CSC311 Final Project Report

Due Date: Friday, Dec.3, at 11:59pm

Yingke Wang, Jun Xing, Tianyu Zhang

University of Toronto

December 2021

Contents

1 Part A

1.1 Question 1

(a) :

User based KNN Validation set Accuracy	
k	Validation Accuracy
1	0.6244
6	0.6781
11	0.6895
16	0.6756
21	0.6692
26	0.6523

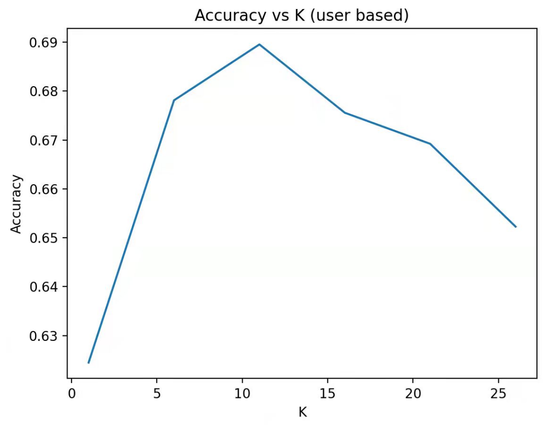


Figure 1: KNN user-base Validation Accuracy

(b) : Choose optimal k to be 11, the final test result is 0.6895

(c) :

Item based KNN Validation set Accuracy	
k	Validation Accuracy (%)
1	0.6071
6	0.6542
11	0.6826
16	0.6860
21	0.6922
26	0.6903

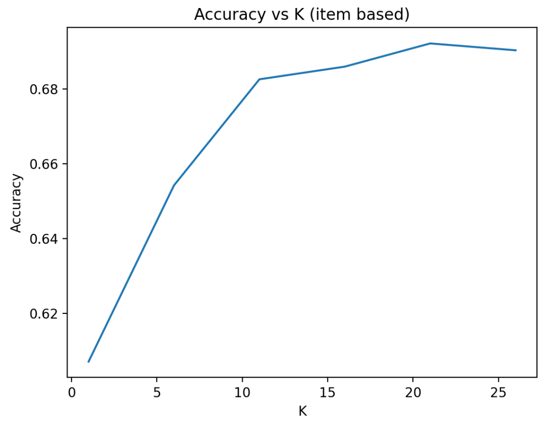
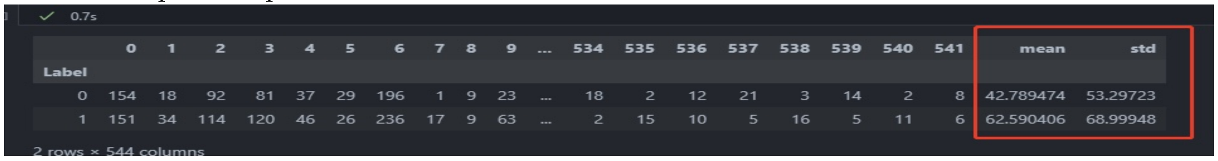


Figure 2: KNN item-base Validation Accuracy

For one question, which represented by a vector and each element in the vector represents the correctness of a student could score on the given question. KNN finds k closest questions, and the correctness of a student is predicted by averaging his/her correctness on those k closest questions. Therefore, the underlying assumption is that if a student could solve similar types of question, he/she would be able to solve this type of questions again.

(d) The final test accuracy for user-base and item-base are 0.68416 and 0.68162 respectively, there is a subtle difference and user-base performs better.

(e)
1. knn is not good at dealing with sparse train data. Since the matrix is sparse, only few data points knn can observe for each question and user, so it has limited observation to calculate accurate similarities among users and questions respectively.
2. Number of question per user answered:



0.7s

	0	1	2	3	4	5	6	7	8	9	...	534	535	536	537	538	539	540	541	mean	std
Label																					
0	154	18	92	81	37	29	196	1	9	23	...	18	2	12	21	3	14	2	8	42.789474	53.29723
1	151	34	114	120	46	26	236	17	9	63	...	2	15	10	5	16	5	11	6	62.590406	68.99948

2 rows × 544 columns

Based on our analysis above, we find out that sample data points varies a lot among the student. For example, student with id0 has 154 negative sample points and 151 positive sample points where student with id541 only has 8 and 6 respectively. While calculating close students, KNN would not be able properly consider that variance and positions of 1s.

1.2 Question 2

(a)

$$\begin{aligned}
\log p(\mathbf{C}|\boldsymbol{\theta},\boldsymbol{\beta}) &= \log \{\prod_{i=1}^{542}\prod_{j=1}^{1774} p(c_{ij}|\theta_i,\beta_j)\} && (1)\\
&= \sum_{i=1}^{542}\sum_{j=1}^{1774} \mathbb{I}[c_{ij} \neq \text{NaN}] \cdot \log p(c_{ij}|\theta_i,\beta_j) && (2)\\
&= \sum_{i=1}^{542}\sum_{j=1}^{1774} \mathbb{I}[c_{ij} \neq \text{NaN}] \cdot \log \{[p(c_{ij}=1|\theta_i,\beta_j)]^{c_{ij}}[1-p(c_{ij}=1|\theta_i,\beta_j)]^{1-c_{ij}}\} && (3)\\
&= \sum_{i=1}^{542}\sum_{j=1}^{1774} \mathbb{I}[c_{ij} \neq \text{NaN}] \cdot \log \{[\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}]^{c_{ij}}[\frac{1}{1+\exp(\theta_i-\beta_j)}]^{1-c_{ij}}\} && (4)\\
&= \sum_{i=1}^{542}\sum_{j=1}^{1774} \{\mathbb{I}[c_{ij} \neq \text{NaN}] \cdot c_{ij} \cdot \log \frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}\} && (5)\\
&+ \sum_{i=1}^{542}\sum_{j=1}^{1774} \{\mathbb{I}[c_{ij} \neq \text{NaN}] \cdot (1-c_{ij}) \cdot \log \frac{1}{1+\exp(\theta_i-\beta_j)}\} && (6)\\
&= \sum_{i=1}^{542}\sum_{j=1}^{1774} \{\mathbb{I}[c_{ij} \neq \text{NaN}] \cdot c_{ij} \cdot (\log \exp(\theta_i-\beta_j) - \log (1+\exp(\theta_i-\beta_j)))\} && (7)\\
&- \sum_{i=1}^{542}\sum_{j=1}^{1774} \{\mathbb{I}[c_{ij} \neq \text{NaN}] \cdot (1-c_{ij}) \cdot (\log (1+\exp(\theta_i-\beta_j))\} && (8)\\
&= \sum_{i=1}^{542}\sum_{j=1}^{1774} \{\mathbb{I}[c_{ij} \neq \text{NaN}] \cdot [c_{ij}\theta_i - c_{ij}\beta_j - \log (1+\exp(\theta_i-\beta_j))]\} && (9)
\end{aligned}$$

$$\frac{\partial[p(\boldsymbol{C}|\boldsymbol{\theta},\boldsymbol{\beta})]}{\partial\theta_i} = \sum_{j=1}^{1774}\{\mathbb{I}[c_{ij} \neq \text{NaN}] \cdot (c_{ij} - \frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)})\} \quad (10)$$

$$\frac{\partial[p(\boldsymbol{C}|\boldsymbol{\theta},\boldsymbol{\beta})]}{\partial\beta_j} = \sum_{i=1}^{542}\{\mathbb{I}[c_{ij} \neq \text{NaN}] \cdot (-c_{ij} + \frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)})\} \quad (11)$$

