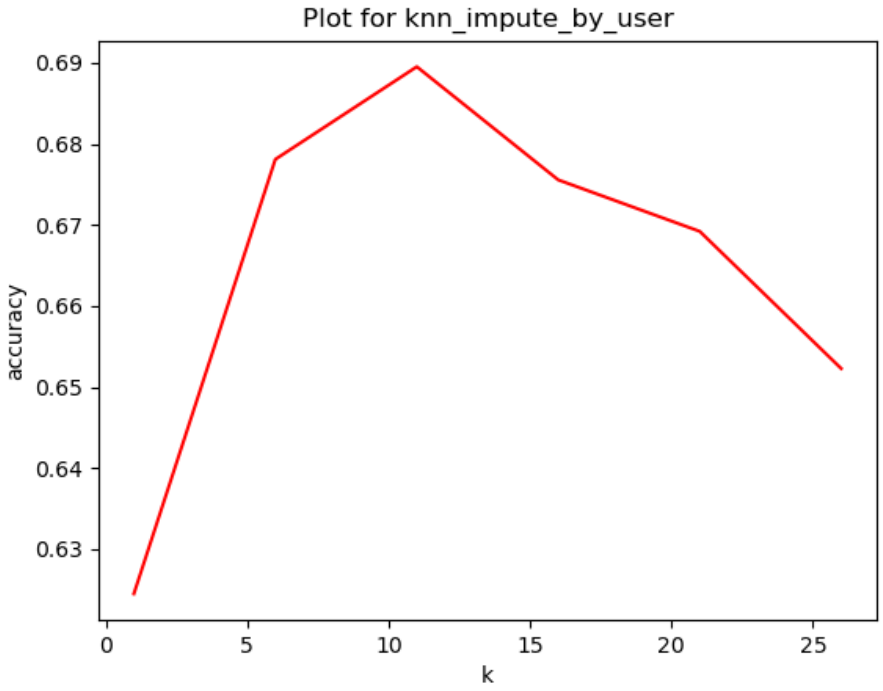
Part A

1. **k-Nearest Neighbor.**

   (a) The plot of accuracy for the validation data:



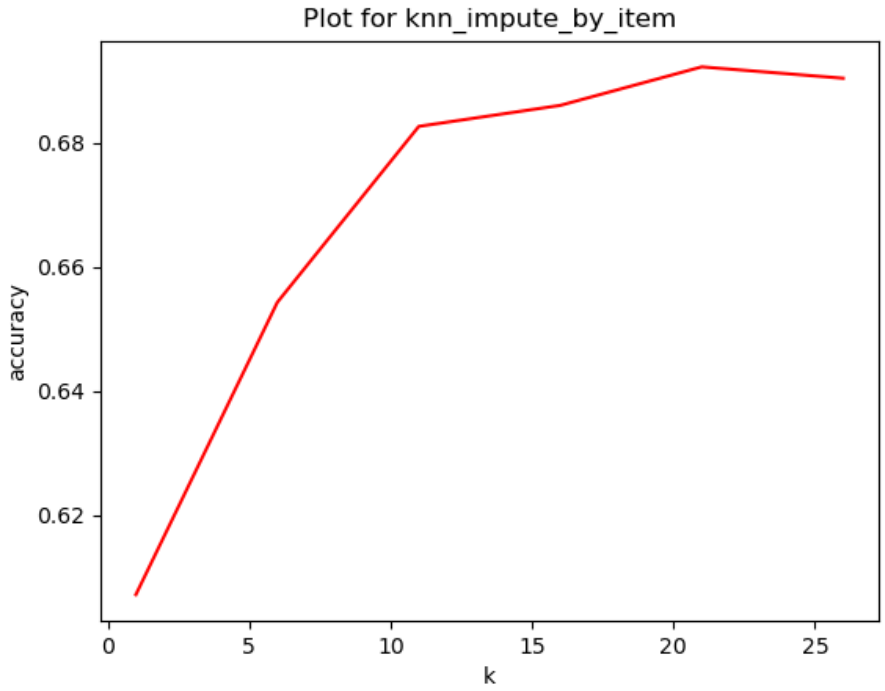
   The report of accuracy:

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

   (b) The highest performance occurs when k is 11. The accuracy is 0.6895286480383855

```
The best k: 11
The best accuracy: 0.6895286480383855
```

(c) The plot for knn_impute_by_item:



The report of accuracy:

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

The best performance occurred at $k = 21$ with the accuracy of 0.6922099915325995.

```
The best k: 21
The best accuracy: 0.6922099915325995
```

(d) The comparison for the test accuracy:

```
User-based. The accuracy on test data: 0.6841659610499576
Item-based. The accuracy on test data: 0.6816257408975445
```

Looking at the prediction on the test data, user-based filtering has higher accuracy than the item-based collaborative filtering. This is a contrasting result to the result of accuracy from the validation data from the above. Therefore, there is no superior method out of user- and item- based collaborative filtering.

(e) One potential limitation of Knn for this problem is that Knn is not efficient for a large data set. For example. if a point the algorithm is to predict may predict for one value for a certain k and predict for other value for other values of k. This happens when there is a large sample size.

Another potential problem is the curse of large dimensions. The dimensions for the data is huge - 1774×542. All the points will seem to be at the same distance and could be misleading to compute the k nearest neighbourhoods.

2. **Item Response Theory**

a) We derive the log-likelihood; log p($\mathbf{C}$| θ, β) for all students and questions.
Let M=542 be the total number of students. Then the i'th student is such that $0 \leq$ i $\leq$ M-1
Let N=1774 be the total number of questions. Then the j'th question is such that $0 \leq$ j $\leq$ N-1.

The probability question j is correctly answered by student i is p(c_{ij}=1 | θ_i, β_j) = $\sigma(\theta_i$ - $\beta_j)$. Then p(c_{ij}=0 | θ_i, β_j) = 1 - $\sigma(\theta_i$ - $\beta_j)$, is the probability question j is not correctly answered by student i. Therefore the likelihood p($\mathbf{C}$| θ, β) = $\prod_{i=0}^{M-1} \prod_{j=0}^{N-1} \sigma(\theta_i - \beta_j)^{c_{ij}} (1 - \sigma(\theta_i - \beta_j))^{1-c_{ij}}$.

