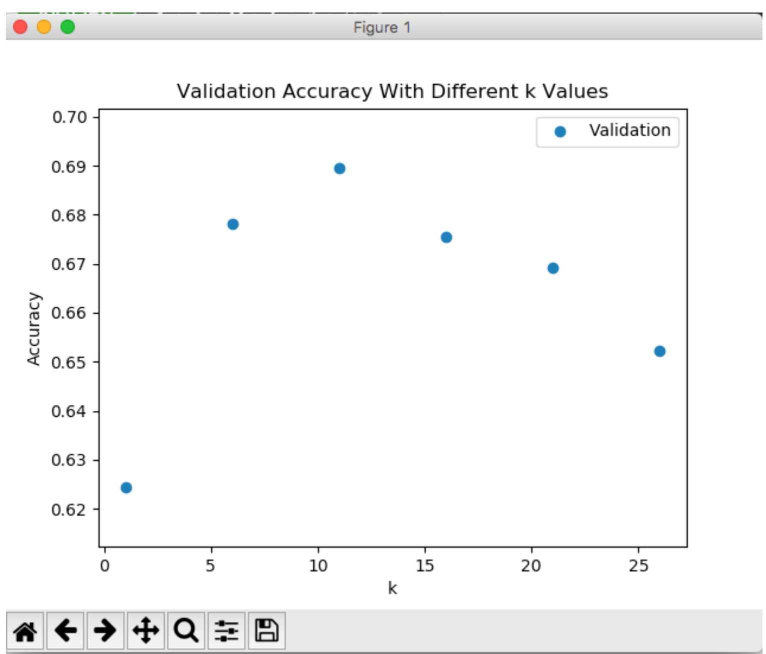
CSC311 Final Project

Haozhen Shen, Nathan Chao, Liang Chen

November 2020

1 Question 1

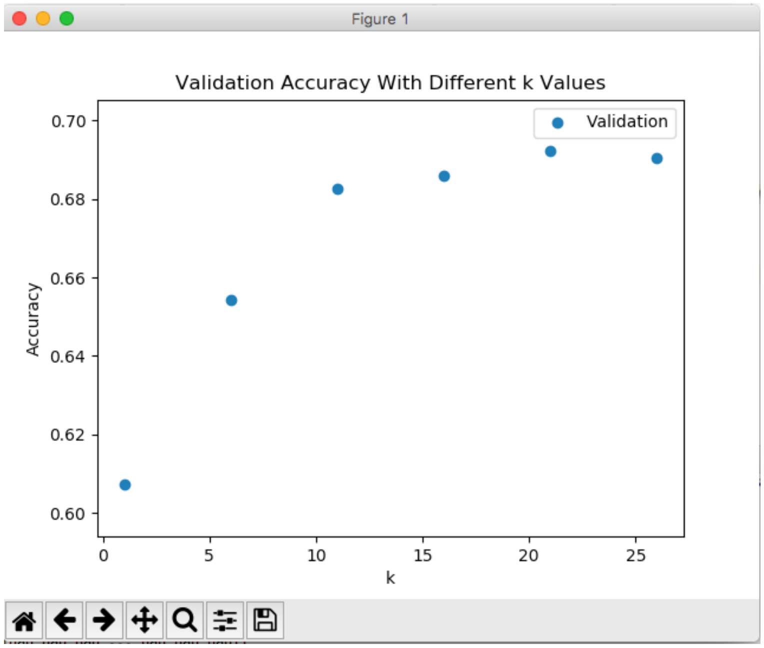
a)



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 k with the highest validation accuracy was $k = 11$ so choose $k^* = 11$. The final test accuracy with $k^* = 11$ is 0.6841659610499576.

c)
The core underlying assumption is that if question A has the same correct and incorrect answers by students as question B, A's correctness by specific students matches that of question B.



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 k with the highest validation accuracy was $k = 21$ so choose $k^* = 21$. The final test accuracy with $k^* = 21$ is 0.6816257408975445.

d) On test data, user-based collaborative filtering performs slightly better.

e)
One limitation of kNN is The Curse of Dimensionality. The data has many dimensions - 542 students and 1774 diagnostic questions. Because there are so many dimensions, nearly all data points will be "far away" from each other. A second limitation of kNN is its space complexity. We need to store the whole data set in memory. In comparison to many other machine learning algorithms in this course, this is bad space complexity. Another limitation of kNN is its time complexity. For every test sample, we must compare it to each sample in the training set. In comparison to many other machine learning algorithms in this course, this is bad time complexity.

2. Item Response Theory

(a) Derive the log-likelihood $\log Prob(\boldsymbol{C}|\boldsymbol{\theta},\boldsymbol{\beta})$ and show the derivative of the log-likelihood with respect to θ_i, β_j separately

<u>*Solution:*</u>

<u>Part 1:</u> format of log-likelihood $\log Prob(\boldsymbol{C}|\boldsymbol{\theta},\boldsymbol{\beta})$

Given response matrix C, whose rows are indexed by students and columns are indexed by questions. The entries of the marks are binary, 1's and 0's. If C_{ij} equals to 1, it means that the student i gets a correct answer to question j.
Assume there are N students and M questions.

For a certain student, some of the questions he answers are correct and some are wrong. Now we write the likelihood of a set of parameter values θ, given a certain pattern of outcomes $\boldsymbol{X}$:

$$\mathcal{L}(\theta|\,\boldsymbol{X}) = P(\boldsymbol{X}|\theta)$$

Based on the assumption that all students are independent, the likelihood function of the entire test is the product of the likelihood function of each student
Thus, we have likelihood function:

$$\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\beta}) = Prob(\boldsymbol{C}|\boldsymbol{\theta},\boldsymbol{\beta}) = \prod_{i=1}^{N}\prod_{j=1}^{M} P(\theta_i,\beta_j)^{m_{ij}} Q(\theta_i,\beta_j)^{(1-m_{ij})}$$

m_{ij} is the indicator for student i correctly answered question j

Log-likelihood:

$$L(\boldsymbol{\theta},\boldsymbol{\beta}) = \log Prob(\boldsymbol{C}|\boldsymbol{\theta},\boldsymbol{\beta}) = \log\prod_{i=1}^{N}\prod_{j=1}^{M} P(\theta_i,\beta_j)^{m_{ij}} Q(\theta_i,\beta_j)^{(1-m_{ij})}$$

$$L(\boldsymbol{\theta},\boldsymbol{\beta}) = \sum_{i=1}^{N}\sum_{j=1}^{M}[m_{ij}\log\left(\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}\right) + (1-m_{ij})\log\left(\frac{1}{1+\exp(\theta_i-\beta_j)}\right)]$$

$$L(\boldsymbol{\theta},\boldsymbol{\beta}) = \sum_{i=1}^{N}\sum_{j=1}^{M}[m_{ij}(\theta_i-\beta_j) - \log(1+\exp(\theta_i-\beta_j))]$$