(b) First, we try with learning rates of 0.1, 0.01, 0.001, 0.0001 and 100 iterations. We modified the irt function so that it will return the iteration with maximum accuracy:

Hyperparameters and Accuracy		
Learning Rate	# of Iteration	Validation Accuracy (%)
0.1	2	69.1
0.01	8	70.9
0.001	77	70.8
0.0001	96	69.4

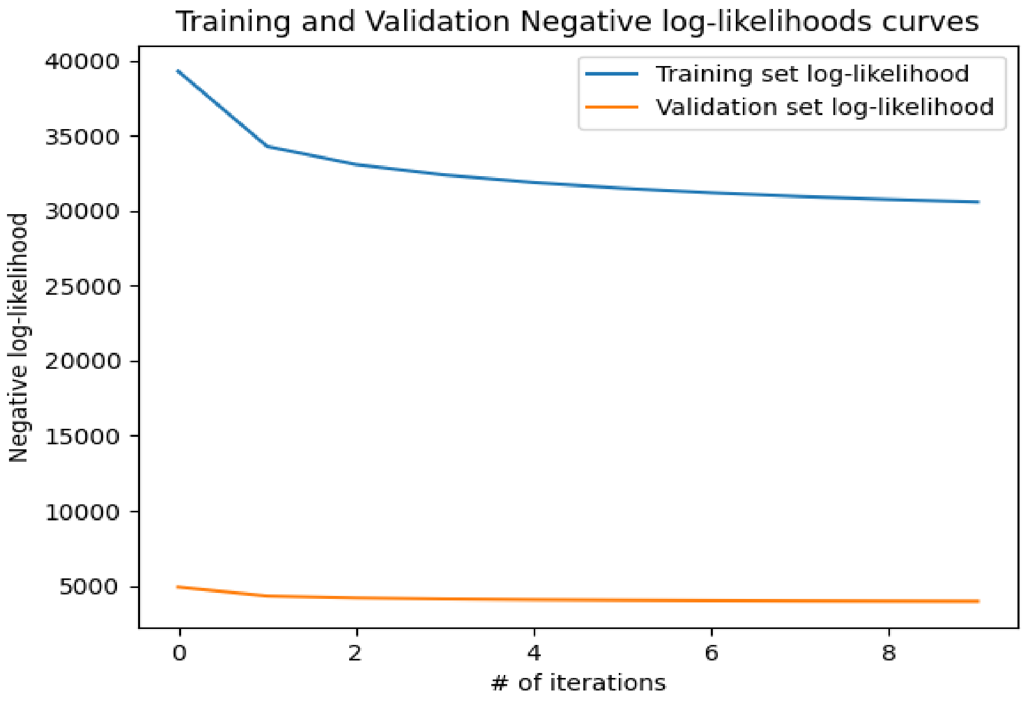
The hyperparameters with highest validation accuracy are: learning rate = 0.01 and # of iterations = 8.

We found that when the learning rate is 0.1, it is too large, and the accuracy result oscillates a lot. For learning rate of 0.001 and 0.0001 that are too small, and the progress is too slow. A learning rate around 0.01 is a better choice. Then we train the models again with new learning rates closer to 0.1:

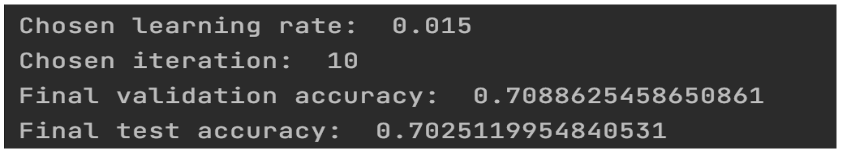
Hyperparameters and Accuracy		
Learning Rate	# of Iteration	Validation Accuracy (%)
0.01	8	70.9
0.011	13	70.8
0.013	10	70.8
0.015	10	70.9

We choose the hyperparameters with highest validation accuracy: learning rate = 0.015 and # of iterations = 10.

The following is the plot of the training curve that shows the training and validation negative log-likelihood as a function of iteration:

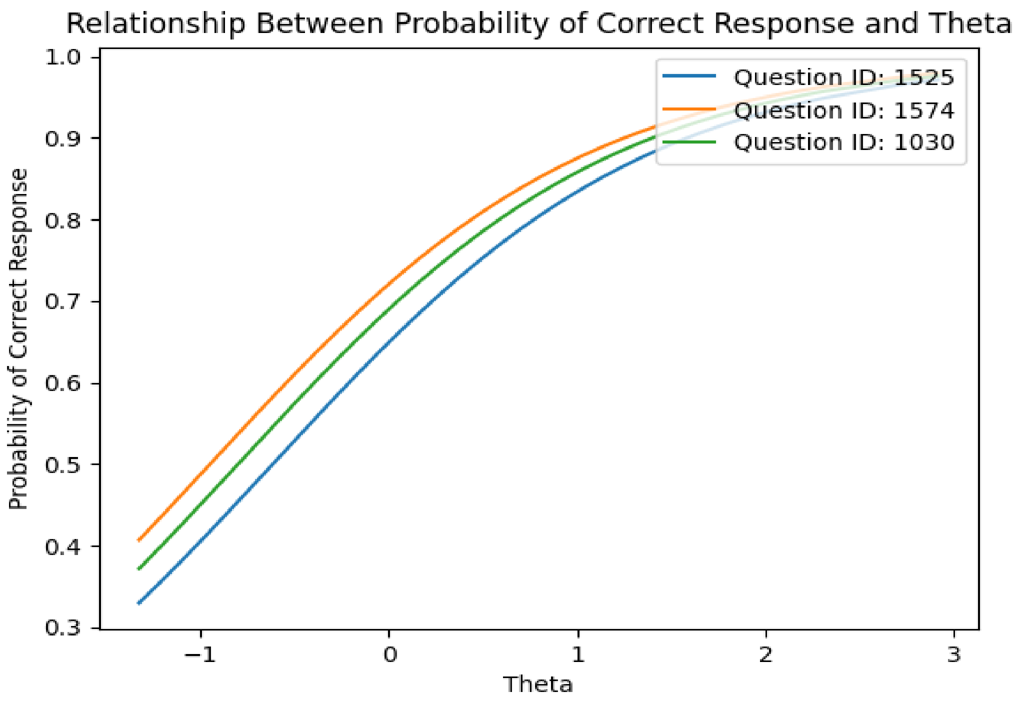


(c) Report on final validation and test accuracies:



```
Chosen learning rate:  0.015
Chosen iteration:  10
Final validation accuracy:  0.7088625458650861
Final test accuracy:  0.7025119954840531
```

(d) The three questions we chose are the first three questions in the training set, *i.e.*, questions with id 1525, 1574, and 1030. The following is the plot of probabilities of the correct response $p(c_{ij} = 1)$ as a function of θ given the question j:



Given the IRT model formula:

$$p(c_{ij} = 1|\theta_i, \beta_j) = \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)} \tag{12}$$

when we are fixing β_j values for specific question j, the expected curve should be in sigmoid shape (but translated in the x-axis direction, given the probability of correct response is a function of θ). Since the iteration number we chose is relatively small (iteration # = 10), the values of theta and beta are not updated much. Therefore, the curves of those three chosen questions are in the shape of the sigmoid function, but the shape does not contain the complete curve of a typical sigmoid function. Nevertheless, from the graph we observe that as theta increases, the probability of getting a correct answer for each question increases, meaning that the students with stronger abilities are more likely to answer a question correctly. Moreover, we observe that the probability of correct answer for question with id 1574 is higher than the probability of correct answer for question with id 1030, which is higher than the probability of correct answer for question with id 1525, *i.e.*:

$$p(c_{i1574} = 1|\theta_i^*, \beta_{1574}) \geq p(c_{i1030} = 1|\theta_i^*, \beta_{1030}) \geq p(c_{i1525} = 1|\theta_i^*, \beta_{1525}), \tag{13}$$

where θ^* denotes the final updated θ for the hyperparameters we chose. This infers that the difficulty β of the questions has the relationship:

$$\beta_{1574} \leq \beta_{1030} \leq \beta_{1525} \tag{14}$$

1.3 Question 3

Implement **Neural Networks** to predict students' correctness on a diagnostic question.