$\ell(\theta, \beta)$ = log p($\mathbf{C}$| θ, β) = $\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} [c_{ij}$ log$(\sigma(\theta_i - \beta_j)) + (1 - c_{ij})$ log(1 - $\sigma(\theta_i - \beta_j))]$.

Derivative of the log-likelihood with respect to θ_i;

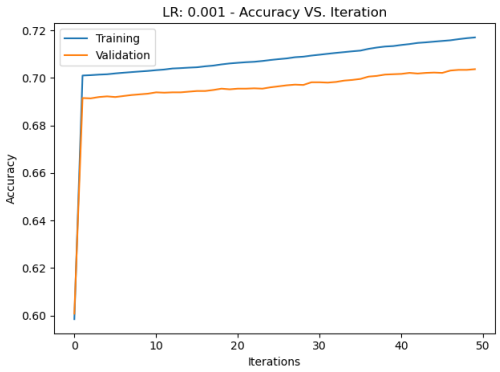
$$\frac{\partial \ell}{\partial \theta_i} = \sum_{j=0}^{N-1} \left[\frac{c_{ij}\sigma(\theta_i-\beta_j)(1-\sigma(\theta_i-\beta_j))}{\sigma(\theta_i-\beta_j)} - \frac{(1-c_{ij})\sigma(\theta_i-\beta_j)(1-\sigma(\theta_i-\beta_j))}{(1-\sigma(\theta_i-\beta_j))}\right] = \sum_{j=0}^{N-1} [c_{ij} - \sigma(\theta_i - \beta_j)]$$

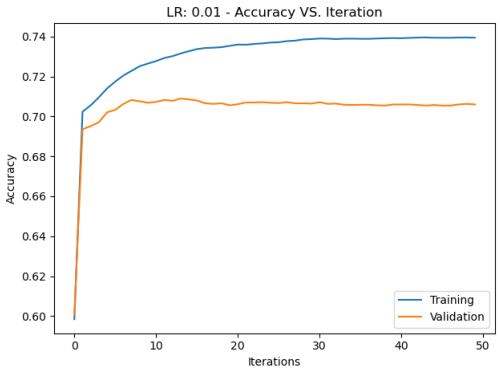
Derivative of the log-likelihood with respect to β_j;

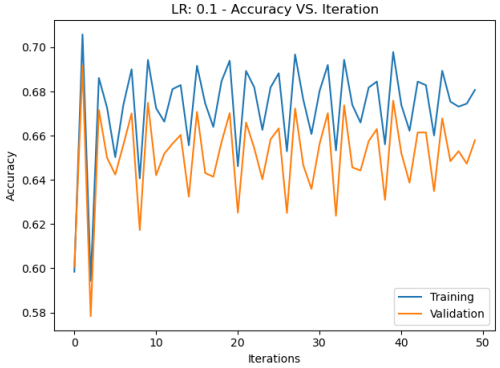
$$\frac{\partial \ell}{\partial \beta_j} = \sum_{i=0}^{M-1} \left[\frac{-c_{ij}\sigma(\theta_i-\beta_j)(1-\sigma(\theta_i-\beta_j))}{\sigma(\theta_i-\beta_j)} + \frac{(1-c_{ij})\sigma(\theta_i-\beta_j)(1-\sigma(\theta_i-\beta_j))}{(1-\sigma(\theta_i-\beta_j))}\right] = \sum_{i=0}^{M-1} [\sigma(\theta_i - \beta_j) - c_{ij}].$$

b)

- Initialize θ as $\mathbf{0}$ vector of dimensions (M, 1).
- Initialize β as $\mathbf{0}$ vector of dimensions (N, 1).
- Learning rate: 0.01
- Iterations: 50

