CSC311 Final Project

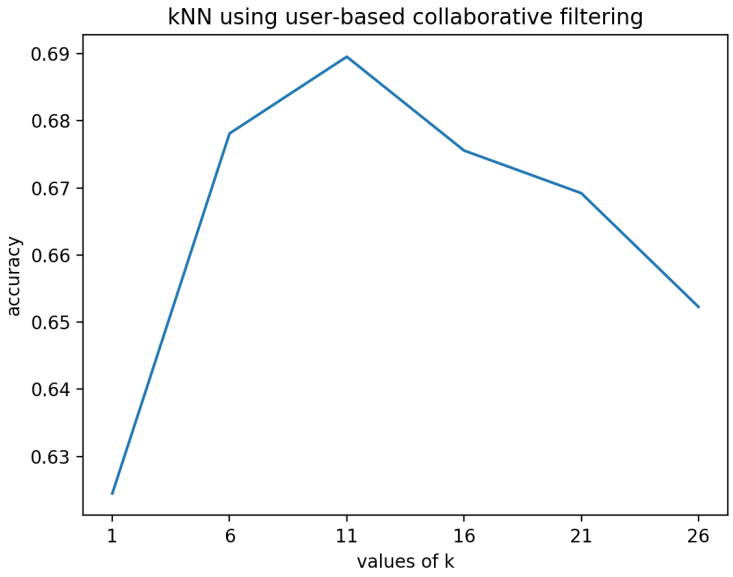
Shujie Deng, Lingfei Li, Saifei Liao

December 2021

1 Part A

1.1 k-Nearest Neighbor. (Lingfei Li)

(a) Plot and report the accuracy on the validation data as a function of k.



For k = 1, 6, 11, 16, 21 and 26, the accuracy is:

```
Validation Accuracy: 0.6244707874682472
Validation Accuracy: 0.6780976573525261
Validation Accuracy: 0.6895286480383855
Validation Accuracy: 0.6755574372001129
Validation Accuracy: 0.6692068868190799
Validation Accuracy: 0.6522720858029918
```

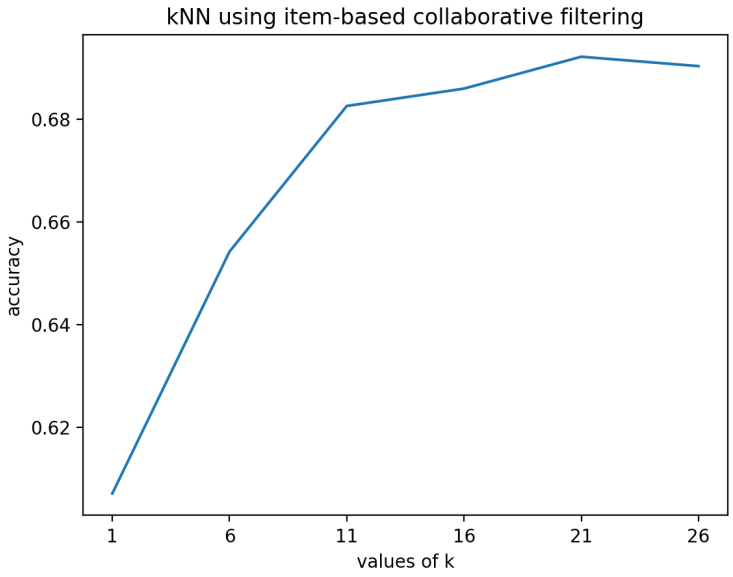
(b) Report the chosen k^* and the final test accuracy.

```
The final test accuracy is: 68.41659610499576% with k* = 11.
```

(c) Item-bases collaborative filtering.

The core underlying assumption is that if diagnostic question A has the same number of students answered correctly and the same number of students answered incorrectly as diagnostic question B, A's correctness on specific students matches that of question B.

Plot and report the accuracy on the validation data as a function of k.



For k = 1, 6, 11, 16, 21 and 26, the accuracy is:

```
Validation Accuracy: 0.607112616426757
Validation Accuracy: 0.6542478125882021
Validation Accuracy: 0.6826136042901496
Validation Accuracy: 0.6860005644933672
Validation Accuracy: 0.6922099915325995
Validation Accuracy: 0.69037538808919
```

The chosen k^* and the final test accuracy is:

```
The final test accuracy is: 68.16257408975444% with k* = 21.
```

(d) From part (b) and (c), we found that: using user-based collaborative filtering, the final test accuracy is around 68.417% with $k^* = 11$ while using item-based collaborative filtering, the final test accuracy is around 68.163% with $k^* = 21$, which is lower than that of user-based collaborative filtering. Therefore, user-based collaborative filtering performs better.