(a) Describe at least three differences between ALS and neural networks.

Answer:

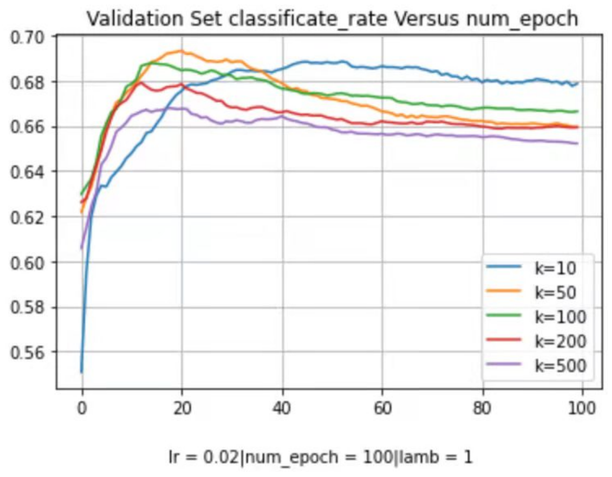
- Alternating Least Squares (ALS) is an optimization algorithm that used in Probability Matrix Factorization(PMF) model/Latent Factor Model which is a common collaborative filtering approach for building recommendation systems. On the other hand, Neural Networks(NN) are much broader computational technique which is commonly used in computer vision and natural language processing.
- During optimization, ALS holds student matrix fixed and runs gradient descent with question matrix, and then holds question matrix fixed and runs gradient descent with student matrix. So the two matrices are optimized alternatively. In contrast, NN use backpropagation to update the whole weight matrix at the same time.
- For PMF with ALS approach, student-question scoring matrix mxn where we have m students and n questions is decomposed into two matrices Umxk and Vkxn where k represents hidden feature dimensions. And we can tune k as one of the hyperparameter. In NN, we can use multiple hidden layers and nonlinear activation functions to obtain the features to be used for prediction.

(b) Implement a class AutoEncoder that performs a forward pass of the autoencoder following the instructions in the docstring.

Answer: See code in neural_network.py

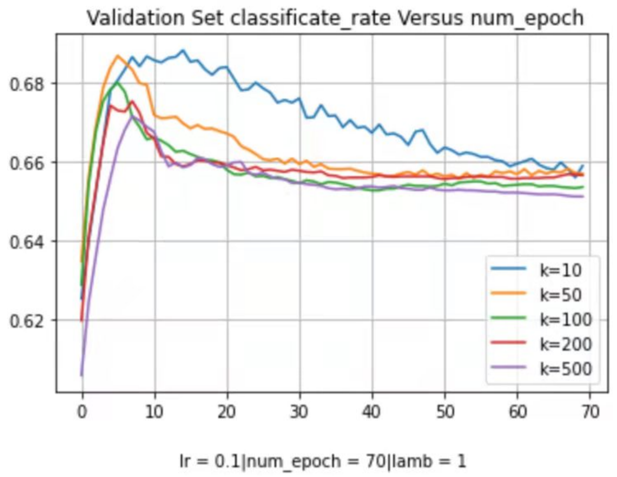
(c) Train the autoencoder using latent dimensions of $k \in \{10, 50, 100, 200, 500\}$. Also, tune optimization hyperparameters such as learning rate and number of iterations. Select kthat has the highest validation accuracy.

Answer:
The baseline NN model uses one hidden layer model, and this question is mainly about selecting proper number of nodes (k) in the hidden layer. If k is too small, the model is more likely to underfit the training data. And if k is too large, the model is more likely to go overfitting. Generally, there are practical rules to follow to determine the number of nodes, but we will be conducting experiments and select k which yields the highest accuracy on given validation dataset. Some experiments results are shown below:



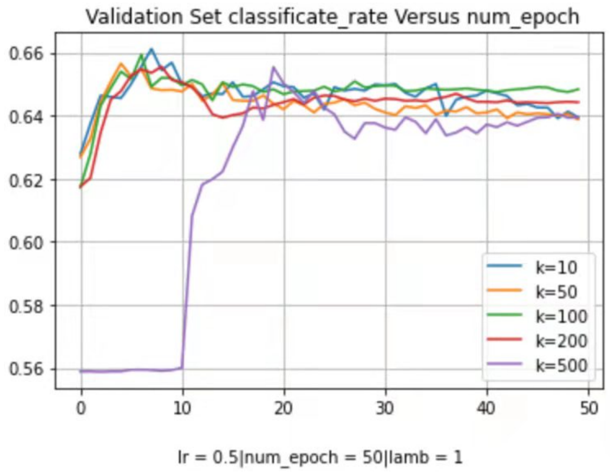
	Highest validation accuracy during training	Accuracy at end of training
k=10	0.6885	0.6787
k=50	0.6933	0.6595
k=100	0.6878	0.6664
k=200	0.6791	0.6594
k=500	0.6678	0.6521

Figure 3: Experiment #1: lr = 0.02, num_epoch = 100, lamb = 1



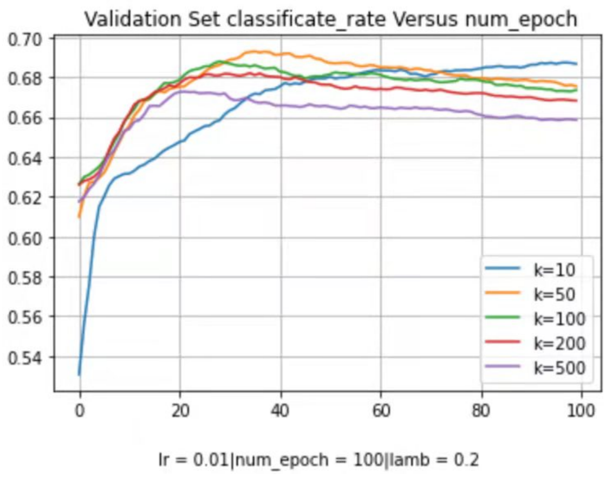
	Highest validation accuracy during training	Accuracy at end of training
k=10	0.6881	0.6589
k=50	0.6867	0.6569
k=100	0.6799	0.6535
k=200	0.6753	0.6565
k=500	0.6715	0.6511

