: 0.6755574372001129
Validation Accuracy for 21 is: 0.6692068868190799
Validation Accuracy for 26 is: 0.6522720858029918
```

b. From the above graph and the printed validation accuracies, we can conclude $k^* = 11$ and its accuracy is 0.6895286480383855.

c. The underlying assumption on item-based collaborative filtering is that if question A has received the same correct and incorrect responses from other students as question B, then the correctness of a specific student's response to question A matches their response to question B.

The plot and validation accuracies for item-based imputation are reported below.




```
Validation Accuracy for 1 is: 0.607112616426757
Validation Accuracy for 6 is: 0.6542478125882021
Validation Accuracy for 11 is: 0.6826136042901496
Validation Accuracy for 16 is: 0.6860005644933672
Validation Accuracy for 21 is: 0.6922099915325995
Validation Accuracy for 26 is: 0.69037538808919
```

From the above graph and the printed validation accuracies, we can conclude $k^* = 21$ and its accuracy is 0.6922099915325995.

d. With $k^* = 11$ for user-based collaborative filtering and $k^* = 21$ for item-based collaborative filtering, we obtained the below test accuracies.

```
Test Accuracy for user-based collaborative filtering is: 0.6841659610499576
Test Accuracy for item-based collaborative filtering is: 0.6816257408975445
```

We can thus conclude the user-based collaborative filtering performs better.

e. Below are 2 potential limitations of knn for the task we are given

(a) In both user-based and item-based collaborative filtering, the underlying assumption might be too simplistic to accurately capture real-life complexities.

(b) Notice that each student vector has a dimension of 1774 and each question vector has a dimension of 542. This high dimensionality may cause most sample points having approximately the same distance due to the curse of dimensionality, which make the inference less effective since we used euclidean distance for the algorithm.

2 Item Response Theory

(a) We are given

$\mathbf{C}$	the sparse matrix
β_j	difficulty of question j
θ_i	ith student's ability

For this question we will assume there are N students and D questions. Then the probability that question j is answered correctly by student i is denoted as:

$$P(c_{ij} = 1|\theta_i, \beta_j) = \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)} = \exp(\theta_i - \beta_j)\sigma(\beta_j - \theta_i) = \exp(\theta_i - \beta_j)\sigma_{ij}$$

where σ_{ij} will be used to abbreviate $\sigma(\beta_j - \theta_i)$. Then we can generalize the probability to all questions for all students.

$$\begin{aligned} P(\mathbf{C}|\theta,\beta) &= \prod_{i=1}^{N}\prod_{j=1}^{D} P(c_{ij}|\theta_i,\beta_j) \\ &= \prod_{i=1}^{N}\prod_{j=1}^{D} \left[P(c_{ij}=1|\theta_i,\beta_j)\right]^{c_{ij}} \left[1-P(c_{ij}=1|\theta_i,\beta_j)\right]^{(1-c_{ij})} \\ &= \prod_{i=1}^{N}\prod_{j=1}^{D} \left[\exp(\theta_i-\beta_j)\sigma_{ij}\right]^{c_{ij}} \left[\sigma_{ij}\right]^{(1-c_{ij})} \end{aligned}$$

Next we will derive the log likelihood.

$$\begin{aligned} \ell(\theta,\beta) &= \log[P(\mathbf{C}|\theta,\beta)] \\ &= \sum_{i=1}^{N}\sum_{j=1}^{D} c_{ij}\log\left[\exp(\theta_i-\beta_j)\sigma_{ij}\right] + (1-c_{ij})\log(\sigma_{ij}) \\ &= \sum_{i=1}^{N}\sum_{j=1}^{D} c_{ij}\left[\theta_i-\beta_j+\log(\sigma_{ij})\right] + (1-c_{ij})\log(\sigma_{ij}) \\ &= \sum_{i=1}^{N}\sum_{j=1}^{D} c_{ij}(\theta_i-\beta_j) + \log(\sigma_{ij}) \end{aligned}$$

To find the optimal θ, β we need to gradient descent that maximizes the log-likelihood. To do this, we will first find the derivative with respects to θ and β.

