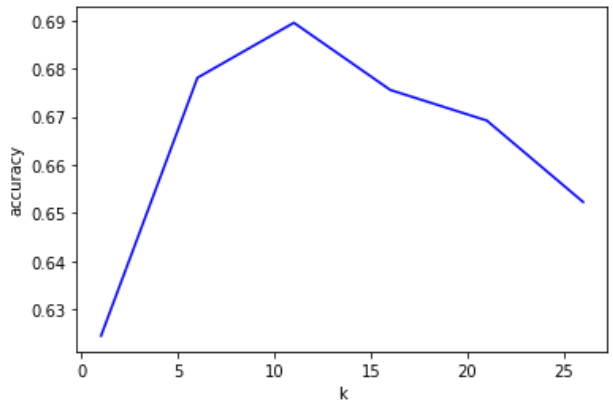
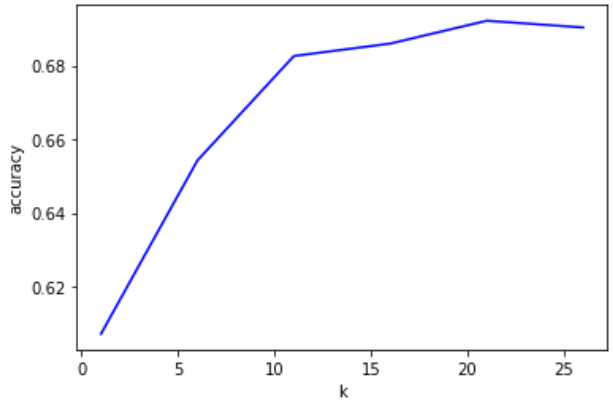
Question 1

(a) (b) User-based knn




```
The chosen k is 11 and the test accuracy is 0.6841659610499576
```

(c) Item-based knn




```
The chosen k is 21 and the test accuracy is 0.6816257408975445
```

(d) Comparison

1. The test accuracy from user-based collaborative filtering is slightly better than test accuracy from item-based collaborative filtering.
2. Besides, the computational time that user-based algorithm takes is much less than the computational time that item-based algorithm takes.

Therefore, user-based collaborative filtering performs better.

(e) Potential Limitations

1. The computation is expensive. The shorter one (user-based knn) still takes a long time compared to other algorithms (e.g. irt in q2).
2. The dimension of data is high, and in high dimensions, most points have approximately the same distance. So the nearest distance might not be useful.

Question 2

(a)

$$p(C \mid \theta, \beta) = \prod_{i=1}^{N} \prod_{j=1}^{M} p(C_{ij} \mid \theta_i, \beta_j)$$

Notice that

$$p(C_{ij} = 1 \mid \theta_i, \beta_j) = \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)}$$

Then

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

Thus

$$p(C_{ij} \mid \theta_i, \beta_j) = \left(\frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)}\right)^{C_{ij}} \left(\frac{1}{1 + \exp(\theta_i - \beta_j)}\right)^{1 - C_{ij}}$$

Therefore,

$$p(C|\theta,\beta)=\prod_{i=1}^{N}\prod_{j=1}^{M}\left(\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}\right)^{C_{ij}}\left(\frac{1}{1+\exp(\theta_i-\beta_j)}\right)^{1-C_{ij}}$$

$$l(\theta,\beta)=\sum_{i=1}^{N}\sum_{j=1}^{M}C_{ij}\log\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}+(1-C_{ij})\log\frac{1}{1+\exp(\theta_i-\beta_j)}$$

$$=\sum_{i=1}^{N}\sum_{j=1}^{M}C_{ij}(\theta_i-\beta_j-\log(1+\exp(\theta_i-\beta_j)))+(1-C_{ij})(-\log(1+\exp(\theta_i-\beta_j)))$$

$$=\sum_{i=1}^{N}\sum_{j=1}^{M}C_{ij}\cdot\theta_i-C_{ij}\beta_j-C_{ij}\log(1+\exp(\theta_i-\beta_j))+C_{ij}\log(1+\exp(\theta_i-\beta_j))-\log(1+\exp(\theta_i-\beta_j))$$

$$=\sum_{i=1}^{N}\sum_{j=1}^{M}C_{ij}(\theta_i-\beta_j)-\log(1+\exp(\theta_i-\beta_j))$$

$$\frac{\partial l}{\partial\theta_i}=\sum_{j=1}^{M}C_{ij}-\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}$$

$$\frac{\partial l}{\partial\beta_j}=\sum_{i=1}^{N}-C_{ij}+\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}$$

(b)

In this part, I user the following hyperparameters:

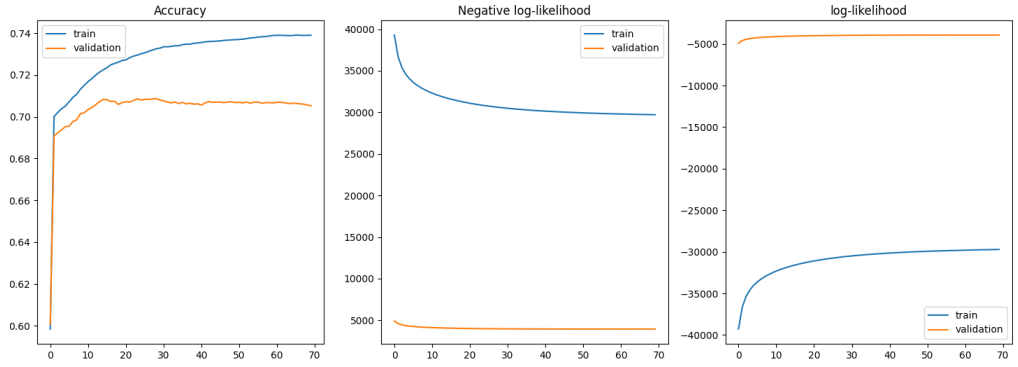
Number of Iterations: 70

Learning Rate: 0.05

Initialize theta: All 0s

Initialize beta: All 0s

Training process is shown as below:



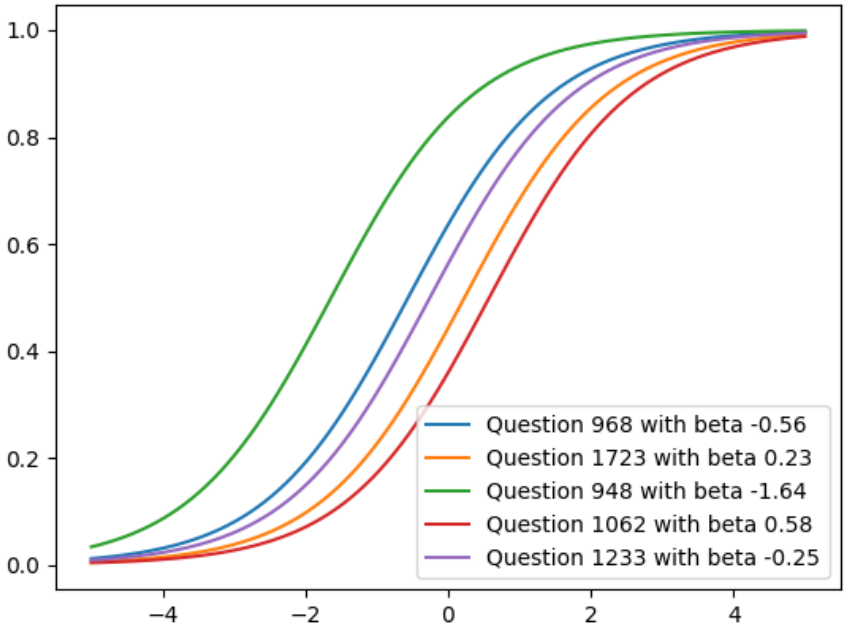
(c)

Final Train Accuracy is: 0.7390100197572679

Final Validation Accuracy is: 0.7053344623200677

Final Test Accuracy is: 0.7047699689528648

(d)



In this part, I use different as $p(c_{ij})$ as a function of θ given 5 different questions. We can see the shape of those 5 curves are in a shape of sigmoid function, which means it is a transformation of sigmoid function. The reason why it looks like sigmoid function is $p(c_{ij}) = sigmoid(\theta_i - \beta_j)$.

We can consider θ as the ability of the students and the β as the difficulty of the question. Then suppose we have the same value of θ, that is we have the same value on x-axis, which means the students have same ability, the higher the value of y-axis is, the easier the question is. This is because the higher value of y-axis represents a higher probability that the students can answer the question correctly. From the graph above, we also can notice the curve of the question with lower β(easier) is at the top of this graph while the ones with larger β(harder) is at the bottom.

Question 3

a)

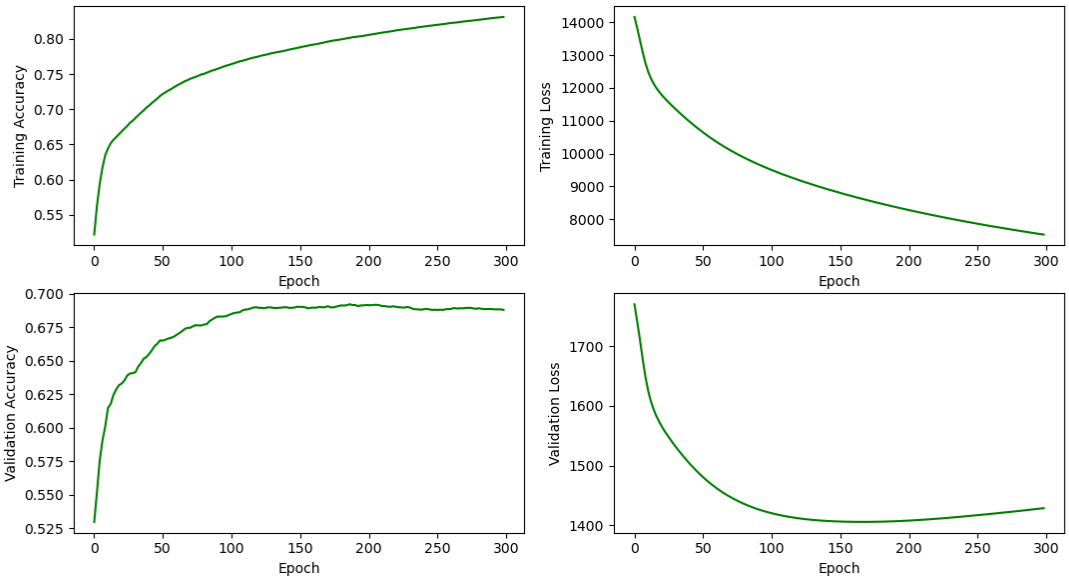
- ALS is used for matrix factorization for reconstruction - that is, factorize a matrix into two matrices U and Z, then combine them. On the other hand, the autoencoder first encodes the input by imposing a bottleneck that forces a compressed knowledge representation, then decodes it to get back the original data.
- The objective for ALS is $min_{U,Z} \frac{1}{2} \sum_{(n,m) \in O} (C_{nm} - u_n^T z_m)^2$, where C is the sparce matrix and $O = \{(n,m) \ : \ entry \ (n,m) \ of \ matrix \ C \ is \ observed\}$. The objective for autoencoder is $min_\theta \sum_{v \in S} |v - f(v;\theta)|_2^2$, where f is the reconstruction of the input v, $v \in R^{N_{questions}}$ from a set of users S.
- In ALS, we minimize the loss with respect to two factors, U and Z. However, we minimize with respect to only one factor, which is the weight matrix W, in autoencoder.
- We use gradient decent to update W at each iteration in autoencoder until convergence. In the case of ALS, the objective is non-convex in U and Z jointly, but is convex as a function of either U or Z individually. Therefore, we fix Z and optimize U, then fix U and optimize Z, so on until convergence.

b)