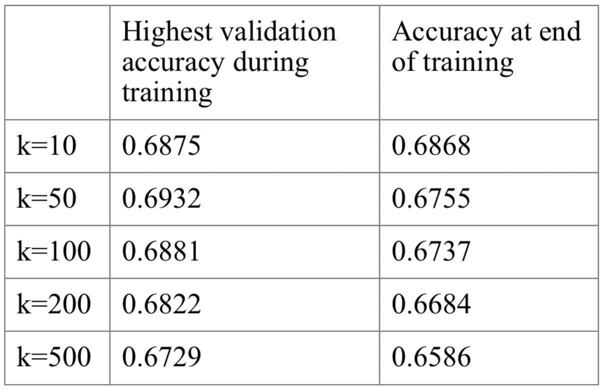
Figure 4: Experiment #2: lr = 0.1, num_epoch = 70, lamb = 1



	Highest validation accuracy during training	Accuracy at end of training
k=10	0.6612	0.6394
k=50	0.6565	0.6387
k=100	0.6593	0.6483
k=200	0.6555	0.6442
k=500	0.6553	0.6394

Figure 5: Experiment #3: lr = 0.5, num_epoch = 50, lamb = 1





	Highest validation accuracy during training	Accuracy at end of training
k=10	0.6875	0.6868
k=50	0.6932	0.6755
k=100	0.6881	0.6737
k=200	0.6822	0.6684
k=500	0.6729	0.6586

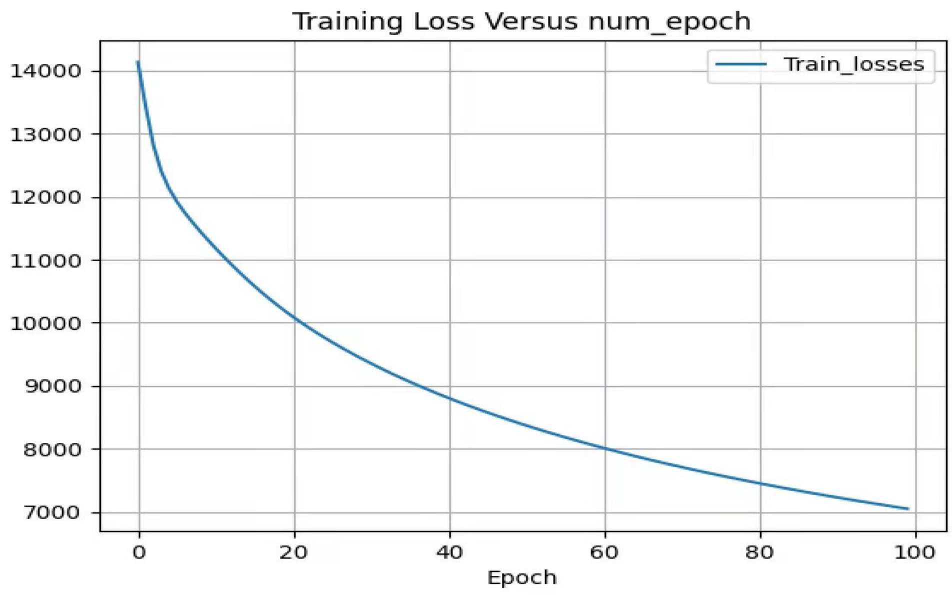
Figure 6: Experiment #4: lr = 0.01, num_epoch = 100, lamb = 0.2

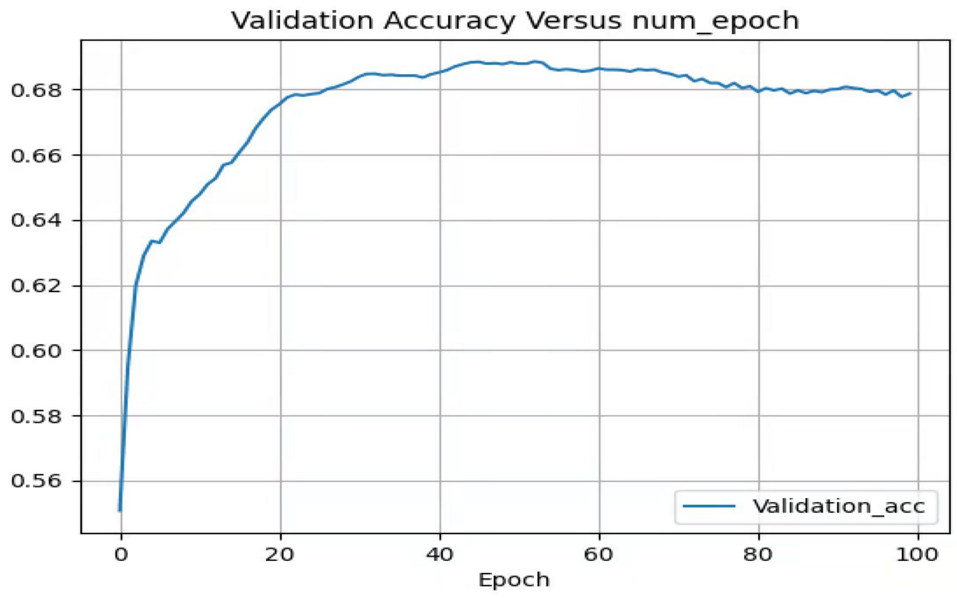
By experimenting on varies combination of k, lr, num_epoch and lamb, the final selection of hyper-parameters of baseline NN model which achieved highest validation accuracy are:

- k = 10
- lr = 0.02
- num_epoch = 100
- lamb = 0.1

(d) With your chosen k, plot and report how the training and validation objectives changes as a function of epoch. Also, report the final test accuracy.

Answer:





final_test_accuracy: 0.6872

Notice that if we use early stopping strategy and stops training at 50th epoch, we could have slightly higher test accuracy of 0.6878.

(e) Modify a function train so that the objective adds the L2 regularization. You may use a method get_weight_norm to obtain the regularization term. Using the k and other hyperparameters selected from part (d), tune the regularization penalty $\lambda \in \{0.001, 0.01, 0.1, 1\}$. With your chosen λ, report the final validation and test accuracy. Does your model perform better with the regularization penalty?

Answer:
By setting k = 10, lr = 0.02, num_epoch=100 and we can tune the regularization penalty λ:

Experiments	final_validation_accuracy	final_test_accuracy
λ =0.001	0.6877	0.6898
λ =0.01	0.6699	0.6616
λ =0.1	0.6248	0.6235
λ =1	0.6249	0.6237

By comparing to the final accuracy of model without regularization, it seems that model with L2 regularization and $\lambda = 0.001$ performs slightly better. But models with larger λ performs worse.