$$\begin{aligned} \frac{d\ell}{d\theta_i} &= \sum_{j=1}^{D} c_{ij} + \frac{1}{\sigma_{ij}}\frac{d}{d\theta_i}(\sigma_{ij}) \\ &= \sum_{j=1}^{D} c_{ij} + (1+\exp(\theta_i-\beta_j))\cdot\left(-\frac{\exp(\theta_i-\beta_j)}{(1+\exp(\theta_i-\beta_j))^2}\right) \\ &= \sum_{j=1}^{D} c_{ij} - \frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)} \end{aligned}$$

$$\begin{aligned} \frac{d\ell}{d\beta_j} &= \sum_{i=1}^{N} -c_{ij} + \frac{1}{\sigma_{ij}}\frac{d}{d\beta_j}(\sigma_{ij}) \\ &= \sum_{i=1}^{N} -c_{ij} + (1+\exp(\theta_i-\beta_j))\cdot\frac{\exp(\theta_i-\beta_j)}{(1+\exp(\theta_i-\beta_j))^2} \\ &= \sum_{i=1}^{N} -c_{ij} + \frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)} \end{aligned}$$

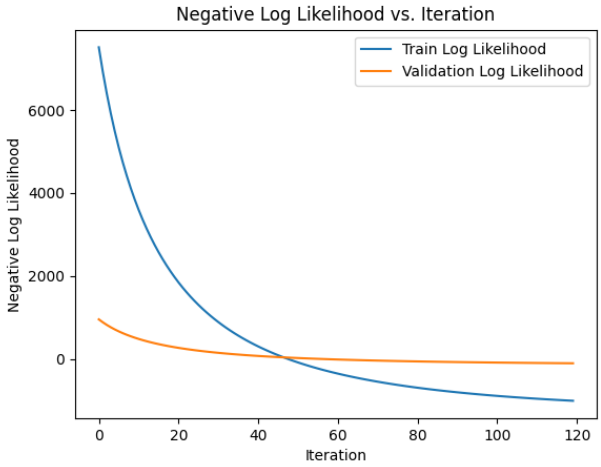
Using our calculations above, we finally arrive at our gradient ascent update rules (note we are trying to maximize likelihood instead of minimize loss):

$$\theta_i \leftarrow \theta_i + \eta\nabla_{\theta_i}\ell(\theta) = \theta_i + \eta\sum_{j=1}^{D} c_{ij} - \frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}$$

$$\beta_j \leftarrow \beta_j + \eta\nabla_{\beta_j}\ell(\theta) = \beta_j - \eta\sum_{i=1}^{N} c_{ij} - \frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}$$

(b) `hyperparameters: num_iteration = 120, learning_rate = 0.001`

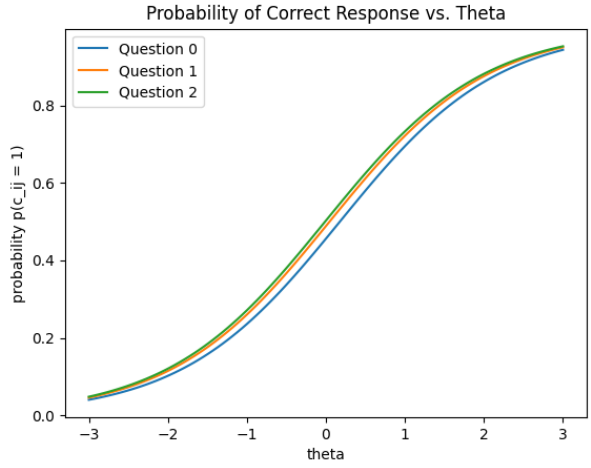
Below is the training curve that shows the training and validation log-likelihoods as a function of iteration.



(c) The final validation and test accuracies are reported below.

```
val accuracy:  0.7081569291560824
test accuracy:  0.7011007620660458
```

(d) We selected the questions 1, 2, 3 and used the optimal θ, β to produce the plot that shows the probability of the correct response $p(c_{ij} = 1)$ as a function of θ given a question j.



Each curve in the graph represents the probability of getting a question correct as θ increases. Specifically, the blue curve corresponds to the probability of getting question 1 correct, the orange curve corresponds to the probability of getting question 2 correct and the green curve corresponds to the probability of getting question 3 correct.

When θ is small, the probability of $p(c_{ij} = 1)$ is close to 0. As θ increases, so does $p(c_{ij} = 1)$ which approaches 1. In the context of this question, this means that as the student's ability, denoted by θ, increases, then the probability of getting question c_{ij} correct, that is $p(c_{ij} = 1)$ also increases.