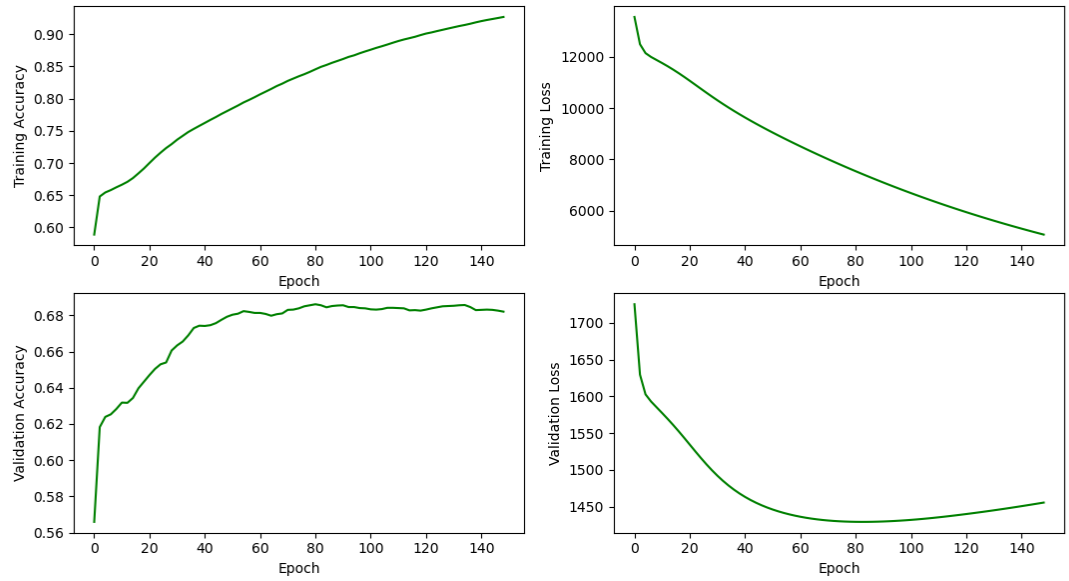
Code in neural_network.py.

c)

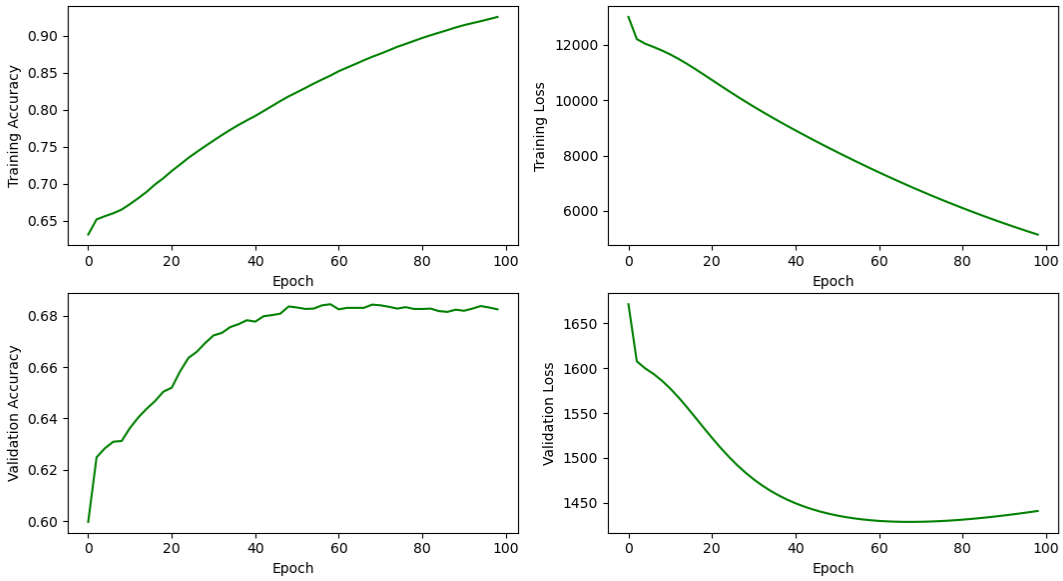
Plot for k = 10, learning rate = 0.005, num epoch = 300.



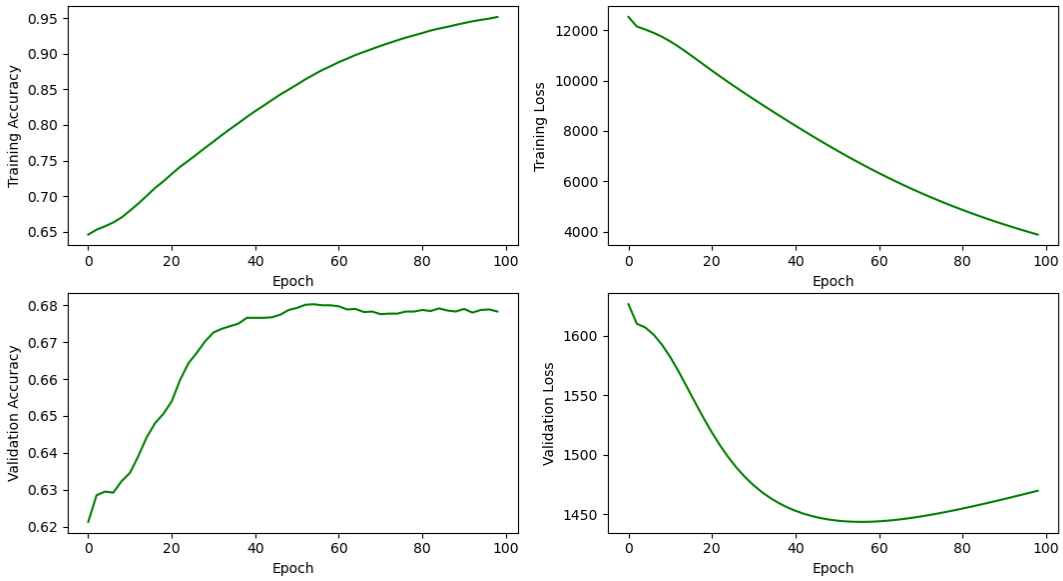
Plot for k = 50, learning rate = 0.005, num epoch = 150.