<u>Part 2:</u> partial derivatives respect to θ_i

$$\frac{\partial}{\partial\theta_i}L(\boldsymbol{\theta},\boldsymbol{\beta}) = \frac{\partial}{\partial\theta_i}\{\sum_{j=1}^{M}[m_{ij}(\theta_i-\beta_j) - \log(1+\exp(\theta_i-\beta_j))]\}$$

$$\frac{\partial}{\partial \theta_i} L(\boldsymbol{\theta}, \boldsymbol{\beta}) = \sum_{j=1}^{M} [m_{ij} - \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)}]$$

<u>Part 3:</u> partial derivatives respect to β_j

$$\frac{\partial}{\partial \beta_j} L(\boldsymbol{\theta}, \boldsymbol{\beta}) = \frac{\partial}{\partial \beta_j} \sum_{i=1}^{N} [m_{ij}(\theta_i - \beta_j) - \log(1 + \exp(\theta_i - \beta_j))]$$

$$\frac{\partial}{\partial \beta_j} L(\boldsymbol{\theta}, \boldsymbol{\beta}) = \sum_{i=1}^{N} [-m_{ij} + \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)}]$$

(b) Implement missing functions that perform alternating gradient descent on $\boldsymbol{\theta}, \boldsymbol{\beta}$ to maximize the log-likelihood and report the hyperparameters selected.
With the chosen hyperparameters, report the training curves that shows the training and validation log-likelihoods as a function of iteration.

<u>Solution:</u>

Part 1: Basic thoughts of code implementation
To find $\boldsymbol{\theta}, \boldsymbol{\beta}$ so as to maximize log-likelihood equivalent to minimize negative log-likelihood

$$\frac{\partial}{\partial \theta_i} L(\boldsymbol{\theta}, \boldsymbol{\beta}) = \sum_{j=1}^{M} [m_{ij} - \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)}]$$

$$\frac{\partial}{\partial \theta_i} [-L(\boldsymbol{\theta}, \boldsymbol{\beta})] = \sum_{j=1}^{M} [-m_{ij} + \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)}]$$

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial}{\partial \theta_i} [-L(\boldsymbol{\theta}, \boldsymbol{\beta})]$$

$$\theta_i \leftarrow \theta_i - \alpha \sum_{j=1}^{M} [-m_{ij} + \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)}]$$

$$\frac{\partial}{\partial \beta_j} L(\boldsymbol{\theta}, \boldsymbol{\beta}) = \sum_{i=1}^{N} [-m_{ij} + \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)}]$$

$$\frac{\partial}{\partial \beta_j} [-L(\boldsymbol{\theta}, \boldsymbol{\beta})] = \sum_{i=1}^{N} [m_{ij} - \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)}]$$

$$\beta_j \leftarrow \beta_j - \alpha \frac{\partial}{\partial \beta_j} [-L(\boldsymbol{\theta}, \boldsymbol{\beta})]$$

$$\beta_j \leftarrow \beta_j - \alpha \sum_{i=1}^{N} [m_{ij} - \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)}]$$

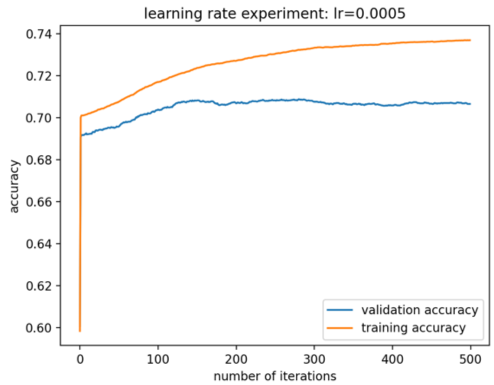
Part 2: tune hyperparameters

Since there are two hyperparameters, learning rate and number of iterations to tune, we did the following experiments:

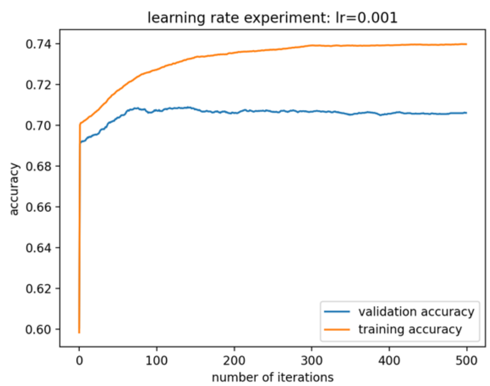
1) Experiment on learning rate

   Fixed number of iterations = 500, and initial values of theta and beta are all 1's