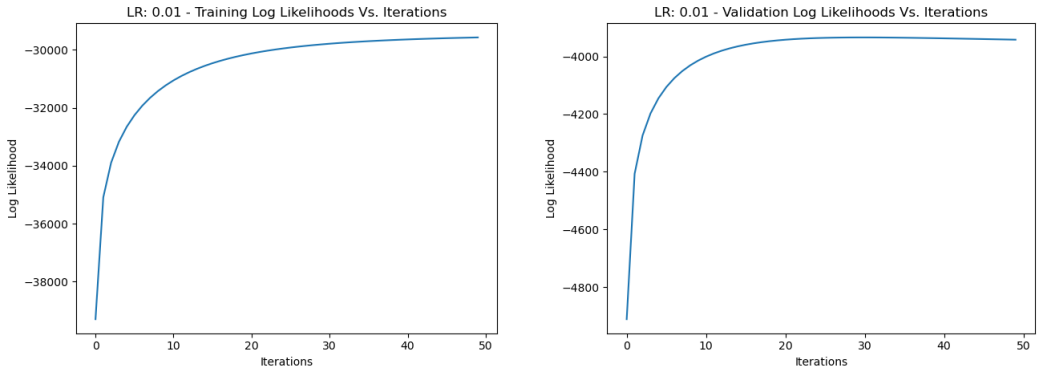






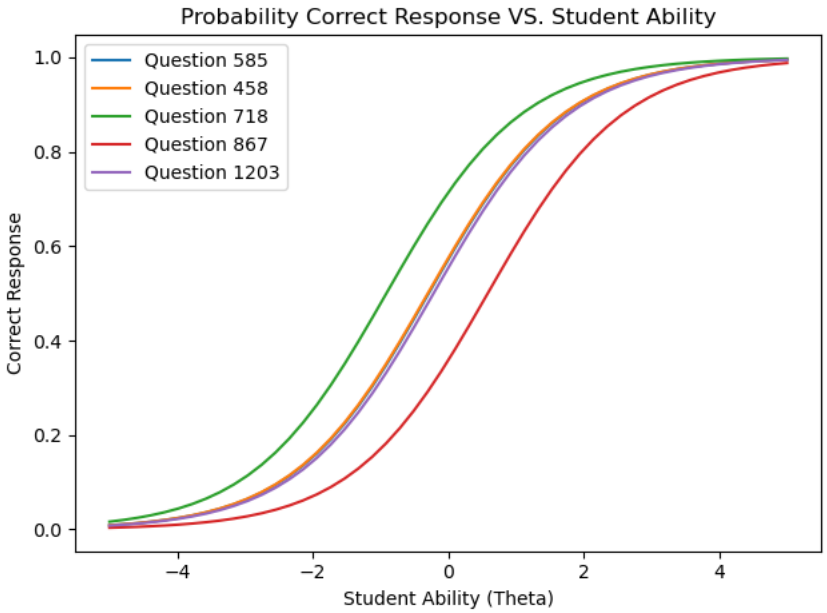
We use the above Accuracy VS. Iteration graphs to determine an appropriate learning rate and number of iterations as LR = 0.01 and Iterations as 50.

With our selected hyperparameters, we report the training curve that shows the training and validation log-likelihoods as a function of iteration.



c)
Please find code in item_response.py
Final Validation Accuracy: 0.7060400790290714
Final Test Accuracy: 0.7070279424216765

d)
Please find code in item_response.py
Using the trained θ and β, we plot the probability of the correct response as a function of θ for 5 questions $\{j_1, j_2, j_3, j_4, j_5\}$.



The 5 curves have a sigmoidal shape. The correct response or the probability that a student answers the question correctly increases as the ability of the student increases, following a sigmoidal shape. If we consider in the range of θ [-5, 5] then we note that a student has a higher correct response for question 718 (green curve) than question 867 (red curve), therefore we can say that question 867 is harder than 718 and that students with higher ability have a better probability of answering the more difficult questions. In general, we note that question 867 is the hardest of the 5 questions and question 718 is the easiest question.

3. **Matrix Factorization - Done by Asad Aleem**

We get the following results for Q3 part (a)

```
(svd with k=1) sparse_matrix_evaluate function gives accuracy   : 0.6782211402766017
on the validation set with k = 1 we get accuracy:  0.6428168219023427
on the test set with k = 1 we get accuracy:  0.6477561388653683


(svd with k=3) sparse_matrix_evaluate function gives accuracy   : 0.7051404177250917
on the validation set with k = 3 we get accuracy:  0.6583403895004234
on the test set with k = 3 we get accuracy:  0.6627152130962461


(svd with k=5) sparse_matrix_evaluate function gives accuracy   : 0.7170829805249789
on the validation set with k = 5 we get accuracy:  0.659046006209427
on the test set with k = 5 we get accuracy:  0.6635619531470506


(svd with k=8) sparse_matrix_evaluate function gives accuracy   : 0.7319714930849562
on the validation set with k = 8 we get accuracy:  0.6603161162856337
on the test set with k = 8 we get accuracy:  0.65961049957663


(svd with k=12) sparse_matrix_evaluate function gives accuracy  : 0.7519757267852103
on the validation set with k = 12 we get accuracy:  0.6560824160316117
on the test set with k = 12 we get accuracy:  0.6576347727914197


(svd with k=30) sparse_matrix_evaluate function gives accuracy : 0.8180920124188541
on the validation set with k = 30 we get accuracy:  0.6548123059554051
on the test set with k = 30 we get accuracy:  0.645780412080158


(svd with k=50) sparse_matrix_evaluate function gives accuracy : 0.8668854078464578
on the validation set with k = 50 we get accuracy:  0.648461755574372
on the test set with k = 50 we get accuracy:  0.6454981653965566


(svd with k=80) sparse_matrix_evaluate function gives accuracy : 0.9105983629692351
on the validation set with k = 80 we get accuracy:  0.64422805532035
on the test set with k = 80 we get accuracy:  0.6429579452441434


(svd with k=100) sparse_matrix_evaluate function gives accuracy : 0.9280623765170759
on the validation set with k = 100 we get accuracy:  0.6470505221563647
on the test set with k = 100 we get accuracy:  0.6370307648885125


(svd with k=150) sparse_matrix_evaluate function gives accuracy : 0.9588272650296359
on the validation set with k = 150 we get accuracy:  0.6429579452441434
on the test set with k = 150 we get accuracy:  0.6373130115721141


(svd with k=200) sparse_matrix_evaluate function gives accuracy : 0.9776319503245837
on the validation set with k = 200 we get accuracy:  0.6326559412926898
on the test set with k = 200 we get accuracy:  0.632514817950889


(svd with k=500) sparse_matrix_evaluate function gives accuracy : 1.0
on the validation set with k = 500 we get accuracy:  0.6243296641264465
on the test set with k = 500 we get accuracy:  0.6260231442280553
```

The smallest value we tried was $k = 1$, while the largest value we tried was $k = 500$. We see that for $k > 50$, the results do not improve but infact gradually degrade very slightly as k keeps increasing. However the variation in the accuracy is quite low, since for $k = 1$ we get accuracy $= 0.64$ on the test set, and for $k = 500$ we get accuracy $= 0.62$ for the test set.

Our best k is k = 5 which gives us accuracy = 0.66

(b) One limitation of SVD in this instance is that we are dealing with a sparse matrix. SVD would perform better if instead of a sparse matrix, we had a matrix with all values filled in it. In the case of a sparse matrix, we implement a generalization of the SVD(or PCA) which is matrix completion, and we do that in the following parts by implementing the ALS algorithm. In the previous part we were able to make our SVD algorithm work by filling in the missing values with the average, but this obviously conceals the patterns that the data in real life would have.

(c) See code in the matrix_factorization.py file.