(e) Limitations of kNN.

- kNN uses more time to do computations.
  This means if the size of data is very large, then it will take a long time to predict. In particular, number of computations at training time is 0. Number of computations at test time per query is: calculate D-dimensional Euclidean distances with N data points takes $O(ND)$ and sort the distances takes $O(N \log N)$.
- kNN requires more memory to store the training data.
  Since our training data contains more than 50,000 training examples, we need a lot of space to store the whole training data set.

1.2 Item Response Theory. (Shujie Deng)

(a)

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

Denote the total number of students as N and the total number of questions as M. Then, the log-likelihood $\log p(\mathbf{C}|\theta, \beta)$ for all students and questions is:

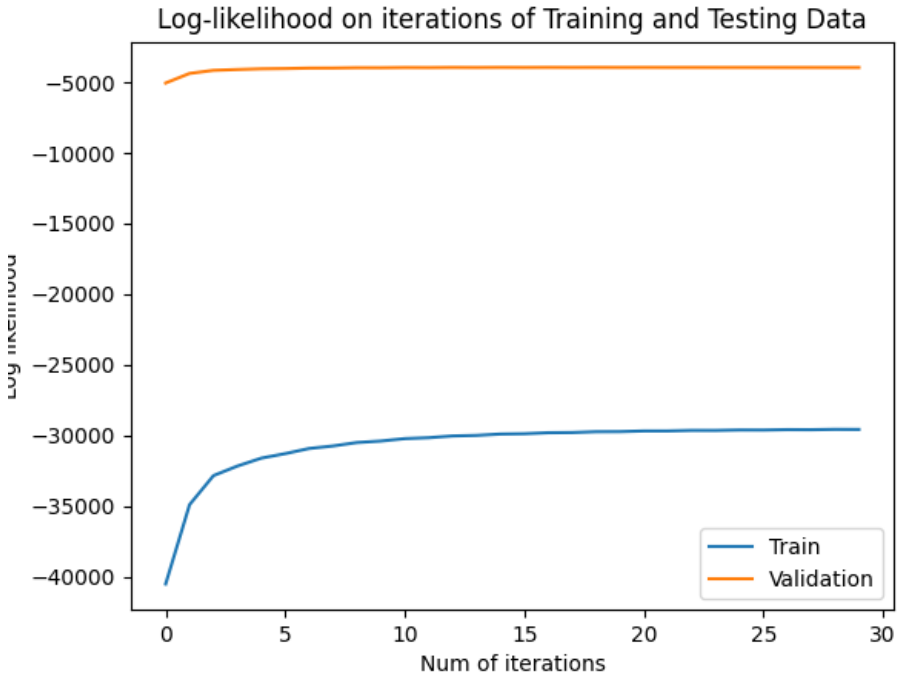
$$\begin{aligned}
l(\mathbf{C}|\theta, \beta) &= \prod_{i=1}^{N}\prod_{j=1}^{M}(\frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)})^{c_{ij}}(\frac{1}{1 + exp(a_j(\theta_i - \beta_j)})^{1-c_{ij}} \\
\log p(c|\theta, \beta) &= \sum_{i=1}^{N}\sum_{j=1}^{M} c_{ij}(\theta_i - \beta_j) - c_{ij}(1 + exp(\theta_i - \beta_j)) \\
&\quad - \log(1 + exp(\theta_i - \beta_j)) + c_{ij}(1 + exp(\theta_i - \beta_j)) \\
&= \sum_{i=1}^{N}\sum_{j=1}^{M} c_{ij}(\theta_i - \beta_j) - \log(1 + exp(\theta_i - \beta_j))
\end{aligned}$$

The derivative of the log-likelihood with respect to θ_i and β_j is given by:

$$\begin{aligned}
\frac{\partial}{\partial\theta_i} l(c|\theta, \beta) &= \sum_{j=1}^{M}(c_{ij} - \frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)}) \\
\frac{\partial}{\partial\beta_j} l(c|\theta, \beta) &= \sum_{i=1}^{N}(-c_{ij} + \frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)})
\end{aligned}$$

(b) The hyperparameters we chosen are:
Number of iterations is 30, and learning rate is 0.02.