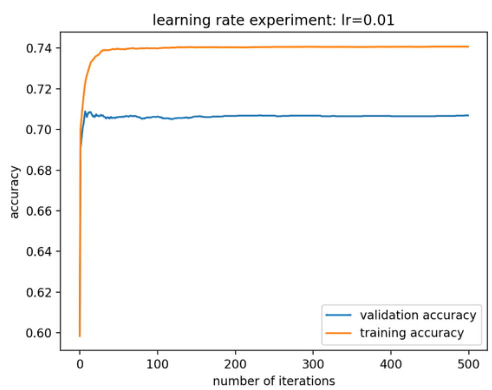
   - Learning rate = 0.0005



   - Learning rate = 0.001



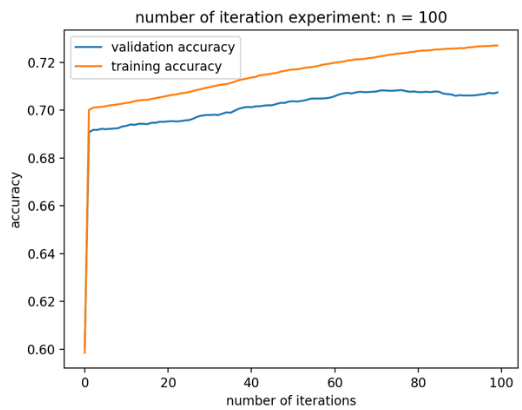
   - Learning rate = 0.01



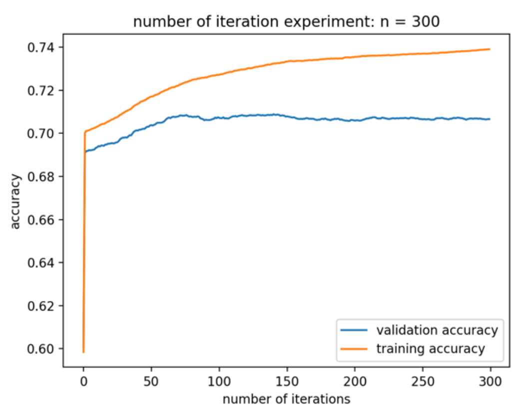
   From all three plots, we can tell that the accuracy rate curves rise rapidly initially and become stable after some number of iterations. Validation accuracy is always lower than training accuracy. The accuracy of training set is at around 0.74, and the accuracy of validation set is about 0.71.

We can also tell that when learning rate equals to 0.01, curves are too steep; when learning rate equals to 0.0005, accuracy is still slightly rising after 500 iterations, thus it rises relatively too slow. Learning rate of 0.001 is neither too fast nor too slow.

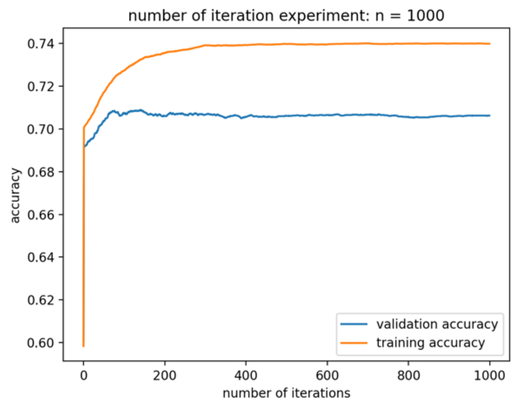
2) Experiment on number of iterations
   Fixed learning rate =0.001, and initial values of theta and beta are all 1's
   - Number of iterations = 100



   - Number of iterations = 300



   - Number of iterations = 1000

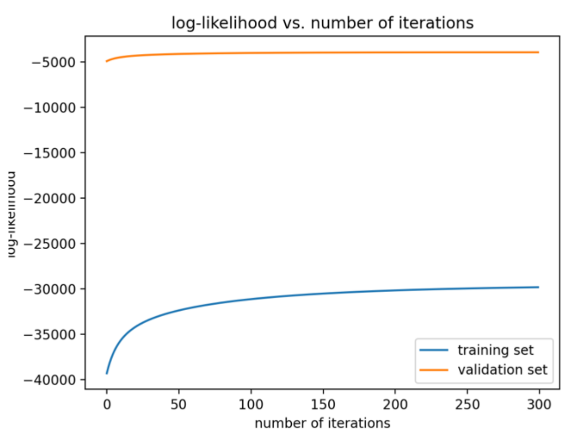


We can tell that after 300 iterations, both training and validation accuracy reach their maximum values at about 0.74 and 0.70 separately and stay stable. Thus, 300 iterations are acceptable in this case.

Based on the experiments above, we finally choose the following hyperparameters as the final parameters:
Initial values of theta and beta are 1
Learning rate = 0.001
Number of iterations = 300

Part 3: plots report

Fixed learning rate = 0.001, number of iterations = 300



We can tell from the plot above that, both training and validation set log-likelihood are negative and rise as number of iterations increases. With number of iterations gets larger, the slopes of both training and validation sets become shallower. This trend is consistent with what we learned: as more iterations go, it converges to its extreme value.

(c) With the implemented code, report the final validation and test accuracy

After running the code, we get the following accuracy output:
final accuracy of validation set is 0.7066045723962744

final accuracy of test set is 0.7053344623200677

We can tell that validation set's accuracy is slightly higher than test set, but they are still quite similar.

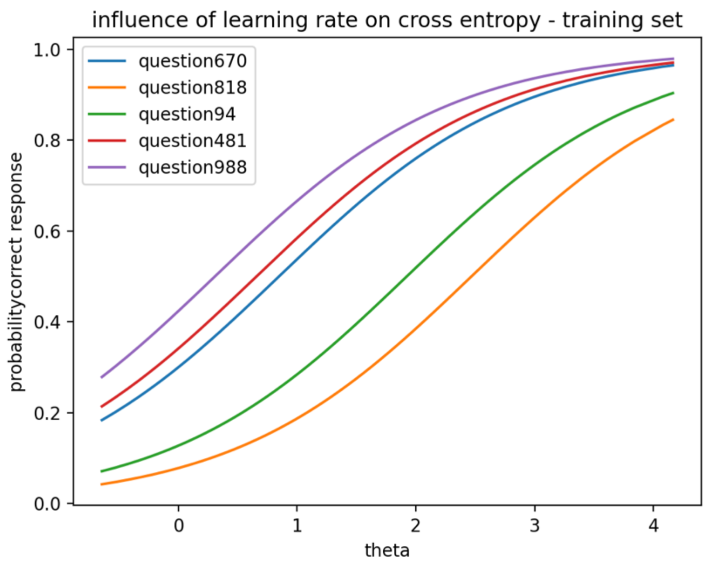
(d) Select five questions and use the trained $\boldsymbol{\theta}, \boldsymbol{\beta}$ to plot five curves on the same plot that shows the probability of the correct response $p(c_{ij})$ as a function of θ given a question j. Comment on the shape of the curve and briefly describe what these curves represent.