1.4 Question 4

We experimented bagging with three different experiments, each with the following combinations of base models:

Experiment 1 (bagging with three KNN models):

- KNN user based model with k = 11
- KNN item based model with k = 21
- KNN item based model with k = 26

Experiment 2 (bagging with three IRT models):

- IRT model with learning rate 0.01 and iteration 8
- IRT model with learning rate 0.015 and iteration 10
- IRT model with learning rate 0.011 and iteration 13

Experiment 3 (bagging with two KNN models and one IRT model):

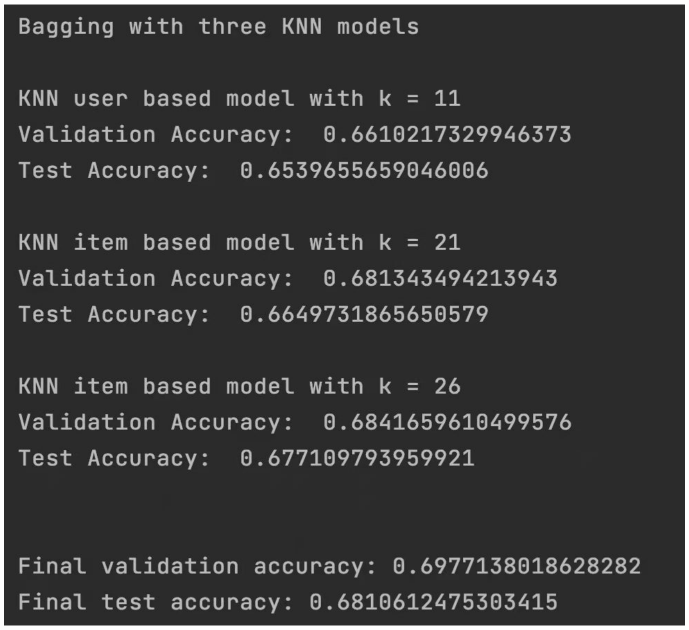
- KNN user based model with k = 11
- KNN item based model with k = 21
- IRT model with learning rate 0.015 and iteration 10

For each experiment, we generated 3 new data sets from the data in train_data.csv, which is the original data set $\mathcal{D}$. Each sample in the new data sets is randomly chosen from $\mathcal{D}$ with replacement. Then we used bootstrap aggregation to combine the three base models by taking the arithmetic mean of the predictions from the three base models and comparing that with the threshold, *i.e.*,

$$y_{bagged} = \mathbb{I}(z_{bagged} > 0.5) = \mathbb{I}(\sum_{i=1}^{3} \frac{z_i}{m} > 0.5) \tag{15}$$

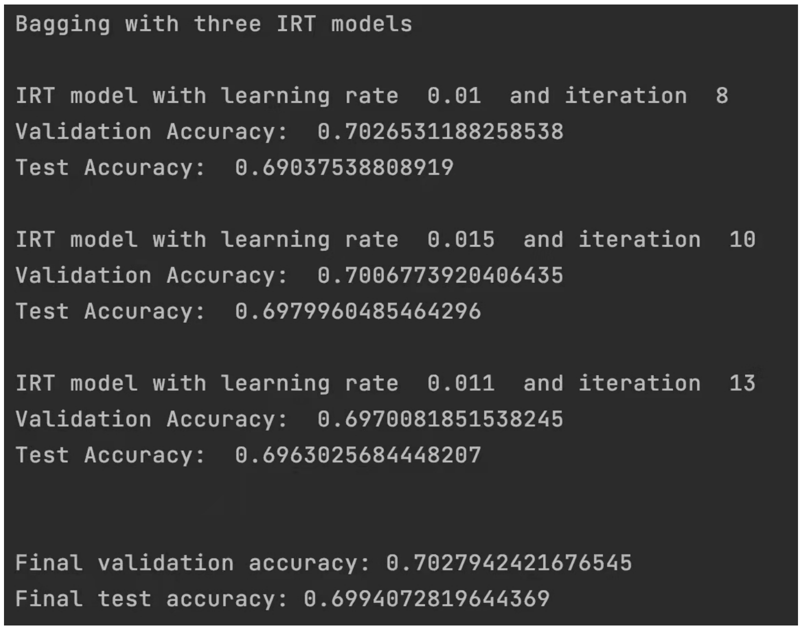
where y_{bagged} denotes the final prediction, z_{bagged} denotes the real-valued probability output of the bagged classifier, and z_i denotes the real-valued probability outputs from the three base models.

The final validation accuracy and test accuracy for the three experiments are:



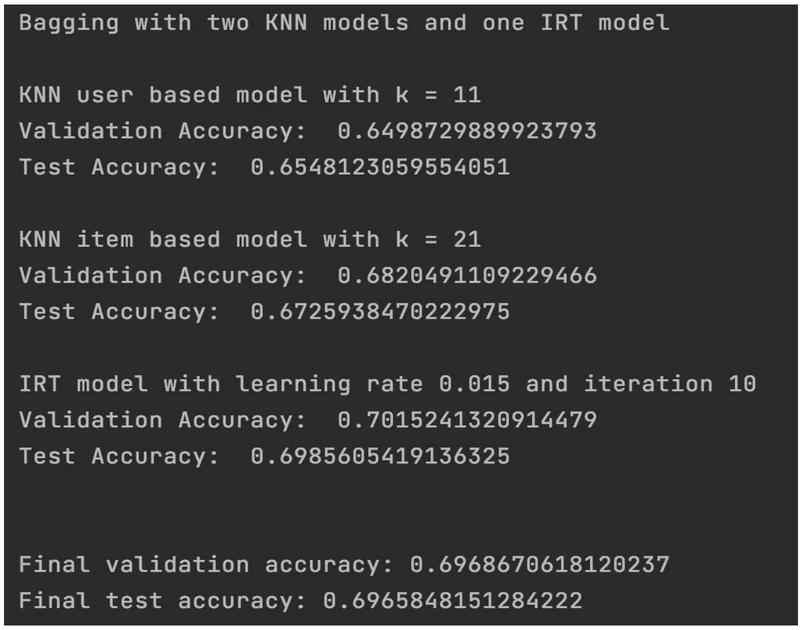
```
Bagging with three KNN models

KNN user based model with k = 11
Validation Accuracy:  0.6610217329946373
Test Accuracy:  0.6539655659046006

KNN item based model with k = 21
Validation Accuracy:  0.681343494213943
Test Accuracy:  0.6649731865650579

KNN item based model with k = 26
Validation Accuracy:  0.6841659610499576
Test Accuracy:  0.677109793959921



Final validation accuracy: 0.6977138018628282
Final test accuracy: 0.6810612475303415
```

Figure 7: Experiment #1: bagging with three KNN models



```
Bagging with three IRT models

IRT model with learning rate  0.01  and iteration  8
Validation Accuracy:  0.7026531188258538
Test Accuracy:  0.69037538808919

IRT model with learning rate  0.015  and iteration  10
Validation Accuracy:  0.7006773920406435
Test Accuracy:  0.6979960485464296

IRT model with learning rate  0.011  and iteration  13
Validation Accuracy:  0.6970081851538245
Test Accuracy:  0.6963025684448207



Final validation accuracy: 0.7027942421676545
Final test accuracy: 0.6994072819644369
```

Figure 8: Experiment #2: bagging with three IRT models



```
Bagging with two KNN models and one IRT model

KNN user based model with k = 11
Validation Accuracy:  0.6498729889923793
Test Accuracy:  0.6548123059554051

KNN item based model with k = 21
Validation Accuracy:  0.6820491109229466
Test Accuracy:  0.6725938470222975

IRT model with learning rate 0.015 and iteration 10
Validation Accuracy:  0.7015241320914479
Test Accuracy:  0.6985605419136325


Final validation accuracy: 0.6968670618120237
Final test accuracy: 0.6965848151284222
```

Figure 9: Experiment #3: bagging with two KNN models and one IRT model

By comparing the results from different experiments, we finally choose bagging with three IRT models, which has validation accuracy of 70.3% and test accuracy of 69.9%.

We observed that the final validation accuracy and test accuracy after bagging don't change much, compared with those accuracies for the three base models. This is because bagging won't change the bias, but it can reduce the variance. In conclusion, our ensemble outperforms single baseline models because it reduces overfitting and gives us a more generalized model.