The training curve that shows the training and validation log-likelihoods as a function of iterations is:

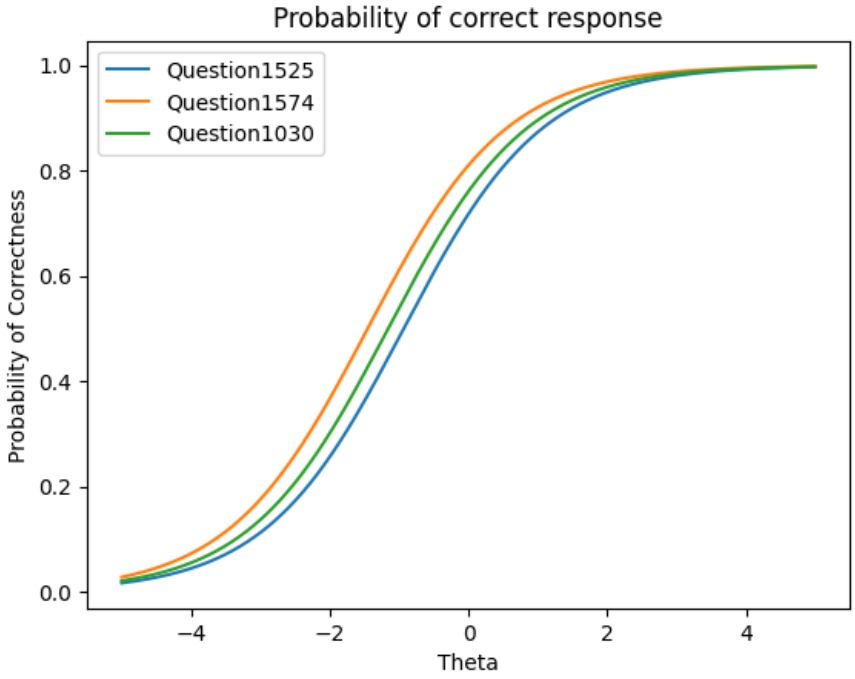


(c) The final validation accuracy is 0.7036409822184589
and the final test accuracy is 0.7075924357888794.

```
The learning rate is 0.02, and the number of iterations is 30.
The validation accuracy is 0.7036409822184589
Test accuracy is 0.7075924357888794.
```

(d) The shape of the graph is similar to the shape of the sigmoid function. Given a question, the function of probability of the correct response is a monotonic increasing function on theta, so as θ (the student ability) increases, the probability of correctness increases. Furthermore, for a fixed θ, the probability of answering Question 1525 correctly is greater than that of Question 1574 and that of Question 1030. This means the difficulty level of questions from highest to lower is 1030, 1574 and 1525.

Below is the plot of three curves that shows the probability of the correct response $p(c_{ij} = 1)$ as a function of θ given a question j.



1.3 Neural Networks. (Saifei Liao)

(a) Differences between ALS and neural networks.

- Neural networks usually require activation functions for layers, but this is not required for ALS.
- When we train the neural networks, we have the concept of layers, that is, the output may go through many layers of the neural networks before outputting anything.
- ALS does not have the concept of layers, the algorithm repeats until the given inputs (Z, U) converge.
- When we use ALS(inputs Z, U) to minimize the cost function for sparse matrix, we alternately optimize the inputs until convergence. However, this is not required in neural networks.

(b) See code.

(c) Epoch 20, k = 10, Learning rate = 0.1

```
Epoch: 19   Training Cost: 7592.822418   Valid Acc: 0.6782387806943269
```

Epoch 50, k = 10, Learning rate = 0.1

```
Epoch: 49   Training Cost: 5945.581810   Valid Acc: 0.6624329664126446
```

Epoch 20, k = 10, Learning rate = 0.001

```
Epoch: 19   Training Cost: 13612.363165  Valid Acc: 0.5637877504939317
```

Epoch 50, k = 10, Learning rate = 0.001

```
Epoch: 49   Training Cost: 12465.889899  Valid Acc: 0.6138865368331922
```

Epoch 20, k = 50, Learning rate = 0.1

```
Epoch: 19   Training Cost: 2254.301318   Valid Acc: 0.659046006209427
```

Epoch 50, k = 50, Learning rate = 0.1

```
Epoch: 49   Training Cost: 5988.966701   Valid Acc: 0.6591871295512278
```

Epoch 20, k = 50, Learning rate = 0.001

```
Epoch: 19   Training Cost: 12283.037250  Valid Acc: 0.6191081004798193
```

Epoch 50, k = 50, Learning rate = 0.001

```
Epoch: 49   Training Cost: 11850.464195  Valid Acc: 0.6289867344058707
```