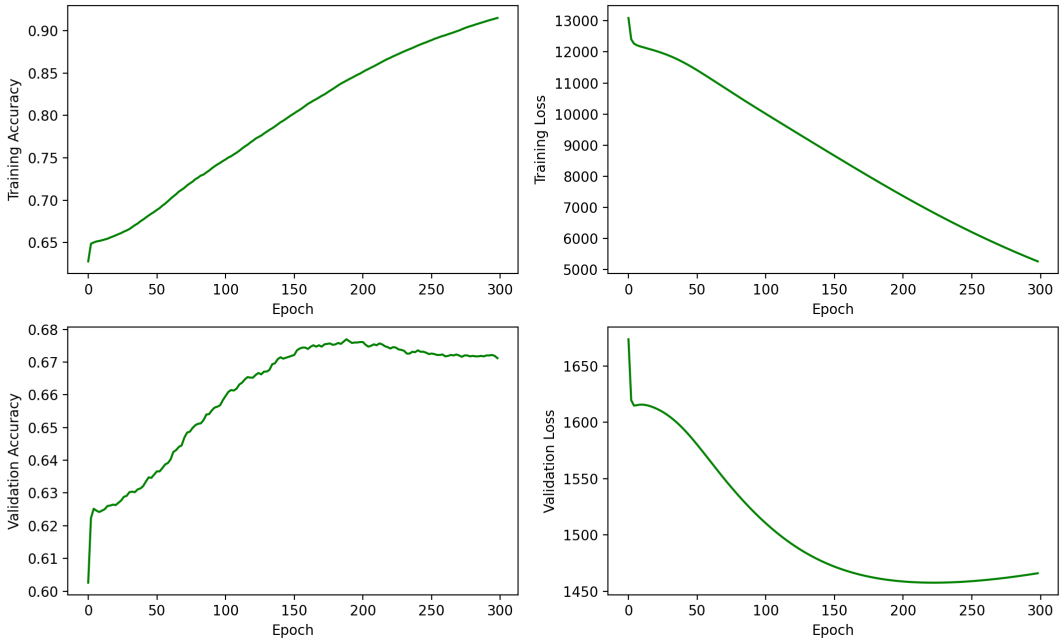
Plot for k = 100, learning rate = 0.005, num epoch = 100.



Plot for k = 200, learning rate = 0.005, num epoch = 100.



Plot for k = 500, learning rate = 0.001, num epoch = 300.

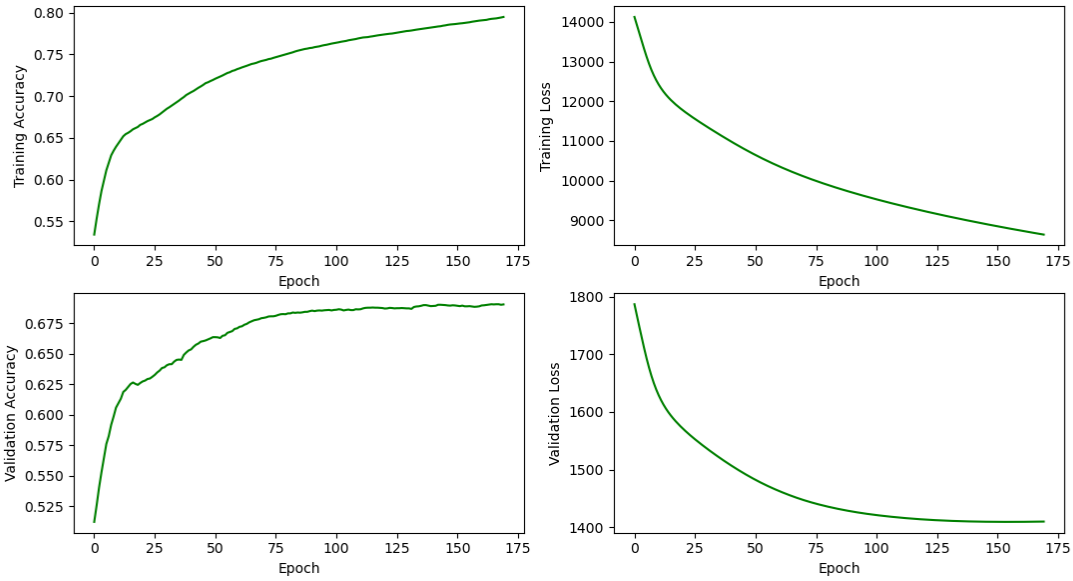


K	Highest Validation Accuracy	Learning Rate	Epoch
10	0.692210	0.005	300
50	0.686142	0.005	150
100	0.684448	0.005	100
200	0.680215	0.005	100
500	0.676969	0.001	300

The k that achieves the highest validation accracy (0.692210) is 10, so we choose it. Note the highest validaiton accuracy is calculated among the validaiton accuracy at all epoches.

d)

Plot for k = 10, learning rate = 0.005, num epoch = 170.

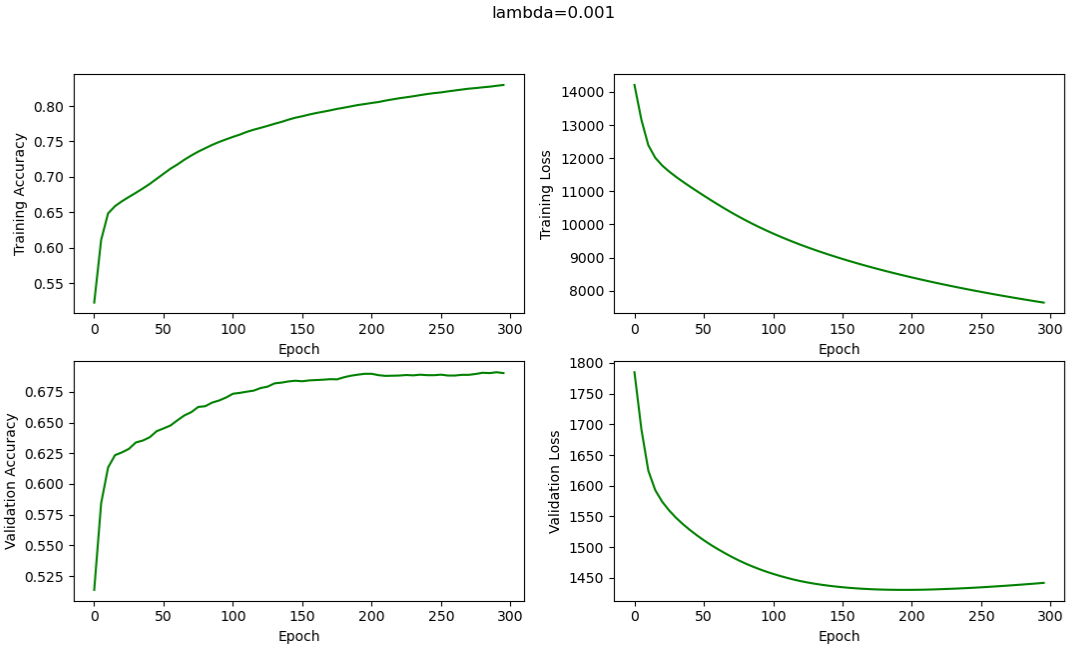


The final test accuracy is 0.691787.