(d) The output for part (d) is on the next page :

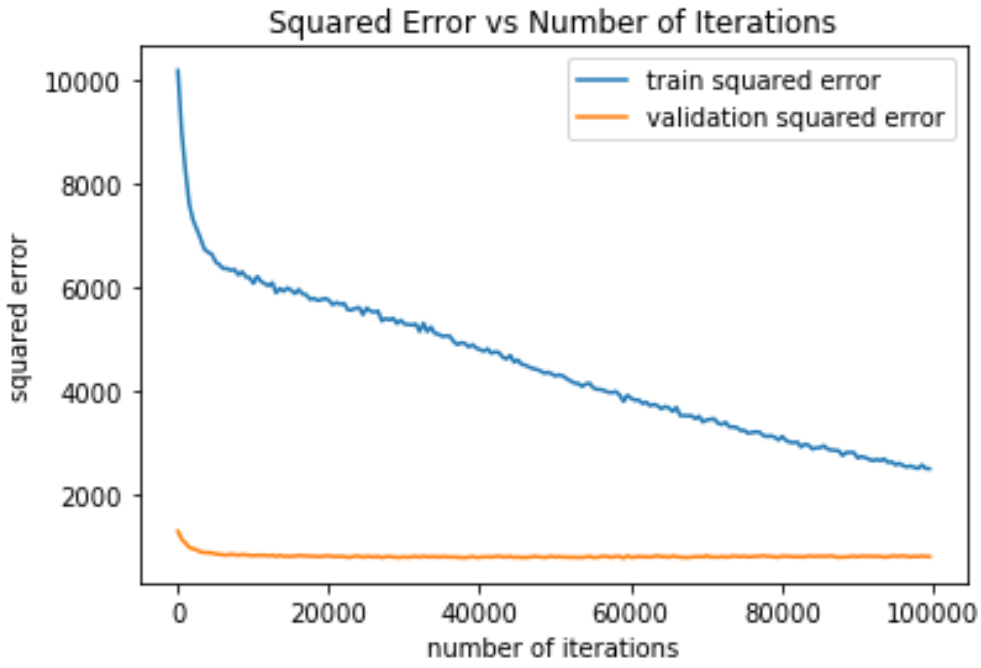
```
squared_error_loss for k = 5  is:  6509.465653787498


our reconstructed matrix is:
[[ 0.18467015  0.35366546  0.03175936 ...  0.05929797  0.1137261
  -0.05271263]
 [ 0.28470959  0.04922738  0.78221061 ...  0.40090636  0.53664743
   0.59437002]
 [ 0.50430969 -0.00698417  0.7352825  ...  0.40448701  0.53125299
   0.6308036 ]
 ...
 [ 0.35855411  0.2265379   0.50815782 ...  0.23847711  0.4064857
   0.30805877]
 [ 0.80402418  0.76163243  0.81162205 ...  0.77989738  0.7995251
   0.79116035]
 [ 0.61525489  0.47573949  1.19459949 ...  0.69635726  0.93182127
   0.88751728]]


sparse_matrix_evaluate function gives accuracy:  0.6884349421394299
on the validation set with k = 5 we get accuracy:  0.6361840248377082
on the test set with k = 5 we get accuracy:  0.6265876375952583


squared_error_loss for k = 100  is:  2647.037506760465


our reconstructed matrix is:
[[0.61599771 0.8608969  0.59436657 ... 0.55876042 0.63292567 0.29461681]
 [0.506313   0.46364486 0.5920723  ... 0.44063349 0.5360074  0.55856253]
 [0.21916359 0.19422208 0.75312658 ... 0.3207964  0.37036911 0.47155265]
 ...
 [0.37688982 0.45035945 0.46724889 ... 0.41921101 0.47406868 0.41732632]
 [0.73009084 0.76320277 0.85955254 ... 0.68263402 0.89442082 0.82396774]
 [0.35676482 0.4632271  0.57856827 ... 0.38559882 0.49163932 0.33344103]]


sparse_matrix_evaluate function gives accuracy:  0.89232289020604


on the validation set with k = 100 we get accuracy:  0.6581992661586227
on the test set with k = 100 we get accuracy:  0.6550945526390065


squared_error_loss for k = 150  is:  2559.3379778950207


our reconstructed matrix is:
[[0.09773514 0.47829281 0.52976507 ... 0.44651038 0.46999902 0.6295515 ]
 [0.69352517 0.95018724 0.61368322 ... 0.50214952 0.7518459  0.69890024]
 [0.50681924 0.90472154 0.42857686 ... 0.29785295 0.66470085 0.70971734]
 ...
 [0.05405222 0.27939499 0.11536077 ... 0.09731177 0.24149785 0.14640777]
 [0.60905864 0.95495911 0.77350866 ... 0.6521639  0.86948844 0.88563841]
 [0.15630313 0.2740406  0.25122653 ... 0.17780193 0.28644847 0.26223657]]


sparse_matrix_evaluate function gives accuracy:  0.8978619813717189


on the validation set with k = 150 we get accuracy:  0.6579170194750211
on the test set with k = 150 we get accuracy:  0.6737228337567034
```