2 Part B

2.1 Highlights of Improvements:

1. The final accuracy our selected model achieved on test set is **70.9%**.
2. Major improvements we implemented include the following:
   - Improve baseline NN by augmenting dimension of input vector and alternating NN structure
   - Improve baseline IRT by adding two more parameters (discrimination α_j and guessing index g_j of questions) in addition to θ and β in the original one-parameter IRT model (1PL) and initialize θ according to different age groups, so that we have better description for each group of students.

2.2 Exploratory Data Analysis:

Before we started to make any improvement, we decided to take better look at the data so we have a better idea why certain baseline models perform poorly and how to improve them. Here are some the findings:

1. *Quantify sparsity of training matrix*

   Once we load the training matrix from given "train_sparse.npz" file, we could calculate the sparsity of the matrix by summing up the NA positions and divided by the size of the matrix.

$$\text{sparsity} = \frac{\text{len(trainmatrix\_np[np.isnan(trainmatrix\_np)])}}{\text{(numStudents*numQuestions)}} \tag{16}$$

   And we will get sparsity equals to 0.9410 and we can conclude that approximately only 6% of interactions between a question and a student are available for learning.

2. *Distribution of postive and negative training data*

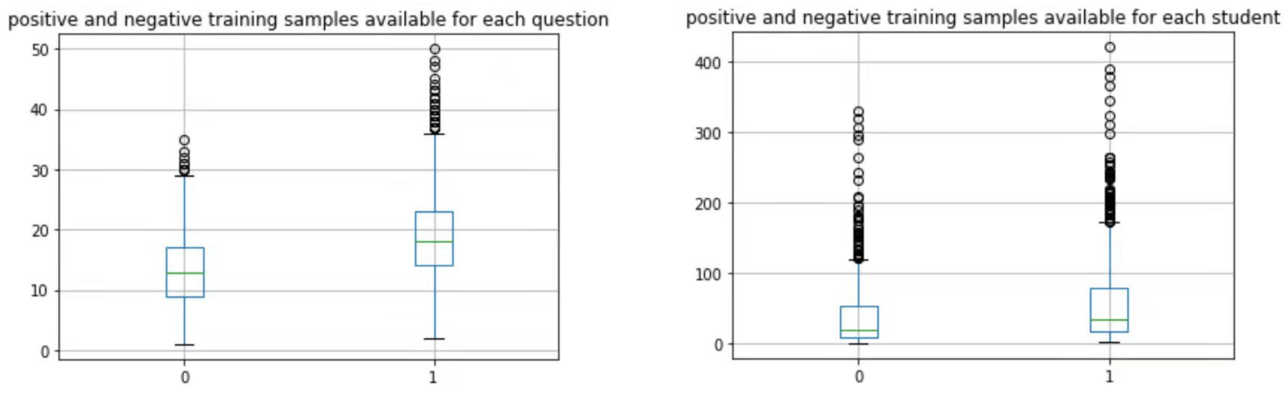
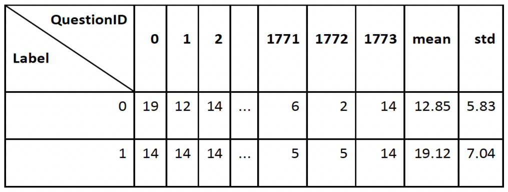
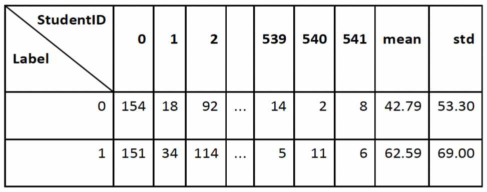


Figure 10: Box Charts of Distribution of Positive and Negative Training Data



QuestionID / Label	0	1	2		1771	1772	1773	mean	std
0	19	12	14	...	6	2	14	12.85	5.83
1	14	14	14	...	5	5	14	19.12	7.04



StudentID / Label	0	1	2		539	540	541	mean	std
0	154	18	92	...	14	2	8	42.79	53.30
1	151	34	114	...	5	11	6	62.59	69.00

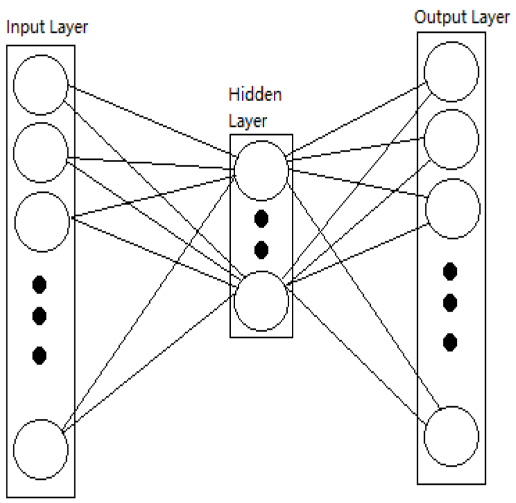
Figure 11: Tables of Distribution of Positive and Negative Training Data

The above plots and table were constructed by summing up the frequency of 0s and 1s in the sparse training matrix horizontally and vertically. And based on these results, we can conclude that:

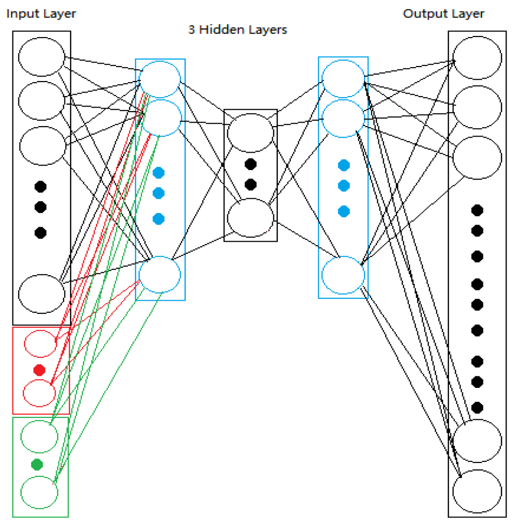
- On average, each student vector has 43 negative training data and 63 positive data which is much more than that of a question vector. It provides explanation why KNN_user model performs better than KNN_item model.
- The number of 0s and 1s varies a lot among students. For example, student with ID 0 has 154 zeros and 151 ones, but student with ID 541 only has 8 zeros and 6 ones. Similarly, large variation also occurs for question vectors. This provides evidence that KNN may not be suitable for given context.

Based on findings from Exploratory Data Analysis, we decided to not to make any improvement on baseline kNN model and focus on the improvements of the other two models.

2.3 Major Improvement for Baseline NN:



(a) Baseline NN



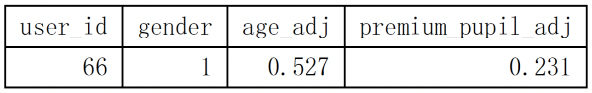
(b) Improved NN

Figure 12: From Baseline NN to Improved NN

As shown above, the baseline NN model consists with 3 layers and the input layer has 1774 nodes where each node contains a binary correctness value of a student scored on a question [Notice that for clarity, not all nodes are shown].

We know that the input layer takes in a student vector where each element could be interpreted as his/her probability to get correct on each question and NA positions in this vector are predicted by encoding and decoding reconstruction of the student vector while optimizing the overall loss function. So, the more informative the student vector is, better performance of the NN model. Therefore, we started to test our hypothesis by extending the original vector with two new vectors as shown in red and green in the graph above.