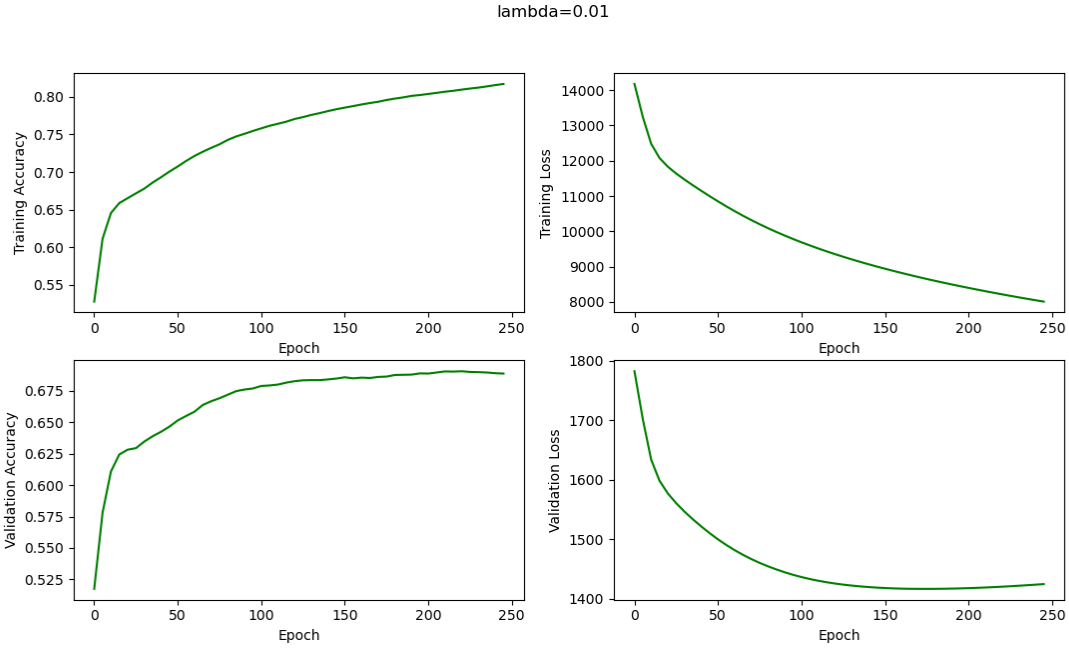
As epoch increases, the training and validation accuracies increase and the training and validation losses decrease, all with a diminishing rate. The training accuracy goes from 0.534117 to 0.794683; training loss from 14125.214311 to 8637.385720; validation accuracy from 0.512278 to 0.690375; validation loss from 1786.580161 to 1409.755266. The validation accuracy and loss eventually converge. The final training accuracy is higher than the validation and test accuracy; if more training is done, validation loss would increase, which indicates a subtle trace of overfitting.

e)

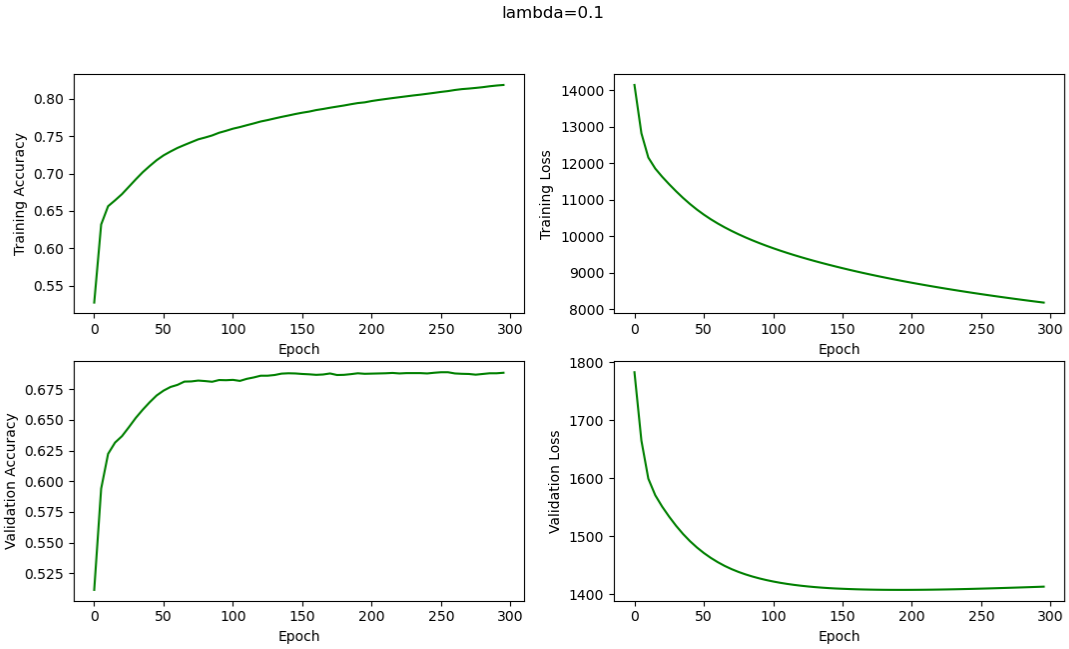
Plot for lambda = 0.001.