A unique feature we see in all curves is their flattened-out tails on both ends and a step increase towards the center of the curve. This shape is explained because we used the sigmoid function to calculate the probability ie. $\sigma(\theta - \beta)$, which produces the same flattened-out ends and increases towards the center. Additionally, while the curves remain close together for all theta, we see that question 2 has the highest probability of being answered correctly, question 3 has the lowest probability of being answered correctly and question 2 lies just in between questions 1 and 3. This suggests that while the questions all have a higher chance of being answered correctly as students' abilities increase, question 1 may be an easier question than question 2, and question 2 is easier than question 3.

3 Matrix Factorization

(a) The first image shows accuracy for each k and the best k is 25, the second image shows the validation and test performance with k=25.

```
k:5, acc:0.659046006209427
k:25, acc:0.6594693762348293
k:40, acc:0.65001411233418
k:60, acc:0.6453570420547559
k:80, acc:0.64422805532035
Best k for SVD: 25 with acc: 0.6594693762348293
```

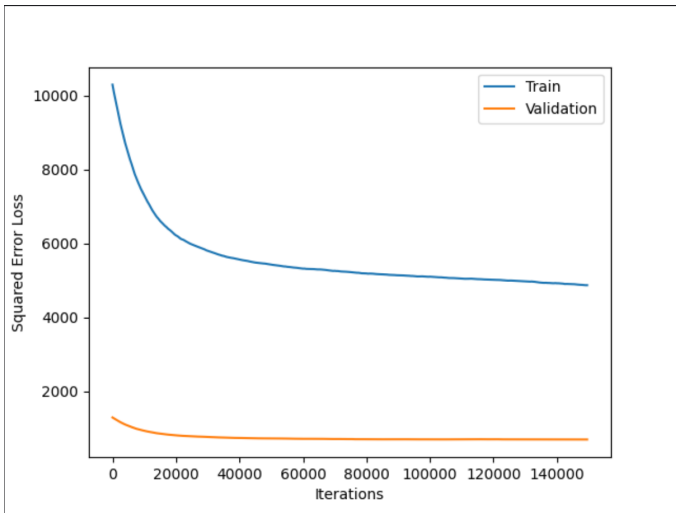
```
accuracy with validation data(svd):  0.6594693762348293
accuracy with test data(svd):  0.6556590460062094
```

(b) One limitation of using SVD is that it does not inherently handle missing entries in the matrix, but there are missing entries for this task, that is (NaN) values. In this case, we fill all missing entries with the mean on the current item. However, the SVD decomposition and its reconstruction do not differentiate between originally observed data and imputed data, which may lead to the model overfitting to the imputed values, which do not accurately represent true user preferences, leading to less accurate predictions when the matrix is reconstructed.

(d) The best choice of hyperparameter and parameter is k = 25, learning rate = 0.05, number of iterations=150000

```
max is  k:25 lr:0.05 iteration:150000 acc:0.7013830087496472
```

(e) graph for squared err lost for train and validation, and accuracy for validation and test data.




```
accuracy with validation data(als):  0.7013830087496472
accuracy with test data(als):  0.7042054755856618
```

4 Ensemble

i Our ensemble process is provided below.

We first generated three bootstrapped datasets, `data_a, data_b, data_c` from the original data. Each bootstrap is sampled randomly and with replacement.

Next, we train three different matrix factorization models, where `matrix_a` uses `data_a`, `matrix_b` uses `data_b` and `matrix_c` uses `data_c`. We used the same hyper-parameters for all three matrix factorization models, that is, they all use the best hyper-parameters derived from Part A, Matrix Factorization part (a), where `k = 25, lr = 0.05, num_iteration = 150000`.

Averaging the predicted matrices from these three models to form an ensemble model, we evaluate this new model on validation and test datasets.

ii We obtain better performance with our new ensemble method. Our validation and test accuracy from our ensemble method are both higher than the base model accuracy of Matrix Factorization.

Method	**Validation Accuracy**	**Test Accuracy**
Matrix Factorization (als)	0.7013830087496472	0.7042054755856618
Ensemble of Matrix Factorization	0.7022297488004516	0.7581145921535422

iii We most likely obtained better performance from using the ensemble of models due to:

(a) Uniqueness of features learned: Because each model in the ensemble was trained on slightly different data (due to bootstrapping), they might learn slightly different aspects of the underlying patterns in the data. When these models are combined by averaging, errors made by one model might be corrected by another, leading to a more accurate final prediction.