"Red" vector is a student vector with 3 elements, where the elements represent gender, age, premium_pupil respective. An example is shown on the right, and notice that age was normalized by max age for feature scaling. And missing values were replaced by mean of the column.



user_id	gender	age_adj	premium_pupil_adj
66	1	0.527	0.231

Figure 13: Example of Student Feature Vector

"Green" vector is a student vector with 388 elements, where each element represents a student ability to score on one type of subject. There are total of 388 subjects, and each question belongs to one or more types of subjects. Algorithm we used to calculate student ability to score on subject is listed below and code of its implementation is available in refact_subject_precent.py.

1) All positions of student-subject start with 5/10, so denominator is 10 and numerator is 5. (We also tested with other start value such as 50/100)

2) Iterate all questions for a given student, every time he/she scores 1 on a question, find corresponding subjects and add 1 to numerator, and for every question that the correctness isn't NA, we add 1 to denominator.

In order for NN to learn features better, we added two additional hidden layers and tuned hyperparameters such as the number of nodes in the hidden layer and number of epochs. The improved_NN model code is available in NN_improved.py and results of improved NN model and discussion are available in the comparison section of this report.

2.4 Major Improvement for Baseline IRT:

2.4.1 Description

Our baseline model is the **one-parameter IRT model (1PL)**, which only has one single parameter β_j, the question difficulty, to describe each question j. For this part, we used the **three-parameter model (3PL)** to optimize the performance of our baseline model. In the three-parameter model, the probability of a question j to be correctly answered by a student i is:

$$p(c_{ij} = 1|\theta_i, \beta_j, \alpha_j, g_j) = g_j + \frac{1 - g_j}{1 + e^{-\alpha_j(\theta_i - \beta_i)}} = g_j + \frac{(1 - g_j)\ \exp[\alpha_j(\theta_i - \beta_j)]}{1 + \exp[\alpha_j(\theta_i - \beta_j)]} \tag{17}$$

Parameters	
Parameter	Interpretation
θ_i	i-th student's ability
β_j	j-th question's difficulty
α_j	j-th question's discrimination
g_j	j-th question's guessing index

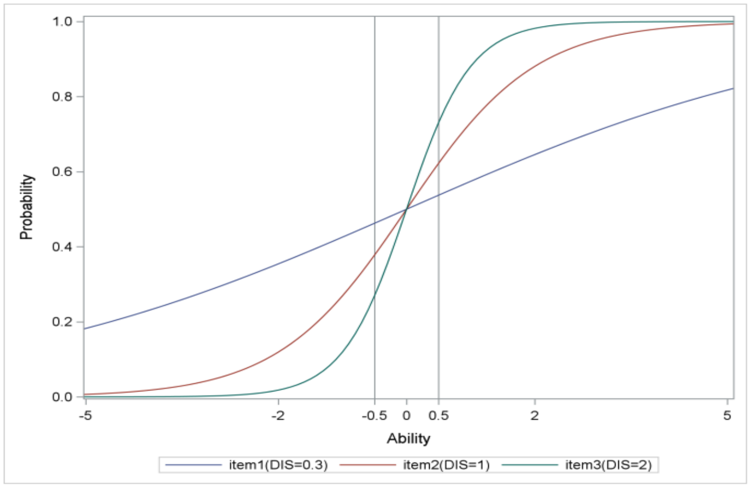


Figure 14: Theta Explanation [1]

The discrimination of a question denotes the slope of the inflection point of the logistic function. The discrimination parameter is a measure of the differential capability of an question. A high discrimination parameter value suggests a question that has a high ability to differentiate students. In practice, a high discrimination parameter value means that the probability of a correct answer of question increases more rapidly as the ability of a student increases. Logistic function curves of three questions, question1, question2, and question3, with different discrimination parameter values are shown in **Figure 14**. The question's difficulty is set to 0. The discrimination parameter values are 0.3, 1, and 2, respectively. We can observe that as the discrimination parameter value increases, the slope becomes more steep around 0. As the ability value changes from –0.5 to 0.5, the probability of a correct response changes from 0.3 to 0.7 for question3, which is much larger than question1. Therefore the higher the discrimination, the better the question can differentiate students' ability.

The guessing index denotes the probability that a student with low ability may answer the question correctly be chance. Since the diagnostic questions are multiple choice question, there are two major cases when a student with low ability may get the correct answer by chance:

1. the students has no idea how to solve the problem but get the correct answer by randomly select an answer (this is mainly influenced by the number of options for the multiple choice question);
2. the question is not so well-designed so that the student can figure out the answer by excluding those definitely wrong options even though they do not know the answer.

Take the example diagnostic question on the handout for example (shown in **Figure 15**). The question is asking for a obtuse angle, but answers B and D are not obtuse angles, so they definitely cannot be the correct answer. Hence, it has a higher probability to give the correct answer to this question even though the student may not know how to solve it. The design of the problem should be considered as a trait of the problem itself.

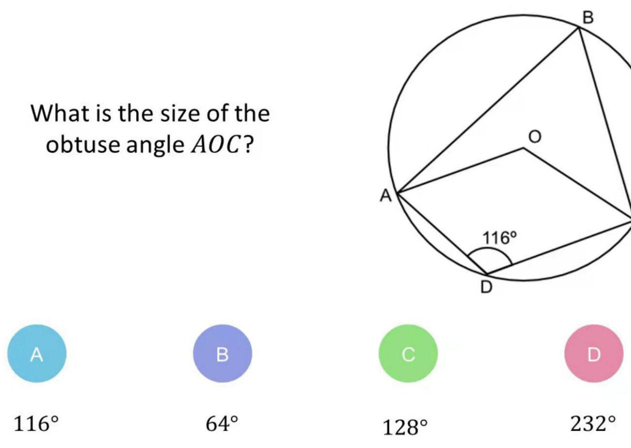


Figure 15: Example diagnostic question [2]

In the baseline IRT model, only β is used to describe the traits of each questions. The underlying assumption is that all the questions have the same discrimination and guessing can be regarded as legitimate abilities of the student to solve the question. Those two additional parameters give better description of the traits of each questions, and therefore enable the optimized model to carry out better performance.

We derived the log-likelihood and partial derivatives with respect to the parameters we are going to update using gradient ascent:

$$\log\, p(\mathbf{C}|\boldsymbol{\theta},\boldsymbol{\beta},\boldsymbol{\alpha},\boldsymbol{g}) = \sum_{i=1}^{542}\sum_{j=1}^{1774}\{\mathbb{I}[c_{ij}\neq \mathrm{NaN}]\cdot[(1-c_{ij})\cdot\log(1-g_j)+c_{ij}\cdot\log[g_j+\exp[\alpha_j(\theta_i-\beta_j)]]-\log[1+\exp[\alpha_j(\theta_i-\beta_j)]]]\} \tag{18}$$

$$\frac{\partial[p(\boldsymbol{C}|\boldsymbol{\theta},\boldsymbol{\beta},\boldsymbol{\alpha},\boldsymbol{g})]}{\partial\theta_i} = \sum_{j=1}^{1774}\{\mathbb{I}[c_{ij}\neq \mathrm{NaN}]\cdot(c_{ij}\cdot\alpha_j\cdot\frac{\exp[\alpha_j(\theta_i-\beta_j)]}{g_j+\exp[\alpha_j(\theta_i-\beta_j)]}-\alpha_j\cdot\frac{\exp[\alpha_j(\theta_i-\beta_j)]}{1+\exp[\alpha_j(\theta_i-\beta_j)]})\} \tag{19}$$