Epoch 20, k = 50, Learning rate = 0.1

```
Epoch: 19   Training Cost: 1178.310977   Valid Acc: 0.6627152130962461
```

Epoch 50, k = 50, Learning rate = 0.1

```
Epoch: 49   Training Cost: 215.988466    Valid Acc: 0.6517075924357889
```

Epoch 20, k = 50, Learning rate = 0.001

```
Epoch: 19   Training Cost: 12154.461083  Valid Acc: 0.6275755009878634
```

Epoch 50, k = 50, Learning rate = 0.001

```
Epoch: 49   Training Cost: 11775.169668  Valid Acc: 0.6318092012418854
```

Epoch 20, k = 50, Learning rate = 0.1

```
Epoch: 19   Training Cost: 833.295393    Valid Acc: 0.6587637595258256
```

Epoch 50, k = 50, Learning rate = 0.1

```
Epoch: 49   Training Cost: 131.720330    Valid Acc: 0.6555179226644087
```

Epoch 20, k = 50, Learning rate = 0.001

```
Epoch: 19   Training Cost: 12138.695169  Valid Acc: 0.6291278577476714
```

Epoch 50, k = 50, Learning rate = 0.001

```
Epoch: 49   Training Cost: 11734.589647  Valid Acc: 0.6346316680779001
```

Epoch 20, k = 50, Learning rate = 0.1

```
Epoch: 19   Training Cost: 922.162483    Valid Acc: 0.6589048828676263
```

Epoch 50, k = 50, Learning rate = 0.1

```
Epoch: 49   Training Cost: 108.439978    Valid Acc: 0.6539655659046006
```

Epoch 20, k = 50, Learning rate = 0.001

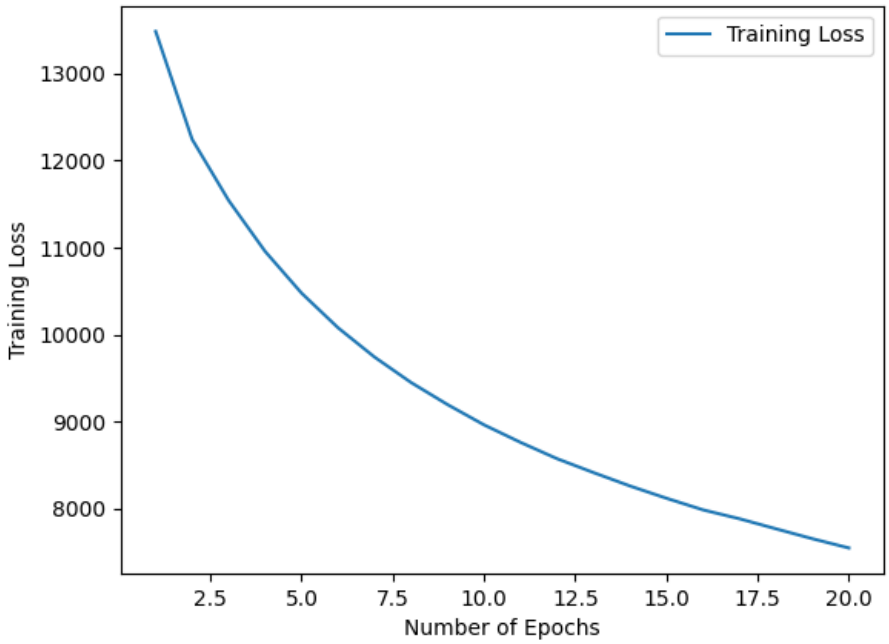
```
Epoch: 19   Training Cost: 12207.692214  Valid Acc: 0.6279988710132656
```