(b) Bias-Variance trade-off: The individual base models (matrix factorization models) might have a certain bias (tendency to underfit) or variance (tendency to overfit). By averaging the predictions of multiple models, you reduce the overall variance of the model, while maintaining (or even slightly reducing) the bias. This trade-off typically leads to improved accuracy.

Overall, by using an average of three models, we reduce overfitting and increase our chances of learning the correct features.

5 Part B

1. We want to improve the performance of our matrix factorization model by introducing weights and regularization into the objective function.

   (a) **Introducing weights.**
   The current approach treats all entries in the sparse matrix C equally, but certain entries may capture more significant information, and we want to emphasize these during training. Therefore, we introduce a weighting scheme that scales how much each entry contributes to the objective function.

   Below are the factors we considered while setting the weight value:

   i. Correct rate of questions

   A question with an extremely high correct rate (/extremely low correct rate) implies that it is relatively easy (/difficult) and might not be as informative about a student's true ability.

   For a specific question, we consider the difference between its correct rate and the mean correct rate of all questions to determine whether it is an outlier.
   The further its correct rate is away from the mean, the lower weights are assigned to it, and thus we deemphasize the importance this question by decreasing its contribution to the loss function.
   In code, this means

   ```
   weights_acc[m] = 1 - (np.abs(mean - weights_acc[m]))
   ```

   ii. The frequency of students and questions

   This idea is inspired by an online source "For example, frequent items (Extremely popular YouTube videos) or frequent queries (for example, heavy users) may dominate the objective function. (Google Developer)".

   Thus, for active students who answer many questions, and for popular questions answered by many students, we want to deemphasize their contribution to the objective function.
   This prevents these frequent data points from dominating the model and ensures a more balanced training process.

   Specifically, we introduce a weight w_{mn} for each student question pair (n, m), which accounts for both the frequency of the student and the question.
   More specifically, since we want to reduce the effect of frequency student and questions, the weight is the inverse of the product of the frequency of the student and the frequency of the question.
   For this whole part, we have

   ```
   1.    for key in dictionary:
   2.       weights_acc[key] = 1 - (np.abs(mean - weights_acc[key]))

   3.    for m in set(question_id):
   4.        w_1 = weights_acc[q]