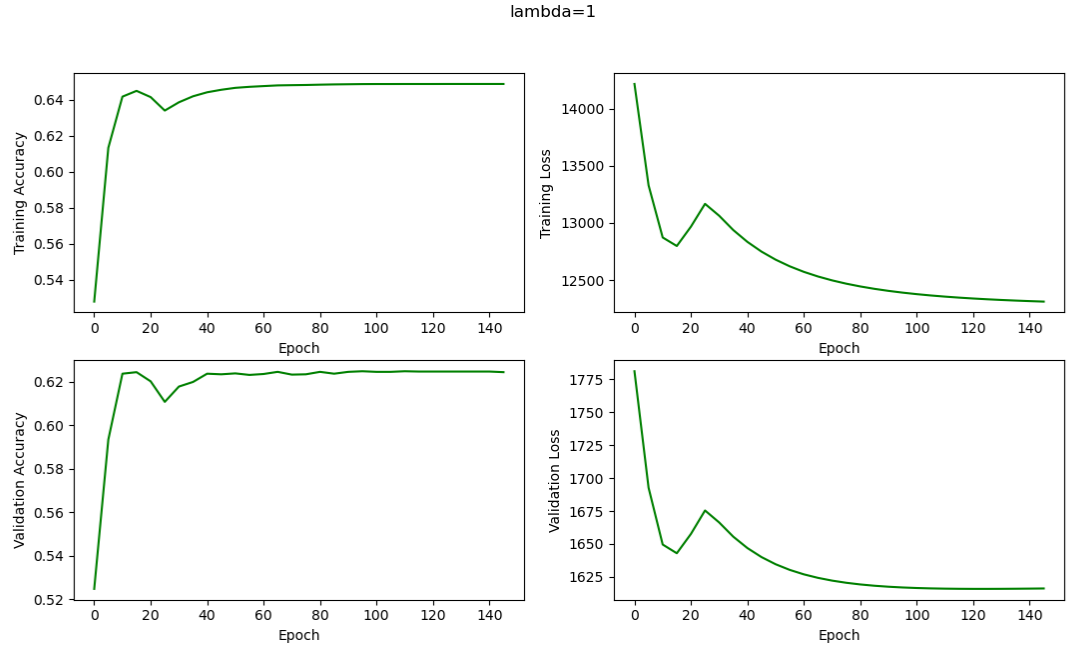
Plot for lambda = 0.01.



Plot for lambda = 0.1.



Plot for lambda = 1.0.



Lambda	Highest Validation Accuracy
0.0	0.692210
0.001	0.690940
0.01	0.690517
0.1	0.688823
1.0	0.624894

We can see model without regularization performs better, as it has a higher maximum validation accuracy. As lambda increases, the maximum validation accuracy decreases. The final validaiton accuracy is 0.690375 and test accuracy 0.691787.

In [ ]:

Question 4

```
Validation Accuracy is: 0.7012418854078465
 Test Accuracy is: 0.6974315551792266
```

Process:

For this ensemble process, we select IRT model developed in q2 as base model.

Steps:

1. Randomly select samples with replacement from train data (size is the same as the original data size).
2. Train the model with selected data.
3. Make predictions for validation and test data.
4. Repeat step 1-3 for 3 times and record the predictions for each round.
5. Average the predictions by formula

$$y_{bagged} = 1(\sum_{i=1}^{m} \frac{y_i}{m} > 0.5)$$

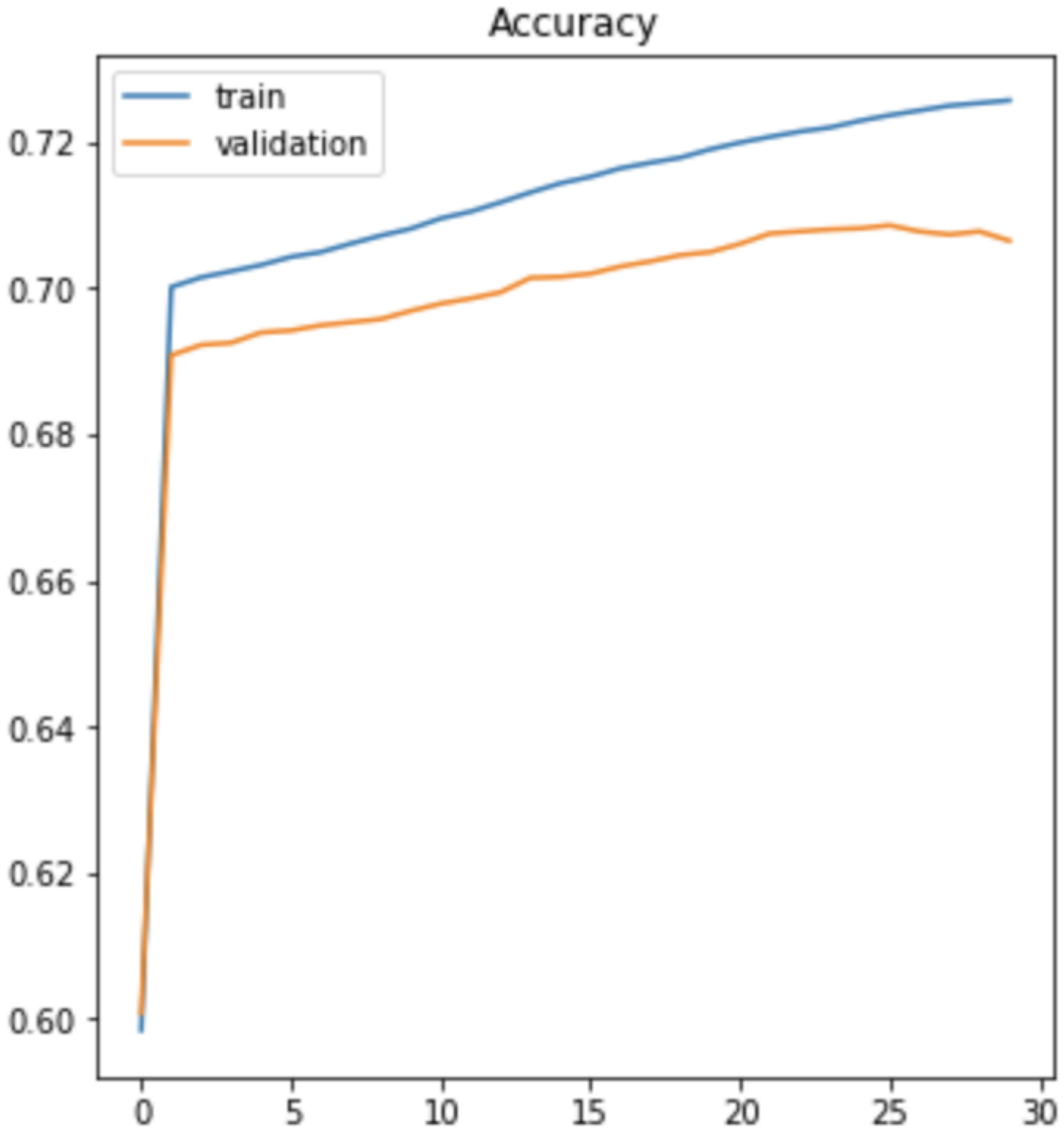
   This is the same as taking a majority vote.
6. Calculate the accuracy using the final average predicted correctness.