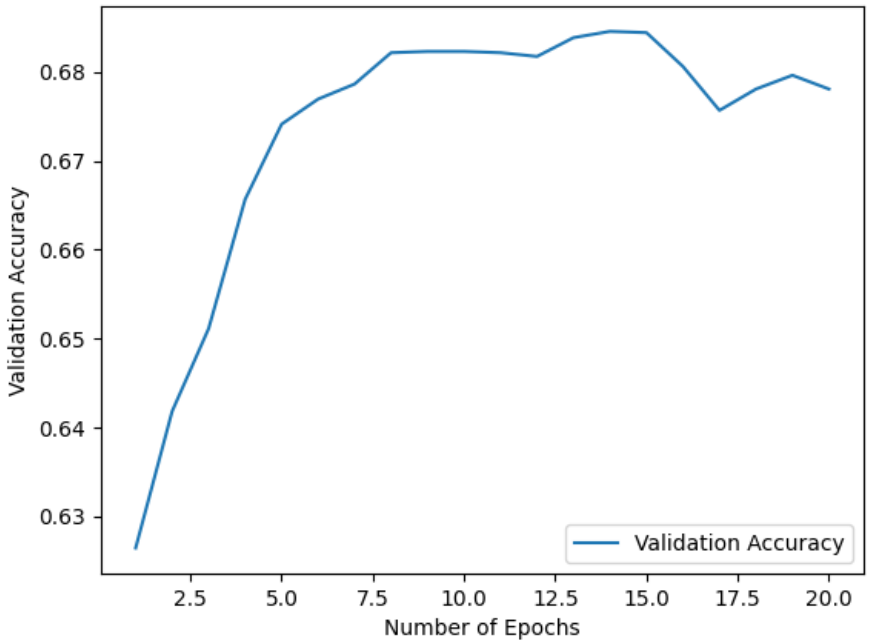
Epoch 50, k = 50, Learning rate = 0.001

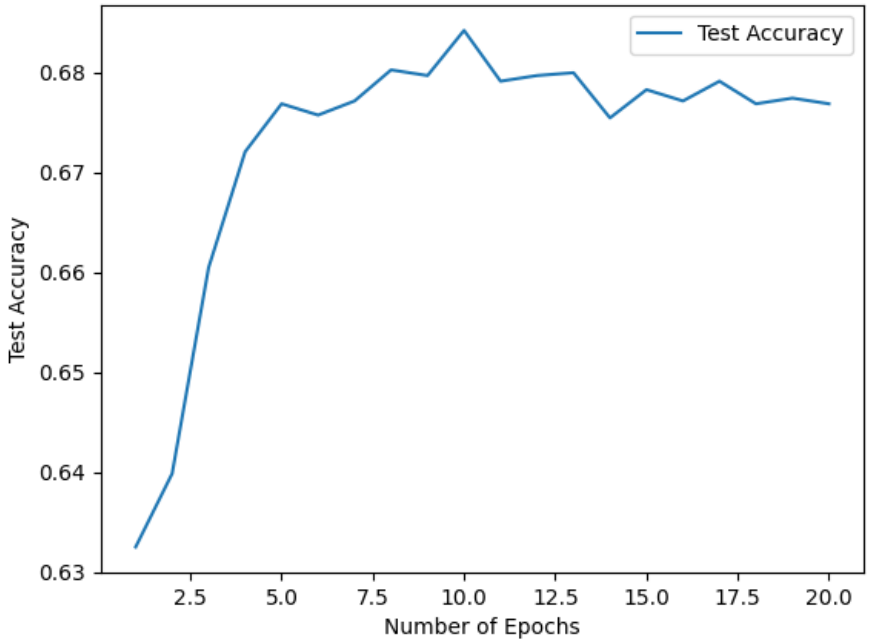
```
Epoch: 49   Training Cost: 11600.302036  Valid Acc: 0.6356195314705052
```

The k^* that has the highest validation accuracy is $k^* = 10$. At that time, we also have: epoch = 20 and learning rate = 0.1.

(d) Below are the plots showing how the training and validation objectives change as a function of epoch.







The final test accuracy is:

```
Epoch: 19   Training Cost: 7500.164066   Test Acc: 0.6768275472763196
```

(e) For k = 10, Epochs = 20, learning rate = 0.1:

(i) $\lambda = 0.001$

```
Epoch: 19   Training Cost: 9421.878088   Valid Acc: 0.6844482077335591
```

(ii) $\lambda = 0.01$

```
Epoch: 19   Training Cost: 12116.116841  Valid Acc: 0.6661021732994638
```

(iii) $\lambda = 0.1$

```
Epoch: 19   Training Cost: 12400.478931  Valid Acc: 0.6240474174428451
```

(iv) $\lambda = 1$

```
Epoch: 19   Training Cost: 12458.334489  Valid Acc: 0.6241885407846458
```

The λ that generates the best performance is 0.001.
The final validation accuracy is 0.684448207, and the final test accuracy is 0.67880327406.

Compared with the model without the regularization penalty, both of the validation and test accuracy are now a little bit higher by a very small amount (~ 0.001). Therefore, we can conclude that although there are some small improvements, the regularization term does not significantly improve the performance of our model.

1.4 Ensemble. (Lingfei Li)

In this problem, we use Item response theory model as our base model. In our bootstrapping process, we generate 3 new data sets of the same size as the training dataset by sampling from the training data, with replacement. After that, we apply the Item response theory model to these data sets and generate three predictions. The final predicted correctness is the averaged predicted correctness of the three predictions.