Solution:
Part 1: ideas to get the formula
Given

$$P\left(c_{ij} = 1 \,\middle|\, \theta_i, \beta_j\right) = \frac{\exp\left(\theta_i - \beta_j\right)}{1 + \exp\left(\theta_i - \beta_j\right)}$$

Now treat the difficulty of a certain question j , β_j, be a constant, then probability of the correct response $p(c_{ij})$ is a function of θ_i

Part 2: plot



Part 3: Interpretation

We randomly selected 5 questions, which are Q818, Q94, Q670, Q481, Q988

From the plot we can tell that these 5 curves have similar shape – "s" curve

For all 5 curves, we can tell that, as θ_i gets larger, the probability of correct response gets larger as well. This is reasonable, since the greater the ability of the student has, the higher chance he or she has to give the correct answer. The probability is bounded by 0 and 1 obviously.

We can also tell that, for the same θ value, the probability to answer correctly for the 5 randomly chosen questions are:
Prob(correct) for question 988 > Prob(correct) for question 481 > Prob(correct) for question 670 > Prob(correct) for question 94 > Prob(correct) for question 818.
Since for the same student, his or her ability is fixed, then the probability of answer correctly is decided by the difficulty of given questions. The simpler the given question is, the higher probability the question is answered correctly. Thus, we would guess: question 818 is the most challenging one among the 5 chosen questions, then question94, then question 670, then question481, question 988 is the simplest one among the 5 question.
To prove it, we printed the difficulty of these 5 questions, which are:
selected five questions has the following difficulty level:
question 670 has difficulty[0.84724177],
question 818 has difficulty[2.46655858],
question 94 has difficulty[1.92220531],
question 481 has difficulty[0.65751092],
question 988 has difficulty[0.30782227]
From the numerical representations of difficulty of questions, we can tell that the difficulty has:
Q818 > Q94 > Q670 > Q481 > Q988
This is consistent with what we observed in the plot

3. Matrix Factorization OR Neural Networks

(i) Option 1: Matrix Factorization

(a)

In [14]:
```
run starter_code/part_a/matrix_factorization.py
```

```
Evaluating SVD with k = 1   Current max accuracy is 0.6428 with k = 1
Evaluating SVD with k = 2   Current max accuracy is 0.6579 with k = 2
Evaluating SVD with k = 3   Current max accuracy is 0.6583 with k = 3
Evaluating SVD with k = 4   Current max accuracy is 0.6583 with k = 3
Evaluating SVD with k = 5   Current max accuracy is 0.659 with k = 5
Evaluating SVD with k = 6   Current max accuracy is 0.659 with k = 5
Evaluating SVD with k = 7   Current max accuracy is 0.6592 with k = 7
Evaluating SVD with k = 8   Current max accuracy is 0.6603 with k = 8
Evaluating SVD with k = 9   Current max accuracy is 0.6613 with k = 9
Evaluating SVD with k = 10   Current max accuracy is 0.6613 with k = 9
Evaluating SVD with k = 11   Current max accuracy is 0.6613 with k = 9
Evaluating SVD with k = 12   Current max accuracy is 0.6613 with k = 9
Evaluating SVD with k = 13   Current max accuracy is 0.6613 with k = 9
Evaluating SVD with k = 14   Current max accuracy is 0.6613 with k = 9
Evaluating SVD with k = 15   Current max accuracy is 0.6613 with k = 9
Evaluating SVD with k = 16   Current max accuracy is 0.6613 with k = 9
Evaluating SVD with k = 17   Current max accuracy is 0.6613 with k = 9
Evaluating SVD with k = 18   Current max accuracy is 0.6613 with k = 9
Evaluating SVD with k = 19   Current max accuracy is 0.6613 with k = 9
Evaluating SVD with k = 20   Current max accuracy is 0.6613 with k = 9

Maximum accuracy achieved on validation set = 0.6613  with k = 9.

Accuracy achieved on test set = 0.6588 with our selected k = 9.
```

(b)

Note that our training matrix is extremely sparse in other words containing lots of missing values. Thus we fill in these values using the average on the each item before doing SVD. This might not be a good practice since because by filling in such values in all the sparse locations could end up having more noise terms than the actual meaningful good data points.

(c)

We have our implementations below with doctrings and comments excluded,

In [15]:
```
def update_u_z(train_data, lr, u, z):
    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]
    u[n] = u[n] + lr * (c - np.dot(u[n], z[q].T))*z[q]
    z[q] = z[q] + lr * (c - np.dot(u[n], z[q].T))*u[n]
    return u, z

def als(train_data, k, lr, num_iteration):
    u = np.random.uniform(low=0, high=1 / np.sqrt(k),
                          size=(len(set(train_data["user_id"])), k))
    z = np.random.uniform(low=0, high=1 / np.sqrt(k),
                          size=(len(set(train_data["question_id"])), k))
    for i in range(num_iteration):
        u, z = update_u_z(train_data, lr, u, z)
    print(squared_error_loss(train_data, u, z))
    mat = u.dot(z.T)
    return mat
```

(d)

After some parameter tunning we learn our model with a 0.08 learning rate with 70000 number of iterations with number of factor k = 130.