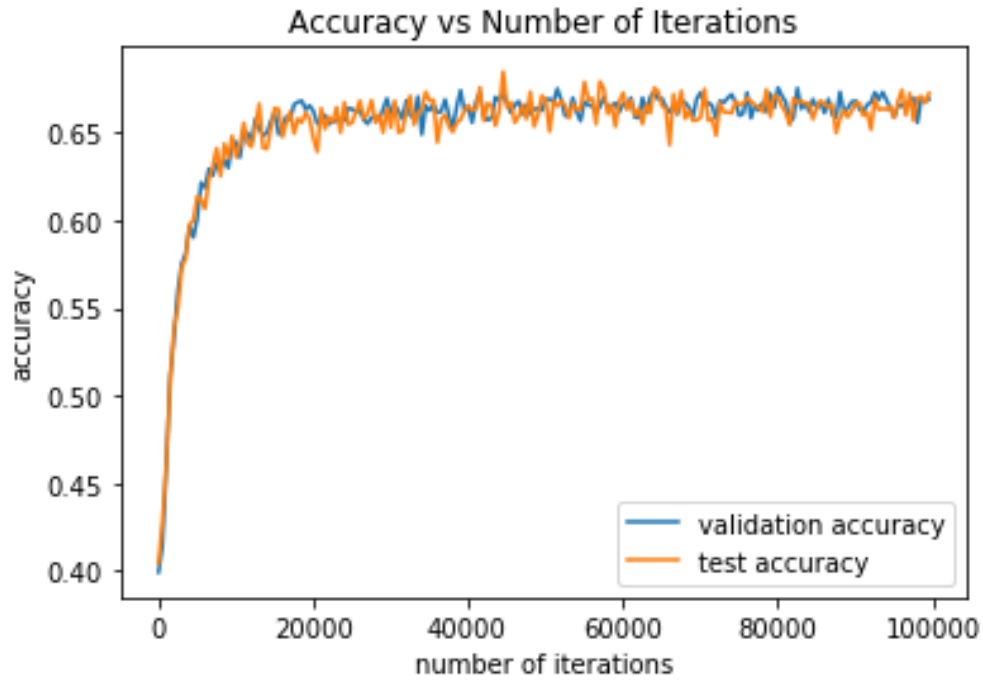
```
squared_error_loss for k = 200  is:  2438.2382775590318


our reconstructed matrix is:
[[-0.06845426  0.54777511  0.14531815 ...  0.37566271  0.44212565
   0.61492859]
 [ 0.35553194  0.59577041  0.63777721 ...  0.28601129  0.6566541
   0.40131551]
 [ 0.12018514  0.37509942  0.45166856 ...  0.13497843  0.35825462
   0.21549145]
 ...
 [ 0.3126661   0.3591988   0.2612028  ...  0.18998328  0.39041753
   0.31295822]
 [ 0.60289736  1.0004326   0.72118128 ...  0.49606089  1.015449
   0.77247352]
 [ 0.40568792  0.60470966  0.41019531 ...  0.28357823  0.57053981
   0.49910768]]


sparse_matrix_evaluate function gives accuracy:  0.9029953429297206


on the validation set with k = 200 we get accuracy:  0.6665255433248659
on the test set with k = 200 we get accuracy:  0.6740050804403048


squared_error_loss for k = 500  is:  2278.741194636234


our reconstructed matrix is:
[[0.66814645 0.42535543 0.69860169 ... 0.70507125 0.68161397 0.56601293]
 [0.56741352 0.63963201 0.79336839 ... 0.58751148 0.75151615 0.68295445]
 [0.18805066 0.24234781 0.85009167 ... 0.34547387 0.40190611 0.27346493]
 ...
 [0.15749882 0.15832207 0.20155588 ... 0.1599242  0.16847558 0.18362757]
 [0.67485463 0.72657542 0.98863632 ... 0.74129724 0.87920609 0.87902877]
 [0.36662488 0.34771475 0.43102758 ... 0.34296571 0.41951576 0.38671666]]


sparse_matrix_evaluate function gives accuracy:  0.9114098221845893


on the validation set with k = 500 we get accuracy:  0.6699125035280835
on the test set with k = 500 we get accuracy:  0.68077900084674
```

As we can see, for some reason we are getting optimal results for learning rate = 0.3, which in my opinion is too high. The more iterations we use, the more accurate our results are, hence we use num_iterations = 100000. When we increase the iterations by another factor of 10, it takes several minutes to produce the result hence we settled on num_iterations = 100000.

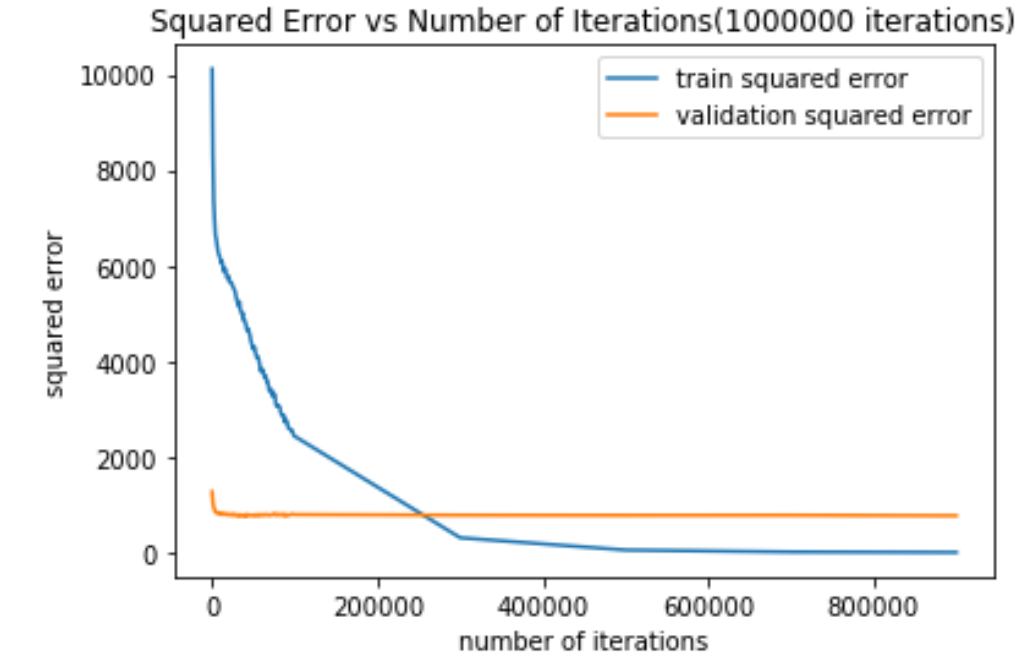
The best k is $k = 200$ since we get the highest accuracy i.e. validation accuracy = 0.667 and test accuracy = 0.674. NOTE: The squared_error_loss in the picture above is the squared error for the train_data. Also, an important point I would like to make, we get marginally better results (within one percent difference) for k = 500, and that too not always. However we choose 200 as our best k, since 500 is almost as close to 542 which is the total number of students, and ideally we would like our k to be significantly smaller than both; the number of students, and the number of questions.