The final validation and test accuracy are:

```
Using bagging ensemble, the accuracy on validation data is:  0.7037821055602597
Using bagging ensemble, the accuracy on test data is:  0.7025119954840531
```

Compared to the validation and test accuracy in Part A Q2. There is only a minor improvement in validation accuracy (~ 0.0001) and no improvement in test accuracy using ensemble.
From the results of IRT in Q2, the validation accuracy is almost the same as that of the test accuracy. This indicates that the IRT model in Q2 does not overfit. The ensemble method is used to reduce the variance and reduce overfitting. However, since our base model does not overfit, the improvement of the performance of the ensemble may not be very significant.
In the meanwhile, we only resampled 3 times and generated 3 new datasets in our implementation, which is a relatively small number. Moreover, since we are sampling with replacement, our model may have seen some repeated data during the training procedure, which will not be very helpful to improve the accuracy. Therefore, if we resample our training data for large enough times, we may be able to see some improvements.

2 Part B (Shujie Deng, Lingfei Li and Saifei Liao)

2.1 Introduction

In part B, our group chose to modify Item Response Theory in part A aiming to predict students' answers to the diagnostic questions with higher accuracy. Comparing the performance of Item Response Theory on validation dataset and test dataset, we found there is little difference in terms of accuracy, which means our model has neither overfitting nor underfitting. However, since the final test accuracy is around 70%, we want to investigate whether we can do some improvements to increase the accuracy and we proposed several modifications. We extend our algorithm as follows.
(a) adding a discrimination parameter α_j for each question
(b) split the training data into different groups according to the difficulty level of questions (i.e. number of students answered correctly / number students answered this question) and then perform Item Response Theory on them respectively

2.2 Formal Description

(a) Discrimination Parameter α_j

In Part A, we are using the 1-Parameter Logistic Model to predict the probability for a correct answer for a student. That is, in the training step, we predict β—the difficulty level of the questions and θ—the students' ability to formulate the parameters for the probability distribution. In the test step, the parameters are used to predict how likely a student can answer a given question correctly. The 1-Parameter Logistic Model is formulated as:

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

However, in real life, it is often the case that students with higher ability will do better on difficult problems than students with low ability. That is, in order to generate better predictions, we need to consider adding a discrimination parameter that allows us to identify this difference. The idea comes from 2-Parameter Logistic Model[1]. The formulation is given by:

$$p(c_{ij} = 1|\theta_i, \beta_j, \alpha_j) = \frac{exp(\alpha_j(\theta_i - \beta_j))}{1 + exp(\alpha_j(\theta_i - \beta_j))}$$

Denote the total number of students as N and the total number of questions as M. The log-likelihood $\log p(C|\theta, \beta)$ and the derivative of the log-likelihood with respect to θ_i and β_j are given by:

$$
\begin{aligned}
l(c|\theta,\beta) &= \prod_{i=1}^{N}\prod_{j=1}^{M}(\frac{exp(\alpha_j(\theta_i-\beta_j)}{1+exp(\alpha_j(\theta_i-\beta_j))})^{c_{ij}}(\frac{1}{1+exp(\alpha_j(\theta_i-\beta_j)})^{1-c_{ij}} \\
\log p(c|\theta,\beta,\alpha) &= \sum_{i=1}^{N}\sum_{j=1}^{M} c_{ij}\alpha_j(\theta_i-\beta_j) - \log(1+exp(\alpha_j(\theta_i-\beta_j)) \\
\frac{\partial}{\partial\theta_i}l(c|\theta,\beta,\alpha) &= \sum_{j=1}^{M} c_{ij}\alpha_j - \frac{exp(\alpha_j(\theta_i-\beta_j))\alpha_j}{1+exp(\alpha_j(\theta_i-\beta_j))} \\
&= \sum_{j=1}^{M}\alpha_j(c_{ij} - \frac{exp(\alpha_j(\theta_i-\beta_j))}{1+exp(\alpha_j(\theta_i-\beta_j))}) \\
\frac{\partial}{\partial\beta_j}l(c|\theta,\beta,\alpha) &= \sum_{i=1}^{N} -c_{ij}\alpha_j + \frac{exp(\alpha_j(\theta_i-\beta_j))\alpha_j}{1+exp(\alpha_j(\theta_i-\beta_j))} \\
&= \alpha_j\sum_{i=1}^{N}(-c_{ij} - \frac{exp(\alpha_j(\theta_i-\beta_j))}{1+exp(\alpha_j(\theta_i-\beta_j))}) \\
\frac{\partial}{\partial\alpha_j}l(c|\theta,\beta,\alpha) &= \sum_{i=1}^{N} c_{ij}(\theta_i-\beta_j) + \frac{exp(\alpha_j(\theta_i-\beta_j))(\theta_i-\beta_j)}{1+exp(\alpha_j(\theta_i-\beta_j))}
\end{aligned}
$$