   5.        for n in set(user_id):
   6.            w_2 = 1 / (question_count[m] * user_count[n])
   7.            weight = w_2 * w_1
   8.            weights[m][n] = 1 - (1 - weight) * 0.05
   ```

   We'll first calculate the correct rate for each question (Line 1-2). In line 3-6, we actually calculate the weight for each (student, question) pair. The weight for each w_{mn} is just
   correct rate of question m / (frequency of student n $*$ frequency of question m).
   However, since we don't want this weight to influence the inference too much so that the model overfit, we'll scale it to make it closer to one as we did in line 8.
   Thus, the formula in line 8 is the weight we added to the model.

(b) **Introducing regularization**

We'll also add a regularization term to reduce over-fitting of matrices U and Z. With newly added weights and regularization, the loss function becomes

$$L(\mathbf{u}, \mathbf{z}) = \frac{1}{2} \sum_{n,m} W_{nm} \left(C_{nm} - \mathbf{u}_n^\top \mathbf{z}_m\right)^2 + \frac{\lambda}{2} \left(\|\mathbf{u}\|^2 + \|\mathbf{z}\|^2\right)$$

$$\frac{\partial L}{\partial \mathbf{u}_i} = -\sum_m W_{im} \left(C_{im} - \mathbf{u}_i^\top \mathbf{z}_m\right) \mathbf{z}_m + \lambda \mathbf{u}_i$$

$$\frac{\partial L}{\partial \mathbf{z}_j} = -\sum_n W_{nj} \left(C_{nj} - \mathbf{u}_n^\top \mathbf{z}_j\right) \mathbf{u}_n + \lambda \mathbf{z}_j$$

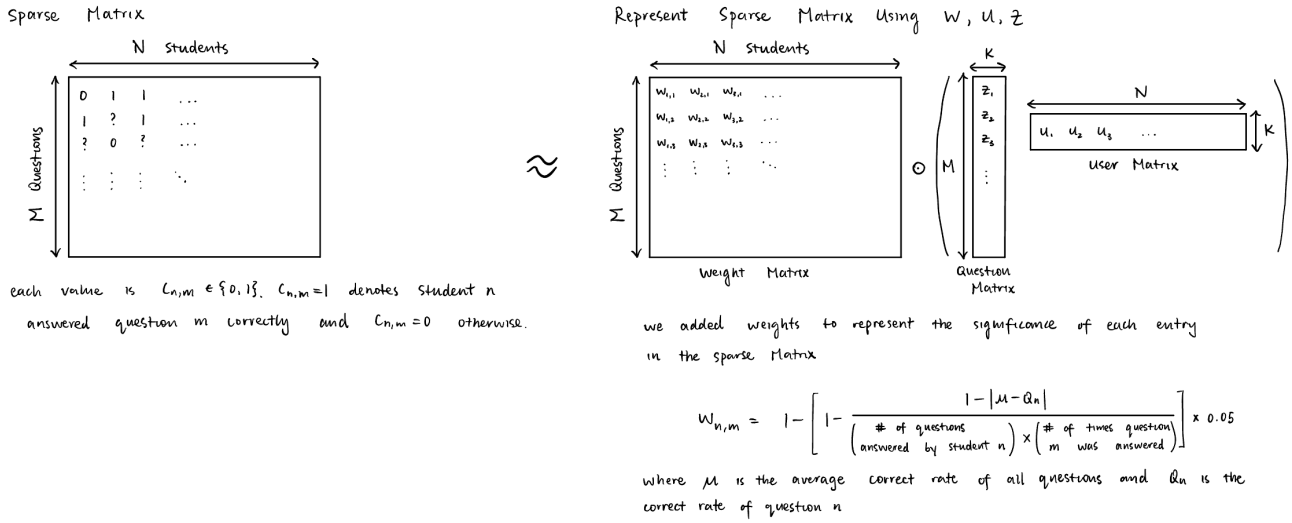
Also, we need to modify the *update_u_z* function with the new update rule:

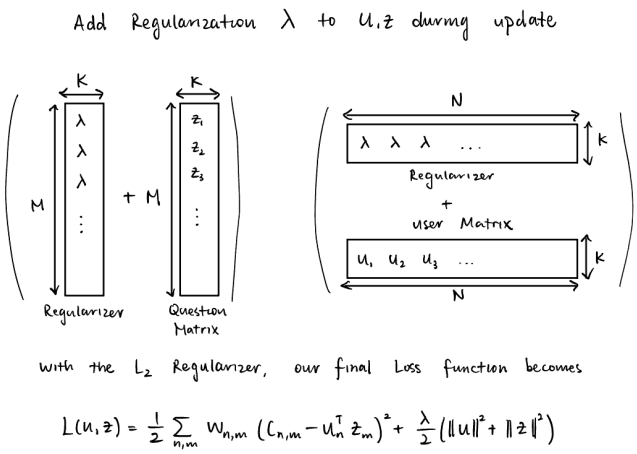
$$\mathbf{u}_i \leftarrow \mathbf{u}_i + \eta \sum_m w_{im} \left(C_{im} - \mathbf{u}_i^\top \mathbf{z}_m\right) \mathbf{z}_m + \lambda \mathbf{u}_i$$
$$\mathbf{z}_j \leftarrow \mathbf{z}_j + \eta \sum_n w_{nj} \left(C_{nj} - \mathbf{u}_n^\top \mathbf{z}_j\right) \mathbf{u}_n + \lambda \mathbf{z}_j$$

The updated *update_u_z* is

```
def update_u_z(train_data, lr, u, z, lam, weights):
    i = np.random.choice(len(train_data["question_id"]), 1)[0]
    c = train_data["is_correct"][i]
    n = train_data["user_id"][i]
    q = train_data["question_id"][i]
    weight = weights[q][n]
    err = weight * (c - np.dot(u[n], z[q]))
    u[n] += lr * (err * z[q] - lam * u[n])
    z[q] += lr * (err * u[n] - lam * z[q])
    return u, z
```

2. Below is the visualization of our modifications.





3. (a) Comparison of two models

While considering the same sets of different hyperparameters, our modified model reaches it's highest accuracy with a different setting of hyperparameters compare to the baseline model.

Table 1 shows the corresponding values for each model. We observe a slight decrease in both validation and test accuracy. This decline could be due to the weights we chose not adequately capturing the relationships between data points, or it could indicate that the regularization function we used is not optimal for this particular model.