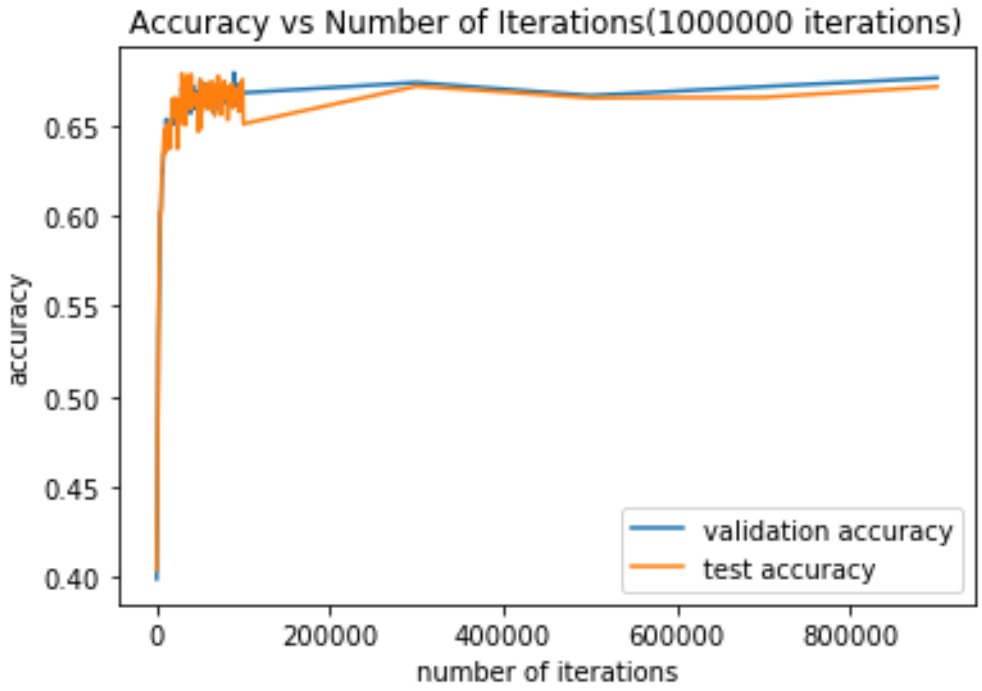
(e) For part (e) see the below two plots





Our final validation accuracy is: 0.67231, and our final test accuracy is: 0.67569. Also below is how the graph would look like if we increased the iterations by a factor of 10. (In the below two graphs the final validation accuracy is: 0.67626 and the final test accuracy is: 0.67146, which shows that the accuracy does not change despite the increase in number of iterations).





(f) Our current loss function contributes to inaccuracy when we make correct predictions with high confidence. Since we do not need to make predictions outside of the interval [0,1], we can compose our current loss function with the sigmoid function. This smooths out the loss function. So we can probably apply the sigmoid function on the product of the transpose of matrix u with z. So basically we need to minimize: $\min_{U,Z} \frac{1}{2} \sum_{(i,j)\in 0} (R_{ij} - \sigma(u_i^T z_j))^2$ where σ is our sigmoid function.

4. **Enesemble**
We bagged the result from Item Responsed Theory. To make a data where each elements represent (student, question) pair with an answer, we generated a list of lists where each list contains three elements of student_id, question_id, and the answer. Each triplet was obtained from the train_data dictionary. The train data contained 56688 (student, question) pair. Thus, we generated the list containing 56688 lists as its elements. We then followed the procedure shown in the lecture to generate 3 base models.

We then used Item Response Theory to optimized the parameters β and θ for each of the 3 base models. Subsequently, using the optimized parameters from each models, we predicted the probability of each students correctly answering the given question.

Here is the result for the ensemble using Item Response Theory The result without ensemble:

```
Final Validation Accuracy: 0.7060400790290714
Final Test Accuracy: 0.7070279424216765
```

The result with ensemble:

```
Bagged Validation Accuracy: 0.7029353655094552
Bagged Test Accuracy: 0.7039232289020604
```

The reason for the decrease in accuracy from using the bagged data may be that the bias of our data contributes to the error far greater than the imprecise nature of it. The bias of the data stays equivalent as a result of the bagging. The procedure improves the result by lowering the variance. Because our classification is binary (either 0 or 1), the variance will not exceed 1. Therefore, bagging of this data will cause information loss due to partial selection of the original data while lowering the variance by some insignificant magnitude.

Part B

Formal Description:
We extend the one-parameter IRT model provided in part A question 2 by using a two-parameter IRT model to predict students' correctness to diagnostic questions. In part A question 2, we used β_j to represent the difficulty of the j'th question. In the two-parameter extension we add second parameter α_j which represents the discrimination index of the j'th question. Using the two-parameter IRT model the probability that the question j is correctly answered by student i is formulated as:

$$p(c_{ij} = 1|\theta_i, \beta_j) = \sigma(\alpha(\theta_i - \beta_j))$$

A discrimination index is a value ranging from [-1, 1] and is calculated for each question j. The discrimination index for a question j (α_j) represents the differential capability of the j'th question, that is, the higher the α_j the better the question differentiates between students with high ability and low ability [1].