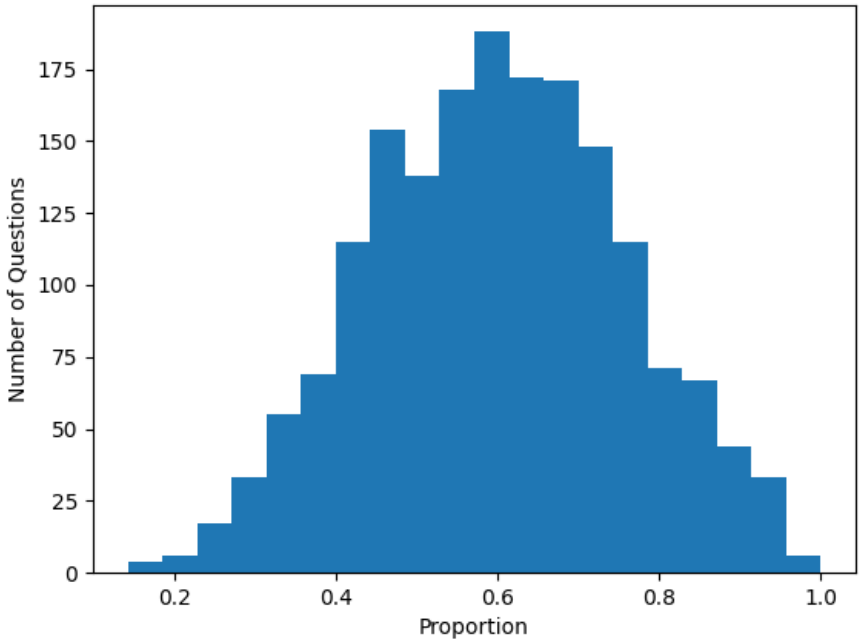
We are going to use similar implementation as Q2 to train our model.

(b) Split the training data by difficulty levels of questions

The original training data set contains more than 50,000 training examples and we treat all the questions the same. Therefore, we proposed that splitting the training data set and grouping it by question difficult levels may help us better figure out the pattern of the relationship between the questions and students' performances and thus better predict a student's answer given a specific question.

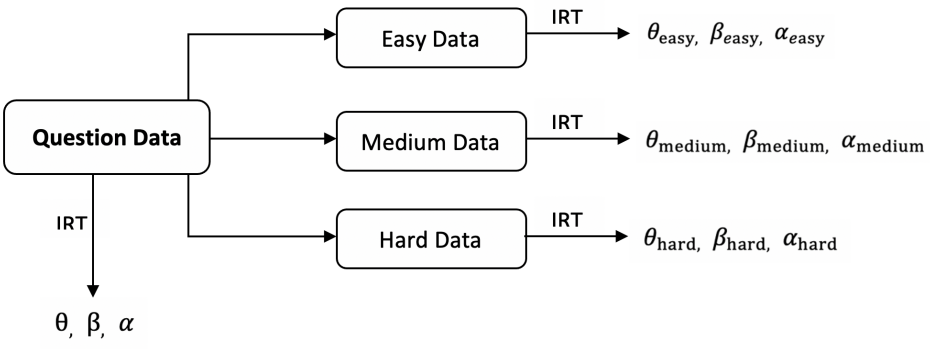
Given a specific question, we define proportion = the number of students answered correctly / the number of students who answered that question, and we observed that it follows a normal distribution, which means a huge number of problems will be categorized into medium level, and a relatively small number of questions will be categorized into easy or hard levels. This will lead to large differences between the sizes of training data, and our model will then be underfitting. So

we use hyperparameters m and n, which are 2 percentiles, to split the data and we can tune the hyperparameters to generate the best performance.
Below is the distribution of the proportion of correct answers:



2.3 Diagram

This is how our model works:



First, we find the best thresholds(m and n) to classify the difficulty levels of the training data set and divide it into three datasets(Easy Data, Medium Data and Hard Data). Then, we use IRT to

train these data sets and generate θ, β and α respectively. Meanwhile, we will also train a model with the discrimination parameter on the question data set using IRT. We also need to consider the possibility that a student i does not answer any questions in a certain difficulty level. To deal with this, if we find any entries in θ_{easy}, θ_{medium} or θ_{hard} that has not changed after the training of the model, we will replace these entries with θ_i, which is the student i's ability obtained by IRT model before splitting training data.
The following is the pseudo code of our algorithm to split data:

def SPLITDATA(data, m, n)

1 compute the proportion of correctness for each question using data
2 proportion = # of students answered correctly / # of students answered the question
3 compare proportion with m and n
4 easy questions = questions with proportion $< m$
5 medium questions = questions with proportion $< n$ and $\geq m$
6 hard questions = questions with proportion $\geq m$
7 **return** easy questions, medium questions and hard questions

Note that we will modify our IRT model in Part A Question 2 by adding the discrimination parameter α, ane we call the modified model IRT. The implementation of IRT is based on the formulas which we provide in the Formal Description part. Also, the EVALUATE function will first find the difficulty level that a question belongs to and then make a prediction from the data corresponding to that difficulty level. For simplicity of explanation, we will not include the exact code of IRT and EVALUATE here.

def DISCRIMINATIVE IRT(training_data, val_data, test_data, m, n)

1 θ, β, α = IRT(training_data)
2 easy, medium, hard = SPLITDATA(training_data, m, n)
3 $\theta_{easy}, \beta_{easy}, \alpha_{easy}$ = IRT(easy)
4 $\theta_{medium}, \beta_{medium}, \alpha_{medium}$ = IRT(medium)
5 $\theta_{hard}, \beta_{hard}, \alpha_{hard}$ = IRT(hard)
6 merge three levels of beta and alpha into beta_ split and alpha_split
7 create a hardness array indicating the difficulty levels of each question
8 update three levels of theta from θ
9 val_acc = EVAULATE(val_data, $\theta_{easy}, \theta_{medium}, \theta_{hard}$, beta_ split, alpha_split, hardness)
10 test_acc = EVAULATE(test_data, $\theta_{easy}, \theta_{medium}, \theta_{hard}$, beta_ split, alpha_split, hardness)
11 **return** val_acc and test_acc

2.4 Comparison

The parameter settings are:

Parameters	Base Model	Modified Model (Full data, Easy data, Medium data, Hard data)
# Of iterations	30	20, 40, 50, 20
Learning rate	0.02	0.01, 0.03, 0.01, 0.01
Lower threshold	/	0.53
Upper threshold	/	0.68

The accuracy rates are:

	Base Model	Modified Model (Full data, Easy data, Medium data, Hard data)
Final Validation Accuracy	0.70364	0.699407
Final Test Accuracy	0.70759	0.70336

(1) We add a discrimination parameter to our model. This parameter allows the model to better identify students' abilities and whether they can answer a question correctly. In real life, students with higher abilities can do better on more difficult problems, and adding this parameter takes this factor into consideration.
(2) It is likely that some questions are either too easy (almost all the students can answer them correctly) or too hard (few students can answer them correctly). This kind of questions isn't very representative of students' ability. Therefore, we consider making predication on correctness with different difficulty levels of questions. After calculating the proportion of correct answers divided by the total number of answers, we realize that overly easy or overly difficult questions do exist. Therefore, we split the training data set given difficulty levels of questions when training our model and we think this might increase the final accuracy.

Unfortunately, there is no significant improvement with our modified model. The final validation and test accuracy are almost the same as the base model.

2.5 Limitations

(1) In our implementation, we are splitting the original data set into three smaller pieces and making predictions respectively. This means each data set becomes smaller and when we train those

data sets, the results may therefore be underfitting.
A possible solution to this is to find more data and add them into the data set, or to apply the method of cross-validation, which is usually used for small data set. The k-th cross validation allows us to hold out the set 1 time and train the model $k - 1$ times. This method usually reduces the bias in the data set and results in better performance of the model.
(2) We didn't consider the case that students are guessing. It is very common that students just randomly guess an answer during the test, but this feature is not implemented in our code.
A possible solution is to use the 3-IRT model, which takes guessing into considerations. However, if we add more features into the model, it will increase the model complexity and result in overfitting. Therefore, we are doing a trade-off here.
(3) The training data matrix is sparse, and this could make our model overfitting though this does not happen in our implementation. A better solution is to add a prior distribution, which will control of the behavior of our model.

References

[1] "Item Response Theory."
https://www.publichealth.columbia.edu/research/population-health-methods/item-response-theory.