In [22]:
```
run starter_code/part_a/matrix_factorization.py
```

```
Evaluating k = 5, current max accuracy achieved on k = 0 with 0
Evaluating k = 10, current max accuracy achieved on k = 5 with 0.6922
Evaluating k = 20, current max accuracy achieved on k = 10 with 0.6928
Evaluating k = 40, current max accuracy achieved on k = 10 with 0.6928
Evaluating k = 70, current max accuracy achieved on k = 10 with 0.6928
Evaluating k = 100, current max accuracy achieved on k = 10 with 0.6928
Evaluating k = 130, current max accuracy achieved on k = 10 with 0.6928
Evaluating k = 160, current max accuracy achieved on k = 130 with 0.698
Evaluating k = 190, current max accuracy achieved on k = 130 with 0.698

Maximum accuracy achieved on validation set = 0.698  with k = 130.

Accuracy achieved on test set = 0.7036 with our selected k = 130.
```

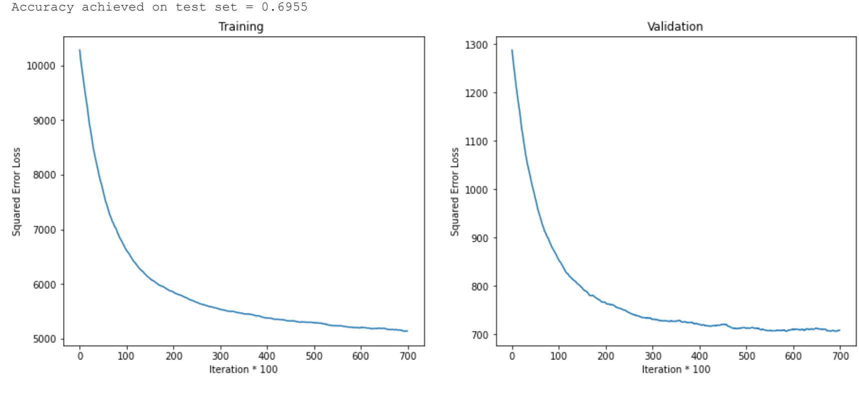
(e)

With k = 130 and learning rate 0.08 with 70000 number of iterations we have our validation squared error loss plot as follows,

In [31]:
```
run starter_code/part_a/matrix_factorization.py
```

```
Maximum accuracy achieved on validation set = 0.6926.

Accuracy achieved on test set = 0.6955
```



(f)

Note that since we are predicting values $\in [0,1]$ a squared error loss might not be that suitable. Instead we can consider a cross entropy loss,

$$y = \sigma(z) = \frac{1}{1+e^{-z}}$$

$$\mathcal{L}_{CE}(y,t) = -tlogy - (1-t)log(1-y)$$

Where in our contex our objective becomes,

$$min \sum_{(x,m)\in\mathcal{O}} -C_{nm}log(\sigma(\mathbf{u}_n^T\mathbf{z}_m)) - (1-C_{nm})log(1-\sigma(\mathbf{u}_n^T\mathbf{z}_m))$$

4. Ensemble

We choose to train one KNN, one Item Response models, and one Matrix factorization model on bootstrapped training data and obatain results by averaging the predicted correctness.

We obatain our 3 bootstaped data that has the same size of our original traning data using the folowing algorithm,

In [1]:
```
def resample(data, m):
    size = len(data["question_id"])
    bags_of_data = []
    num_bags = 0
    while num_bags < m:
        bag = {'user_id': [], 'question_id': [],
               'is_correct': []}
        print("Creating bag number {}". format(num_bags + 1))
        while len(bag["question_id"]) < size:
            index = np.random.choice(size, 1)[0]
            uid = data['user_id'][index]
            qid = data['question_id'][index]
            aid = data['is_correct'][index]
            bag['user_id'].append(uid)
            bag['question_id'].append(qid)
            bag['is_correct'].append(aid)
        bags_of_data.append(bag)
        num_bags += 1
    return bags_of_data
```

Next we construct our ensemble by training different models on such data,

In [ ]:
```
# Specify number of bags to train on each model
########################################################
vanilla_mf = 1
knn_item = 1
knn_user = 0
vanilla_irt = 1
irt_with_bias = 0
mf_with_bias = 0
mf_with_bias_gender = 0
mf_with_CE_loss = 0

NUM_OF_BAGS = (knn_item + knn_user + vanilla_irt + irt_with_bias +
               vanilla_mf + mf_with_bias + mf_with_bias_gender + mf_with_CE_loss)
########################################################
# Generate bootstrapped data
bags = resample(train_data, NUM_OF_BAGS)
N = 542
M = 1774
sum_conf_mat = np.zeros((N, M))
# Ensemble of each model specified above
for i in range(len(bags)):
    if i < NUM_OF_BAGS - irt_with_bias - vanilla_irt:
        if i < vanilla_mf:
            print("Fitting vanilla Matrix Factorization on data set number: {}.".format(i + 1))
            mat = als(bags[i], 120, 0.08, 140000)

        elif i < vanilla_mf + mf_with_bias_gender:
            print("Fitting Matrix Factorization with user bias, question bias, and gender bias "
                  "on data set number: {}.".format(i + 1))
            mat = als_with_bias_and_gender(bags[i], 70, 0.04, 350000, 0.02, gender_matrix)

        elif i < vanilla_mf + mf_with_bias_gender + mf_with_CE_loss:
            print("Fitting Matrix Factorization with cross entropy loss on data set number: {}.".format(i +
            mat = als_logistic(bags[i], 240, 0.08, 280000, 0.02)

        elif i < vanilla_mf + mf_with_bias_gender + mf_with_CE_loss + knn_item:
            print("Fitting Nearest neighbor by item on data set number: {}.".format(i + 1))
            nbrs = KNNImputer(n_neighbors=21)
            new_mat = resampled_data_to_matrix(bags[i])
            mat = nbrs.fit_transform(np.transpose(new_mat)).T
        elif i < vanilla_mf + mf_with_bias_gender + mf_with_CE_loss + knn_item + knn_user:
            print("Fitting Nearest neighbor by user data set number: {}.".format(i + 1))
            nbrs = KNNImputer(n_neighbors=11)
            new_mat = resampled_data_to_matrix(bags[i])
            mat = nbrs.fit_transform(new_mat)
        else:
            mat = als_with_bias(bags[i], 120, 0.08, 230000, 0.001)

        sum_conf_mat += mat
    else:
        if i < NUM_OF_BAGS - vanilla_irt:
            print("Fitting Item Response Model with bias on data set number: {}.".format(i + 1))
            num_iteration = 232
            theta = np.ones((N, 1))
            beta = np.ones((M, 1))
            alpha = np.ones((M, 1))
            for _ in range(num_iteration):
                new_mat = sparse.csc_matrix(resampled_data_to_matrix(bags[i]))
                theta, beta, alpha = update_theta_beta_alpha(new_mat, 0.001, theta, beta, alpha)
            sum_conf_mat += get_pred_matrix_IRT(theta, beta, alpha)
        else:
            print("Fitting vanilla Item Response Model on data set number: {}.".format(i + 1))
            num_iteration = 76
            theta = np.ones((N, 1))
            beta = np.ones((M, 1))
            for _ in range(num_iteration):
                new_mat = sparse.csc_matrix(resampled_data_to_matrix(bags[i]))
                theta, beta = update_theta_beta(new_mat, 0.001, theta, beta)
            sum_conf_mat += get_pred_matrix_no_alpha_IRT(theta, beta)

avg_mat = sum_conf_mat/NUM_OF_BAGS
final_acc_valid = sparse_matrix_evaluate(val_data, avg_mat)
final_acc_test = sparse_matrix_evaluate(test_data, avg_mat)
```