The discrimination index for each question j is calculated by first determining each student's test score, that is the number of questions they answered correctly out of all the questions attempted. Using these test scores, the students are split into to equally sized groups h and l representing the higher-scoring group and lower-scoring group respectively. The group l is subtracted from h and then divided by the number of students in a group t [2]. The discrimination index for each question is calculated as follows;

$$\alpha = \frac{h - l}{t}$$

Similar to part A question 2, we compute the likelihood for the two-parameter IRT model. Consider

$$p(C|\theta, \beta) = \prod_{i=0}^{M-1} \prod_{j=0}^{N-1} \sigma(\alpha_j(\theta_i - \beta_j))^{c_{ij}} (1 - \sigma(\alpha_j(\theta_i - \beta_j)))^{1-c_{ij}}$$

$$\ell(\theta, \beta) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} c_{ij} log(\sigma(\alpha_j(\theta_i - \beta_j))) + (1 - c_{ij}) log(1 - \sigma(\alpha_j(\theta_i - \beta_j))) \tag{1}$$

Derivative of the log-likelihood with respect to θ_i;

$$\frac{\partial \ell}{\partial \theta_i} = \sum_{j=0}^{N-1} \alpha_j(c_{ij} - \sigma(\alpha_j(\theta_i - \beta_j))) \tag{2}$$

Derivative of the log-likelihood with respect to β_j;

$$\frac{\partial \ell}{\partial \beta_j} = \sum_{i=0}^{M-1} \alpha_j(\sigma(\alpha_j(\theta_i - \beta_j)) - c_{ij}) \tag{3}$$

Our algorithm is similar to part A question 2, however we now include a pre-processing step of computing the discrimination index for each question. We provide pseudo-code of our algorithm;

COMPUTE_TEST_SCORES (data):

1: $M \leftarrow$ Number of students
2: $num\_correct \leftarrow \mathbf{0}_M$
3: $num\_attempt \leftarrow \mathbf{0}_M$
4: **for** i in data **do**
5: $cur\_user\_id \leftarrow \text{student}_i$
6: $c \leftarrow \text{correct}_i$
7: $num\_correct[cur\_user\_id]$ += c
8: $num\_attempt[cur\_user\_id]$ += 1
9: **end for**
10: $test\_score\_per\_user \leftarrow num\_correct$ / $num\_attempt$
11: **return** $test\_score\_per\_user$

COMPUTE_DISCRIMINATION_INDICES (sparse_matrix, test_scores):

1: $sparse\_matrix\_scores \leftarrow$ sparse_matrix with additional column for test_scores
2: $sorted\_sm \leftarrow$ sort rows by test_scores
3: $non\_entries \leftarrow$ sum all non entries per question
4: low_score, $high\_score \leftarrow$ split sorted_sm into two equal sized groups
5: {#compute discrimination index column wise for each question j}
6: **for** j in range(len(high_score)) **do**
7: $h_j \leftarrow$ high_score[:, j].sum()
8: $l_j \leftarrow$ low_score[:, j].sum()
9: $t_j \leftarrow$ (sorted_sm.shape[0] - non_entries[j]) / 2
10: $\alpha_j \leftarrow (\text{h}_j - \text{l}_j) / \text{t}_j$
11: **end for**
12: **return** α

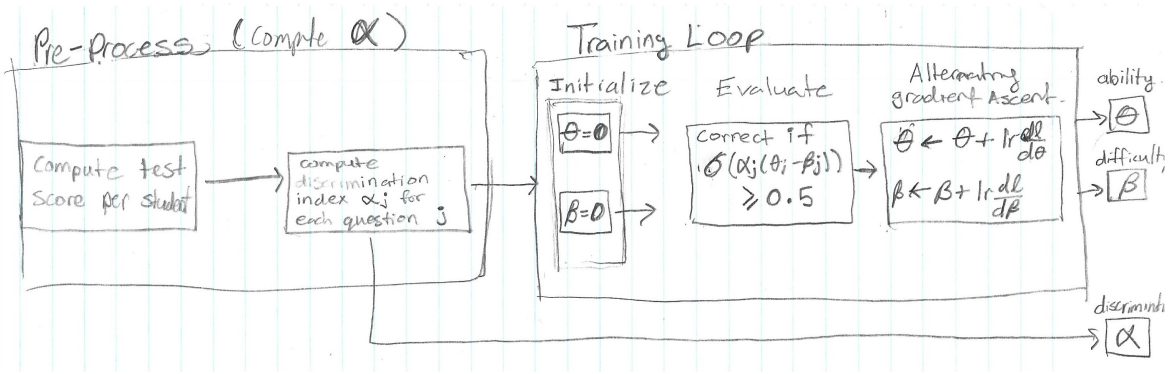
IRT_TWO_PARAM

1: $test\_scores \leftarrow$ **COMPUTE_TEST_SCORES (train_data)**
2: $\alpha \leftarrow$ **COMPUTE_DISCRIMINATION_INDICES (sparse_matrix, test_scores)**
3: $\theta \leftarrow \mathbf{0}$
4: $\beta \leftarrow \mathbf{0}$
5: **for** i in range(iterations) **do**
6: $neg\_lld \leftarrow$ neg_log_likelihood(train_data, θ, β, α)
7: $val\_neg\_lld \leftarrow$ neg_log_likelihood(val_data, θ, β, α)
8: $train\_score \leftarrow$ evaluate(train_data, θ, β, α)
9: $val\_score \leftarrow$ evaluate(val_data, θ, β, α)
10: θ, $\beta \leftarrow$ update_theta_beta
11: **end for**
12: **return** θ, β

*line 6 and 7 is performed using equation (1)
*line 10 is performed using equation (2) and (3) in alternating gradient ascent to maximize log likelihood.

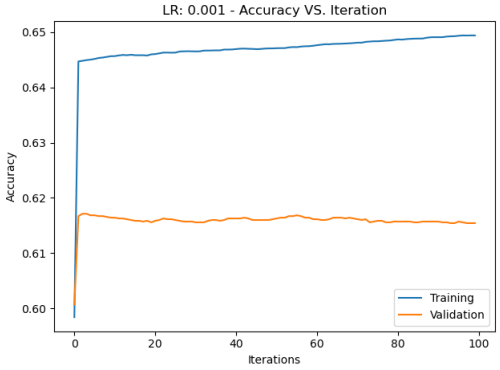
We expect our algorithm to perform better than part A question 2, since the discrimination index is not constant for each question which means that for each question we should have a better representation of a student's probability to answer the question correctly given that each sigmoidal curve for a question will be scaled by its discrimination index. For example, for questions with a high discrimination index we can better represent the rate at which the probability of a correct response increases as student ability increases, whereas in part A question 2 using only the difficulty of each question the rate of increase of the correct response probability was the same for all questions as seen in shape of the sigmoidal curves. We intend for this algorithm to be an optimization to part A question 2, by using the difficulty parameter of a question along with its discrimination index.