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

Ans: No, we didn't obtain better performance using the ensemble. The ensembled accuracy (around 0.69) is always close to non-ensembled (around 0.69) accuracy. This might be because there is adequate data to train all models, the validation/test data matches the training distribution. Under this condition, generalizing data is relatively useless.

Notice: The following graph indicates the training process of the not ensemble model, and we also can use the validation accuracy and test accuracy of non-ensemble model to compare with the ensemble model and we can see there is no big difference between the accuracies of these two models.

Out[4]:




```
Validation Accuracy is: 0.7061812023708721
Test Accuracy is: 0.6996895286480384
```

Algorithm Description

Previous Algorithm Analysis

We want to improve the performance of **Item Response Theory (IRT) Model** in **Question 2**.

As reported in Question 2, we know the metrics of the original IRT Model as following:

Train Accuracy: 0.739, Validation Accuracy: 0.705, Test Accuracy: 0.7047

Observing the training and validation curves during the training process, we can know that there is no huge difference between the accuracy of training data, validation data. This means the variance of the model is low and the overfitting of model is not severe. However, the accuracy of training dataset is only 0.739, which is not high and close to test accuracy. This indicates that the model is underfitting.

In this way, in order to avoid underfitting, **we better increase the complexity of the original model.** Following this opinion, we come up with one key opinion and two modifications ideas.

Algorithm Modification Ideas

Key Intuition: student's ability to solve different types of questions are different.

Some students might be good at algebra and statistics but not good at geometry. Some students might be interested in number but not keen on algebra.

Instead of using overall ability of the student and overall difficulty of the questions, we can use more detailed information to train the model and predict student's behavior. This improve the complexity of this model and it supposed to have a better performance.

There are **two main modification ideas:**

1. Train conditional Item Response Theory Model.

- Mathematical definition: Train θ_{ik} and $\beta_{j|k}$ for each question type k, using the $p(c_{ij}|\theta_{ik}, \beta_{j|k}) = sigmoid(\theta_{ik} - \beta_{j|k})$.
- Explanation: θ_{ik} represents i-th student's ability to solve k type of question, $\beta_{j|k}$ represents the difficulty of j-th question under we know it is k type of question.

2. Add a discriminative term to item response theory model.

- Mathematical Definition: Train θ_i, β_j and k_j using $p(c_{ij}|\theta_i, \beta_j) = c + (1-c) \times sigmoid(k_j(\theta_i - \beta_j))$
- Explanation: k_j is how steep the sigmoid looks (i.e. how discriminative this type of question is), c is the probability of getting question right via random guess.
- This idea comes from project tutorial slides, possible extension.

Model Training Process

We classify the questions into 5 categories: **Number, Algebra, Geometry, Statistics and Other**. This is because when we look into the subjects of questions, it is easy to observe that all questions are belonging to at least one of those five categories (1, 17, 39, 68 and others).

After training several models, we leave 2 models and combine them to get a final model.

1. Model 1: Discriminative IRT.

- This is based on modification idea 2, $p(c_{ij}|\theta_i, \beta_j) = c + (1-c) \times sigmoid(k_j(\theta_i - \beta_j))$

2. Model 2: Conditional Discriminative IRT.

- This is based on both modification idea 1 and 2, using $p(c_{ij}|\theta_{ik}, \beta_{j|k}) = c + (1-c) \times sigmoid(k_j(\theta_{ik} - \beta_{j|k}))$

3. Final Combinational Model: combination of Model 1 and Model 2.

- We use Model 1 in terms of predicting questions in category Number and Algebra, and use Model 2 in terms of predicting questions in other categories.

Result Comparison

Overall, the test accuracy of the **original model** is 0.7047 and the test accuracy of the **final combinational model** is 0.7144.

To be see the detailed improvement of our accuracy, we will first show metrics for each category of **Original Model**, **Conditional Discriminative IRT** (Model 2) and **Discriminative IRT** (Model 1) as following:

	Original Validation Accuracy	Original Test Accuracy	Model 2 Validation Accuracy	Model 2 Test Accuracy	Model 1 Validation Accuracy	Model 1 Test Accuracy	Number of data
Number	0.7202	0.7007	***0.72062***	***0.7182***	0.7186	0.7044	25073
Algebra	0.6887	0.7079	***0.70***	***0.71842***	0.7024	0.7118	12187
Statistics	0.7186	0.6950	0.7003	0.6918	***0.7262***	***0.7012***	14388
Geometry	0.6952	0.71786	0.7040	0.6962	***0.6947***	***0.7144***	5177
Other	0.65517	0.6428	0.6379	0.4643	***0.6551***	***0.6785***	508

Compared the result above, it is obvious that our **Conditional Discriminative IRT** can acheive a much better prediction result in **Number and Algebra** categories than the original model. We believe it is because we extract some feature of discriminative term k_j for these two categories.

However, it is not very useful in predicting questions in **Statistics, Geometry and Other** category. This might be because the number of data for each question category influences the quality of the discriminative parameter k_j. Since the performance doesn't become better, we can know that for these 3 categories, we don't have sufficient data to extract the qualitative feature for k_j.