Next we fit our three models to the bootstrapped data which has the same size of our original training data, after we done training we average our user-item interaction matrix obtained from the three models and take the average of the three. In the end we use our averaged prediciton matrix and evaluate on validation set and test set.

In [10]:
```
run starter_code/part_a/ensemble
```

```
Creating bag number 1
Creating bag number 2
Creating bag number 3
Fitting naive Matrix Factorization on data set number: 1.
Fitting Nearest neighbor by item on data set number: 2.
Fitting Item Response Model with bias on data set number: 3.

Maximum accuracy of mf achieved on validation set = 0.7125.

Accuracy of mf achieved on test set = 0.694
```

We see that after ensembling our three base models and traning them on three bootstrapped data the final prediction matrix obtained has an improve in performace over all the base models just by themselves. From the bias-variance decomposition lecture we see that how ensembling multiple moderate learners togetheres reduces variance, this fully applies in our example since we have three moderate learners that learn through different methods and it is resonable to say that the errors of the three models are mostly un-corellated thus through bagging and ensembling we should expect such improvement in accuracy.

Part b.

1. Formal Description:

We consider a approach based on matrix factoriazation with the extension of a user bias, question bias, and a gender bias.

First, we recap on the vanilla matrix factorization approach with mean squared error loss

$$R_{nq} = u_n z_q$$

$$\min_{\mathbf{U},\mathbf{Z}} \frac{1}{2} \sum_{(n,q)\in\mathcal{O}} \left(C_{nq} - \mathbf{u}_n^T\mathbf{z}_q\right)^2$$

Note that in this vanilla version of matrix factorization the model caputures the iteraction between students and questions that whether for a specific student for a specific question will have a correct or incoreect response. However, when considering student responses it is really likely there are effects that are limited to a specific student or question and is independent of iteraction.

Consider the case where a student has a solution manuel of some of the responses. Then he is more likely to score correctly on a repsonse independent of the student question interaction. Similarly, where a question is at a level to high above the student's abilities then for any students attempting this question they are unlikely to get it correct independet of which student attemtps. Thus we propose two bias terms to try in capture this type of behavior so that our model accounts for user bias and question bias accordingly.

Consider the new model,

$$R_{nq} = u_n z_q + b + \theta_n + \beta_q$$

$$\min_{\mathbf{U},\mathbf{Z}} \frac{1}{2} \sum_{(n,q)\in\mathcal{O}} \left(C_{nq} - \mathbf{u}_n^T\mathbf{z}_q - b - \theta_n - \beta_q\right)^2$$

Where b is the average score through all observed responses, θ_n bias of student n, and β_q bias of question q.

Now that our model accounts for student question interactions and student, question biases. In order to make our model mor meaningful and have more accurate predictions of student responses. We make use of the student metadata Gender and incorporte a gender bias term to our model.

Thus we have,

$$R_{nq} = u_n z_q + b + \theta_n + \beta_q + g(n)$$

$$\min_{\mathbf{U},\mathbf{Z}} \frac{1}{2} \sum_{(n,q)\in\mathcal{O}} \left(C_{nq} - \mathbf{u}_n^T\mathbf{z}_q - b - \theta_n - \beta_q - g(n)\right)^2$$

Where $g(n)$ acounts for the gender bias of user n.

At last, to account for overfitting we apply L2 regularization to our model variabels. Thus, we have our final model,

$$R_{nq} = u_n z_q + b + \theta_n + \beta_q + g(n)$$

$$\min_{\mathbf{U},\mathbf{Z}} \frac{1}{2} \sum_{(n,q)\in\mathcal{O}} \left(C_{nq} - \mathbf{u}_n^T\mathbf{z}_q - b - \theta_n - \beta_q - g(n)\right)^2 + \lambda\left(\|\mathbf{u}_n\|^2 + \|\mathbf{z}_q\|^2 + \|\theta_n\|^2 + \|\beta_q\|^2 + \|g(n)\|^2\right)$$

2. Figure or Diagram:

We take a look of the following figure to ease our understanding of the process. We illustrate how user-item matrix can be factorized as latent factors representing latent user and item features respectively.

$$R = \mathbf{U}\mathbf{Z}$$

Where R represents the etsimated User-item matrix by multiplying the User latent factors $\mathbf{U}$ and Item latent factors $\mathbf{Z}$ in our case students and questions respectively.

In [11]:
```
from IPython.display import Image
Image(filename='mf.png')
```

Out[11]:

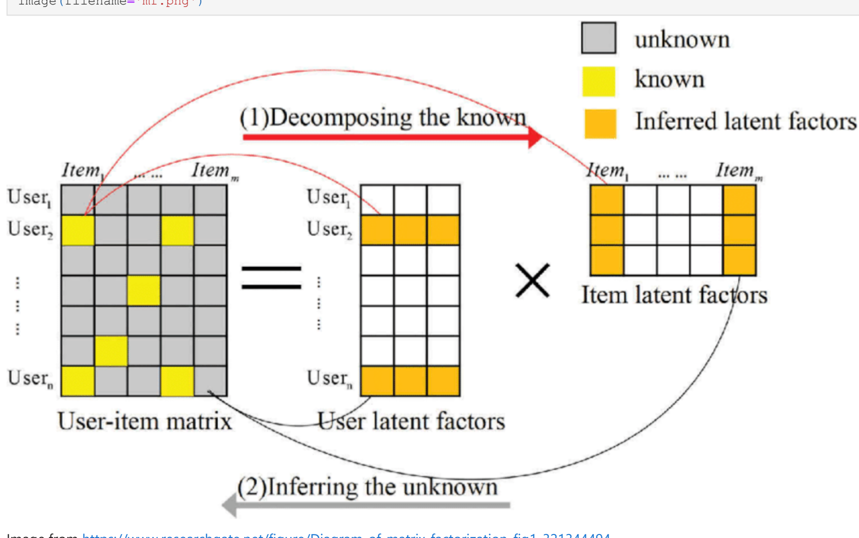


Image from https://www.researchgate.net/figure/Diagram-of-matrix-factorization_fig1_321344494