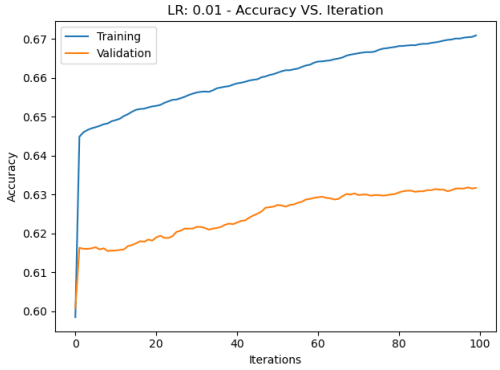
Diagram:

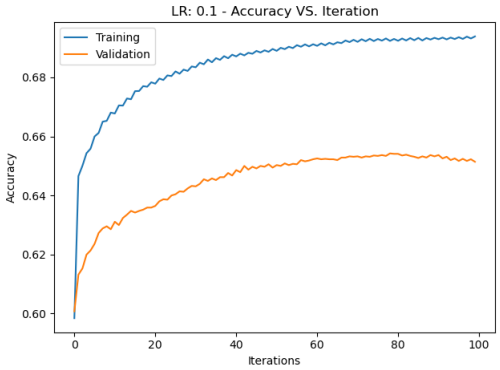


Comparison:

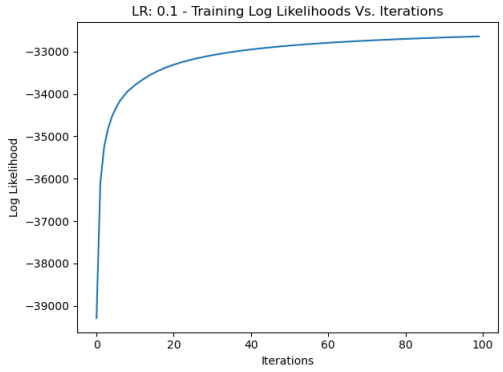
Experiment to test hypothesis, results using the two-parameter IRT model;

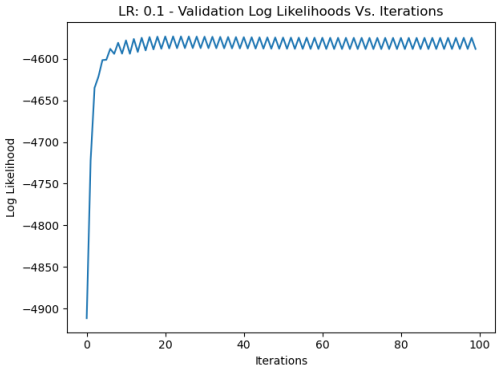






We use the above Accuracy VS. Iteration graphs to determine an appropriate learning rate and number of iterations as LR = 0.1 and Iterations as 100.
With our selected hyperparameters, we report the training curve that shows the training and validation log-likelihoods as a function of iteration.



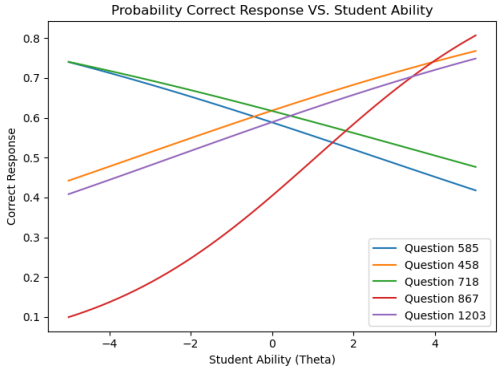


Accuruacy Table				
KNN	Item Resp. Theory	Matrix Fact.	Ensemble	part b
VA (User-based):0.6895 (Item-based):0.6922 TA (User-based):0.6841 (Item-based):0.6816	VA.: 0.7060 TA.: 0.7070	VA: 0.67626 TA.: 0.67146	Bagged VA: 0.709 Bagged TA: 0.7039	VA: 0.6519 TA: 0.6491

(VA is Validation Accuracy and TA is Test Accuracy)

Limitations:
Our hypothesis does not hold and the two-parameter IRT model performs worse than the one-parameter IRT model. A reason why the two-parameter model performed worse could be that when calculating the test scores for each student, the questions attempted by each student was not standard amongst all students. In our scenario, different students attempted different amounts of questions as well as different subsets of questions. The calculation of test score is a ratio of correct answers to questions attempted on a test where each student has attempted the same amount and subset of questions [2]. In our calculation we take each student's test score as the ratio of their correct responses to the number of questions they attempt, where the number and subset of questions varies between students. The test score calculation for each student dictates the high scoring and low scoring groups, which directly affects the discrimination index of each question.

Another observation is that of our train data with 1774 questions, 1234 questions have a discrimination index $\alpha \leq 0.4$. Approximately 70% of the questions in the train data have a relatively low differential capability. Additionally, of the 1234 questions with low differential capability we note that 402 questions have a negative discrimination index. These negative discrimination indices indicate that students with lower abilities answer the questions correctly whereas the higher ability student's answer incorrectly, and so our model's performance suffers. Perhaps our model performs better in scenarios where questions have high discrimination indices. We can see the effects of negative discrimination indices in the following plot for questions 545 and 718.



References:

[1] Slavin, Mozias, *How Does a Grading Curve Work?*. Feb. 2019. URL: https://www.theclassroom.com/how-does-a-grading-curve-work-12146346.html

[2] An, Yung, *Item Response Theory: What It Is and How You Can Use the IRT Procedure to Apply It: Semantic Scholar*

Contributions:

Muneeb Ansari: Part A: Question 2 a,b,c,d | Part B: 1,2,3,4
Asad Aleem: Part A: Question 3
Hokyun Park: Part A: Question 1, 4