In that case, it is better to make predictions using less complex **Discriminative Model** (Model 1) to avoid overfitting for the other three category. This is why we choose to combine Model 1 and Model 2 and obtain **Final Combinational Model**.

In conclusion:

1. We predict the questions in category Number and Algebra by Model 2
2. We predict the questions in category Geometry, Statistics and Other by Model 1.
3. The Final Combinational Model can achieve a better performance (test accuracy 0.7144) than the original model (test accuracy 0.7047).

Limitations

For both model 1 and 2, since we split the data into different categories, the size of data in each category is actually smaller than the original category. Due to this reason, our model might suffer from overfitting and results in a bad performace, and this phenomenon is extremely obvious in some categories. Therefore, it is better not to use the modified algorithm without having sufficient data.

One method to solve the lack of data is **using all data to train the model and predict the value using this model.** For those categories with less data, we can use those parameters trained by all data to predict the result since the overall ability of students and difficulties of questions also can reflect the result of the performance of that students on some questions to some extent. **This method has already been implemented in this question.**

Another method to solve the lack of data is we can use **bagging** to improve the accuracy of the model. As we all know, bagging technique can be helpful to reduce the variance of the model.

In addition, to extend the model further, we can also use metadata of students to improve the performance of the model. For example, we know the age of the students in the students metadata, based on common sense, if the student is too young, the student is not able to solve some hard questions especailly for some difficult math questions. In this way, we can find more detailed classes based on both age of students and question category and probably construct a better model.

Appendix

Algorithm Box

Result: Split the datasets
Input : Original Dataset
Output: Sub-datasets for each category

for *datapoint* $\in$ *OriginalDataset* **do**
  Initialize NumberData, GeometryData, AlgebraData, StatisticsData, OtherData
  if *datapoint can be classified as number, geo, alg, sta question* **then**
    put the datapoint into corresponding dataset
  else
    put the datapoint into OtherData
  end
end
return *NewDatasets*

Algorithm 1: Split Dataset

Result: Calculate Accuracy for Dataset
Input : Dataset, C, θ, β,
Output: Accuracy of the Dataset

total $= 0$, correctCount $= 0$
for *datapoint* $\in$ *OriginalDataset* **do**
  Find the category of the datapoint
  Calculate the probability of the datapoint with corresponding C, θ, β, k *by* $p(c_{ij}|\theta_i, \beta_j) = c + (1-c) \times sigmoid(k_j(\theta_i - \beta_j))$
  $predictedResult = \lfloor probability + 0.5 \rfloor$
  Compare predictedResult and true result of the datapoint
  update total and correctCount
end
return $\frac{correctCount}{total}$

Algorithm 2: Calculate Accuracy

Result: Train Model
Input : Dataset, learningRate, iterations, C
Output: θ, β, k

Initialize θ, β, k
while $iterations > 0$ **do**
  Calculate θ, β, k by the formula on the last page
  $\theta = \theta - lr \times d\theta$
  $\beta = \beta - lr \times d\beta$
  $k = k - lr \times dk$
  $iterations = iterations - 1$
end
return θ, β, k

Algorithm 3: Train Model

θ, β, k Updates

We know $p(C_{ij}=1 \mid \theta_i, \beta_j) = c + (1-c)\,\text{sigmoid}(k_j(\theta_i - \beta_j))$

Then $p(D \mid \theta, \beta) = \prod_{i=1}^{N} \prod_{j=1}^{M} \left[p(C_{ij}=1 \mid \theta_i, \beta_j)\right]^{C_{ij}} \left[1 - p(C_{ij}=1 \mid \theta_i, \beta_j)\right]^{(1-C_{ij})}$

$$l(\theta, \beta) = \log p(D \mid \theta, \beta) = \sum_{i=1}^{N} \sum_{j=1}^{M} C_{ij} \cdot \log\left[p(C_{ij}=1 \mid \theta_i, \beta_j)\right] + (1-C_{ij}) \log\left[1 - p(C_{ij}=1 \mid \theta_i, \beta_j)\right]$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{M} C_{ij} \log\left[c + (1-c)\,\text{sigmoid}(k_j(\theta_i - \beta_j))\right] + (1-C_{ij}) \log\left[1 - c - (1-c)\,\text{sigmoid}(k_j(\theta_i - \beta_j))\right]$$

In the following steps, I will use $S = \text{sigmoid}(k_j(\theta_i - \beta_j))$ for convenience.

$$\frac{\partial l}{\partial \theta_i} = \sum_{j=1}^{M} C_{ij} \cdot \frac{(1-c)S(1-S)\cdot k_j}{c + (1-c)S} - (1-C_{ij}) \frac{(1-c)S(1-S)k_j}{1-c-(1-c)S}$$

$$\frac{\partial l}{\partial \beta_j} = \sum_{i=1}^{N} C_{ij} \frac{(1-c)S(1-S)\cdot(-k_j)}{c + (1-c)\cdot S} + (1-C_{ij}) \cdot \frac{(1-c)S(1-S)k_j}{1-c-(1-c)S}$$

$$\frac{\partial l}{\partial k_j} = \sum_{i=1}^{N} C_{ij} \frac{(1-c)S\cdot(1-S)(\theta_i - \beta_j)}{c + (1-c)S} - (1-C_{ij}) \frac{(1-c)S(1-S)\cdot(\theta_i - \beta_j)}{1-c-(1-c)S}$$

Then consider negative log-likelihood

The derivatives are $-\frac{\partial l}{\partial \theta_i}$, $-\frac{\partial l}{\partial \beta_j}$, $-\frac{\partial l}{\partial k_j}$

Contribution:

Part A

Every student completed the question individually.

- Q1: Yun-Hsiang Chan
- Q2: Hanzi Jiang
- Q3: Sixuan Wu
- Q4: Yun-Hsiang Chan

Part B

- Discussion: Yun-Hsiang Chan, Hanzi Jiang, Sixuan Wu
- Algorithm Implementation: Sixuan Wu, Hanzi Jiang
- Report Writing: Hanzi Jiang, Yun-Hsiang Chan, Sixuan Wu