In addition to the above we also add the three $n \times m$ bias matrices element wise to R,

Where user bias,

$$\boldsymbol{\theta} = \begin{pmatrix} \theta_1 & \theta_1 & \theta_1 & \dots & \theta_1 \\ \theta_2 & \theta_2 & \theta_2 & \dots & \theta_2 \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \theta_n & \theta_n & \theta_n & \dots & \theta_n \end{pmatrix}$$

Question bias,

$$\boldsymbol{\beta} = \begin{pmatrix} \beta_1 & \beta_2 & \beta_3 & \dots & \beta_m \\ \beta_1 & \beta_2 & \beta_3 & \dots & \beta_m \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \beta_1 & \beta_2 & \beta_3 & \dots & \beta_m \end{pmatrix}$$

At last the gender bias,

$$\mathbf{G} = \begin{pmatrix} g(1) & g(1) & g(1) & \dots & g(1) \\ g(2) & g(2) & g(2) & \dots & g(2) \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ g(n) & g(n) & g(n) & \dots & g(n) \end{pmatrix}$$

Thus our resulting R equals,

$$R = \begin{pmatrix} R_{11} & R_{12} & R_{13} & \dots & R_{1m} \\ R_{21} & R_{22} & R_{23} & \dots & R_{2m} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ R_{n1} & R_{n2} & R_{n3} & \dots & R_{nm} \end{pmatrix} = UZ + \begin{pmatrix} \theta_1 & \theta_1 & \theta_1 & \dots & \theta_1 \\ \theta_2 & \theta_2 & \theta_2 & \dots & \theta_2 \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \theta_n & \theta_n & \theta_n & \dots & \theta_n \end{pmatrix} + \begin{pmatrix} \beta_1 & \beta_2 & \beta_3 & \dots & \beta_m \\ \beta_1 & \beta_2 & \beta_3 & \dots & \beta_m \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \beta_1 & \beta_2 & \beta_3 & \dots & \beta_m \end{pmatrix} + \begin{pmatrix} g(1) & g(1) & g(1) & \dots & g(1) \\ g(2) & g(2) & g(2) & \dots & g(2) \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ g(n) & g(n) & g(n) & \dots & g(n) \end{pmatrix}$$

3. Comparison or Demonstration:

We compare our augmented model with the vanilla version matrix factorization.

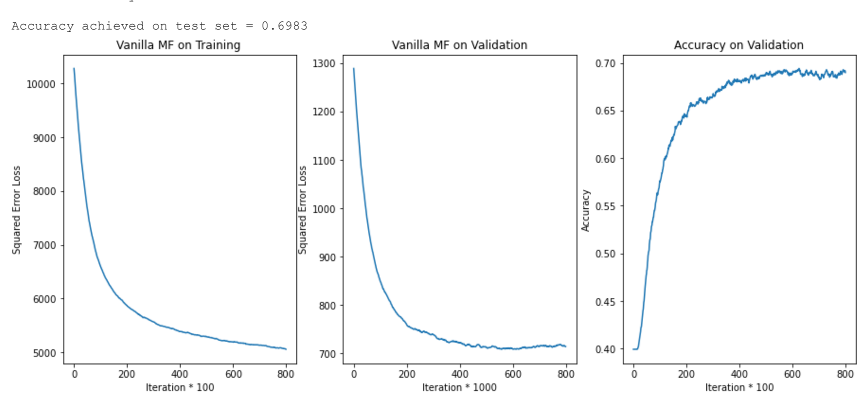
We first take a look of the vanilla matrix factorization model we train the vanilla matrix factorization with k = 130 and learning rate 0.08 with 70000 number of iterations.

In [18]:
```
run starter_code/part_a/matrix_factorization
```

```
Maximum accuracy achieved on train set = 0.7368.

Maximum accuracy achieved on validation set = 0.6901.

Accuracy achieved on test set = 0.6983
```



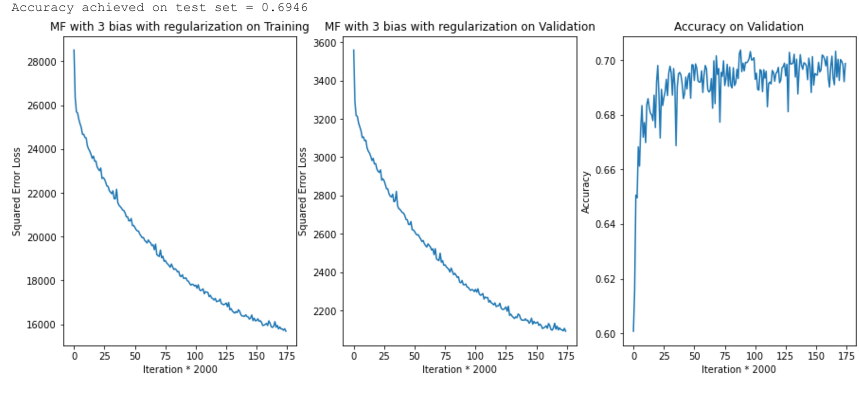
Next we take see how our new aumented model performs, we train our augmented model with k = 70, learning rate= 0.04, regularization parameter = 0.02 with 350000 number of iterations,

In [21]:
```
run starter_code/part_a/matrix_factorization
```

```
Maximum accuracy achieved on train set = 0.7666.

Maximum accuracy achieved on validation set = 0.7001.

Accuracy achieved on test set = 0.6946
```



Comparing the two models we see that in temrs of Mean Squared error loss, our new proposed model has a higher loss in general and also decays more slowly. This is resonable since we incorporated regularization of each the parameters to our model in order to prevent overfitting.