	Baseline Model	**Modified Model**
k	25	80
Learning rate (lr)	0.05	0.05
# iterations	150000	150000
Training Accuracy	0.7013830087496472	0.7019475021168501
Lambda	/	0.01
Validation Accuracy	0.7013830087496472	0.6975726785210273
Test Accuracy	0.7042054755856618	0.6971493084956252

Table 1: Comparison of Two Models

We further examined these two models by comparing the change in accuracy for training data, with learning rate = 0.05, number of iterations = 150000, and the lambda chose for l2 regularization for modified model is 0.01.

The result is illustrated in table 2.

k	baseline_model_acc	modified_model_acc
5	0.6950324583686142	0.6930567315834039
25	0.7056167090036692	0.696725938470223
40	0.6974315551792266	0.7029353655094552
60	0.698842788597234	0.7011007620660458
80	0.7013830087496472	0.7067456957380751

Table 2: comparison of accuracy for training data

From this table, we see that the baseline model has the highest accuracy when `k=25` and the modified model has the highest accuracy when `k=80`. This suggests that the modified model performs better on more complex representations of data, likely due to the regularizing term reducing `U` and `Z` values allowing the model to avoid overfitting on complex data.

(b) **Optimization / Regularization**
Table 3 compares the accuracies with weights only and regularization only.

	Baseline Model	Baseline Model with Weights only	Baseline Model with l2 Regularization only	Modified Model
Training Accuracy	0.7013830087496472	0.7008185153824442	0.7039232289020604	0.7019475021168501
Validation Accuracy	0.7013830087496472	0.6972904318374259	0.6996895286480384	0.6975726785210273
Test Accuracy	0.7042054755856618	0.6971493084956252	0.6971493084956252	0.6971493084956252

Table 3: Accuracy for Different Models

For training accuracy, we can see that the weights-only model slightly decreases and the regularization-only model slightly increases.
For validation accuracy, we can see both models (columns 2 & 3) decrease. Since the weights-only model has a lower validation accuracy, we believe the addition of weights is more responsible for the reduction of validation accuracy in the final modified model.
For test accuracy, both the wights-only model and the regularization-only model slightly decreased compared to the baseline model, but they obtained the same value.

4. **Limitation 1:** Reducing weights for students who answer many questions can introduce bias, particularly if their response patterns reflect general behaviour, and those who answered very few questions may be biased(e.g. they answer all questions extremely well/poorly). Undervaluing these patterns can lead to biased predictions for students who answer fewer questions.

   Similarly, decreasing weights for frequently answered questions can cause the model to miss important patterns. Because sometimes, a frequently answered question may be well designed that can reflect a student's understanding to certain topics well, reducing weights to these questions will skew the model's understanding and resulting in less accurate predictions.

   Limitation 2: Our model may struggle to make accurate predictions for new questions that were not present in the training data, as it fails to capture any patterns for these unseen questions.

   Limitation 3: Our model currently doesn't account for the possibility of guessing in the data. While adding weights helps to emphasize questions that best represent a student's true abilities, it may also inadvertently include answers that were guessed. These guessed answers can introduce noise, leading to inaccuracies in assessing a student's actual skill level.

   Extension 1: We can incorporate subject and student data into the probability of question correctness. For example, we may be able to learn patterns such as certain genders having better performances in particular categories, such as females having higher performances in subject A or males having higher performances in subject B. Subject and student data also allow for further considerations such as if the student is older they may have more experience, thus they may have a higher chance of answering a question correctly, or students at certain ages may learn specific subjects faster/slower. Incorporating student and subject considerations may also allow the model to perform better when filling in missing values in the sparse matrix (unanswered questions) because we can pick out similar students or similar question types to make a more accurate guess on what to fill this missing value with.

   Extension 2: Matrix Factorization does not inherently have a method of filling out missing values. Adding onto the previous extension, we can use other models such as kNN to pick out similar students and similar question types. This is just one example where we may potentially use a different model to fill in where the Matrix Factorization does not perform well. However, we can extend this idea further, and use other models/algorithms to help us pick out hyperparameters such as k or learning rate. This may reduce computation power (since storing matrices can be memory-consuming) or computation time (since minimizing U and Z is not a convex problem).

6 Contribution

We all discussed the idea together and viewed each other's written work afterwards.
For the write-up:
Luke: item response theory, ensemble, part b question 2
Ximing: knn, item response theory, part b question 1 & 3
Zhijing: matrix factorization and ensemble, part b question 3 & 4

7 Citation

1. "Matrix Factorization | Machine Learning | Google for Developers." Google, Google Developers, developers.google.com/machine-learning/recommendation/collaborative/matrix. Accessed 8 Aug. 2024.