In terms of accuracy, we see that our new proposed model has slight improvements. We also observe as we get near the 70 percent mark both model's accuracy starts to plateu but our new model remains with some small flutuations. Both model seems like to converge at 70 percent accuracy mark.

Disentangle the effect of each component.

In order to disentangle the benefits of our proposed models, we break our model into smaller augmentation components and then add in each components one by one to see the effects.

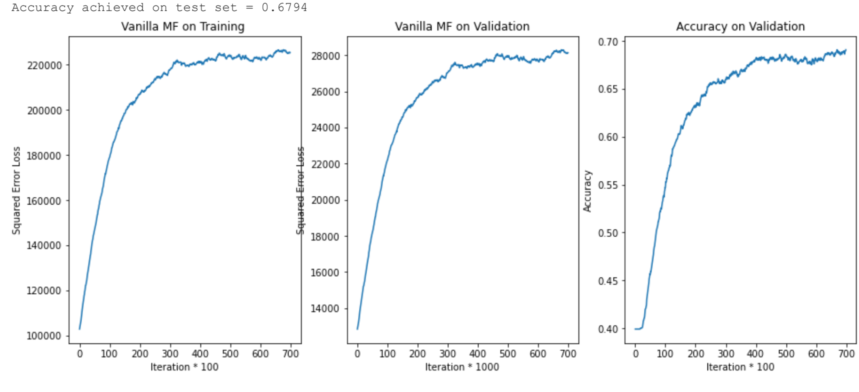
We first consider the benefits of adding in the regularization term, we examine the performance of the vanilla matrix factorization with regularization. We train the model with k = 130, and learning rate 0.08, regularization coefficient = 0.05 with 70000 number of iterations.

In [22]:
```
run starter_code/part_a/matrix_factorization
```

```
Maximum accuracy achieved on train set = 0.7206.

Maximum accuracy achieved on validation set = 0.6904.

Accuracy achieved on test set = 0.6794
```



We see that after adding the regularization component into our model with the same number of iteration of the previous vanilla matrix factorization our loss were able to decay at a slower pace. Thus it it resonable to say that our regularization does reduce overfitting.

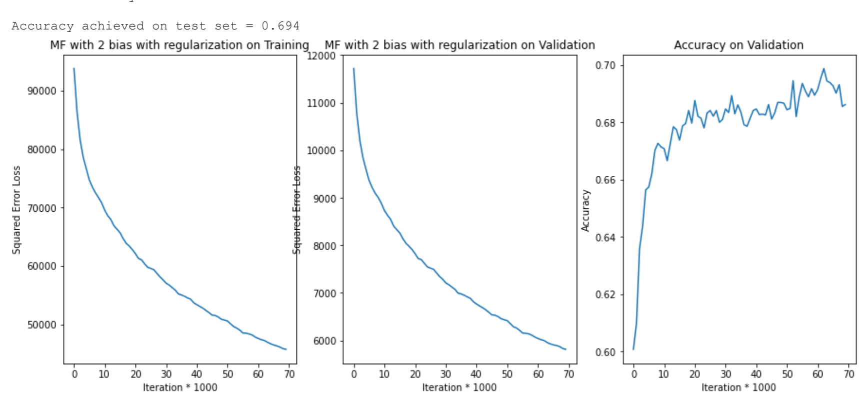
Next we consider adding the user bias and question bias into our model. We train our model with k = 130, and learning rate 0.08, regularization coefficient = 0.05 with 70000 number of iterations.

In [25]:
```
run starter_code/part_a/matrix_factorization
```

```
Maximum accuracy achieved on train set = 0.7191.

Maximum accuracy achieved on validation set = 0.6932.

Accuracy achieved on test set = 0.694
```



Compared to the previous model, we see that we have slight increase in validation acccuracy. Thus we can say that adding such bias terms does improve the model. Compared to the resultes of the final model we see that after adding the gender bias into our model, we have further improve in validation accuracy thus we can also say adding the gender bias term also improves our model a bit.

4. Limitations:

There are still multiple limitation to the model, where some are caused by the nature of the problem, and some are due to the architecture of our model. We discuss some main settings where our model behaves poorly.

1. When they are no responses or only a few responses for some students or for some questions.

In this case our model only has little information about these specific students or questions, and thus cannot make meaningful predictions for them.

Note that the problem of data sparsity in such collaborative filtering problems does not only limit to our context but is in fact a general problem in recommender systems. Since our model is fit through a mean squared error loss, model predictions would deeply rely on number of responses of these user-item pairs in order to make prediction for them.

To address this problem, one might consider a different recommending approach. Instead of a collaborative filtering system, we might want to try a content based system which only relies on user preferences and item descriptions.

1. When many new students or many new questions are added into the model.

Since we are trying to predict student reponses on online education services, the amount of users and questions can get extremly large. There are mutiple problems that this will cause, first our model's training time would scale significanly as datasize increases. Second, as we add new students and new question into the model we do not have any previous information on them, thus we cannot make meaningful predictions for them until we have suffient responses from them. Also, as dimension gets even larger we have the curse of dimensinality which makes the problem of datasparsity even worse.

To address this problem, one might also consider a different recommending approach. Instead of a collaborative filtering system, we might want to try a content based system which only relies on user preferences and item descriptions.

1. When student does not respond to questions according to their ability or question responses has no reflection on a student.

In order for our model to be meaningful and have accurate predictions, we assume that every student-question pair response some how reflect the student's ability as well as the question's difficulty. However, since most of the questions in such online education platforms are multiple choice based questions, in the case where a student blindly guesses a question and submits a response our model would not be able to capture such behaviors and thus such responses will contribute as pure noise to our model. Another case is where the question is not meaningful at all. In this case reponses for such questions will also contribute as pure noise.

To address this problem, one way is to try incorporating such behaviors of students and questions into the model, namely a guessing parameter and a rating parameter of how good the problem is. We might want to account for the probability of a student guessing and the rating parameter of the problem when designing our models.