$$\frac{\partial[p(\boldsymbol{C}|\boldsymbol{\theta},\boldsymbol{\beta},\boldsymbol{\alpha},\boldsymbol{g})]}{\partial\beta_j} = \sum_{i=1}^{542}\{\mathbb{I}[c_{ij}\neq \mathrm{NaN}]\cdot(-c_{ij}\cdot\alpha_j\cdot\frac{\exp[\alpha_j(\theta_i-\beta_j)]}{g_j+\exp[\alpha_j(\theta_i-\beta_j)]}+\alpha_j\cdot\frac{\exp[\alpha_j(\theta_i-\beta_j)]}{1+\exp[\alpha_j(\theta_i-\beta_j)]})\} \tag{20}$$

$$\frac{\partial[p(\boldsymbol{C}|\boldsymbol{\theta},\boldsymbol{\beta},\boldsymbol{\alpha},\boldsymbol{g})]}{\partial\alpha_j} = \sum_{i=1}^{542}\{\mathbb{I}[c_{ij}\neq \mathrm{NaN}]\cdot(c_{ij}(\theta_i-\beta_j)\frac{\exp[\alpha_j(\theta_i-\beta_j)]}{g_j+\exp[\alpha_j(\theta_i-\beta_j)]}-(\theta_i-\beta_j)\cdot\frac{\exp[\alpha_j(\theta_i-\beta_j)]}{1+\exp[\alpha_j(\theta_i-\beta_j)]})\} \tag{21}$$

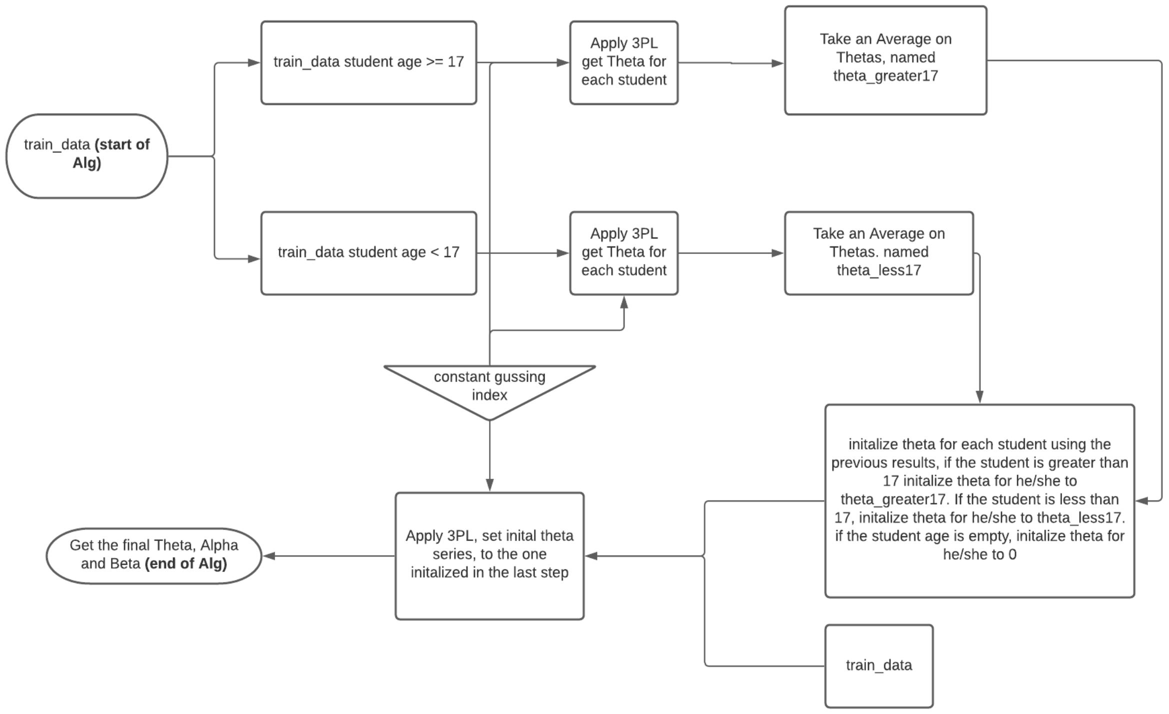
$$\frac{\partial[p(\boldsymbol{C}|\boldsymbol{\theta},\boldsymbol{\beta},\boldsymbol{\alpha},\boldsymbol{g})]}{\partial \alpha_j} = \sum_{i=1}^{542}\{\mathbb{I}[c_{ij} \neq \text{NaN}] \cdot [(c_{ij}-1)\frac{1}{1-g_j} + c_{ij} \cdot \frac{1}{g_j + \exp[\alpha_j(\theta_i - \beta_j)]}]\} \tag{22}$$

We updated theta, beta, and alpha during gradient ascent by the following algorithm (implementation is in opt_irt.py):

Algorithm 1 Updating theta, beta, and alpha

```
$\theta_i \leftarrow \theta_i + \text{lr} * \frac{\partial[p(\boldsymbol{C}|\boldsymbol{\theta},\boldsymbol{\beta},\boldsymbol{\alpha},\boldsymbol{g})]}{\partial\theta_i}$
$\beta_j \leftarrow \beta_j + \text{lr} * \frac{\partial[p(\boldsymbol{C}|\boldsymbol{\theta},\boldsymbol{\beta},\boldsymbol{\alpha},\boldsymbol{g})]}{\partial\beta_j}$
$\alpha_j \leftarrow \alpha_j + \text{lr} * \frac{\partial[p(\boldsymbol{C}|\boldsymbol{\theta},\boldsymbol{\beta},\boldsymbol{\alpha},\boldsymbol{g})]}{\partial\alpha_j}$        ▷ We did gradient ascent as we are maximizing log-likelihood
```

After improving the baseline **one-parameter IRT model (1PL)** to **three-parameter model(3PL)**, it is hypothesised that separating groups may help improve parameter θ_i and we attempted to implement it (as shown in **Figure 16**). We considered two groups: one contains all training examples whose student age is less than 17; the other one contains those are greater or equation than 17. For instance, an adult student could be more likely to answer more questions correctly than a student just enter elementary school, since it spends more years on education and more likely to have more knowledge. Therefore, training theta independently on them can capture the group of people's distribution more accurately. Also notice that age only affect the student's ability but not neither α_j nor β_j. Hence, we will still train α_j and β_j and $\theta_i$on the entire training set. Just set initial value of θ to the one we get from the separate age training in two groups. Therefore, the algorithm's flow go as follow in **Figure 16**.



train_data (start of Alg)
train_data student age >= 17
train_data student age < 17
Apply 3PL get Theta for each student
Apply 3PL get Theta for each student
Take an Average on Thetas, named theta_greater17
Take an Average on Thetas. named theta_less17
constant gussing index
initalize theta for each student using the previous results, if the student is greater than 17 initalize theta for he/she to theta_greater17. If the student is less than 17, initalize theta for he/she to theta_less17. if the student age is empty, initalize theta for he/she to 0
Apply 3PL, set inital theta series, to the one initalized in the last step
train_data
Get the final Theta, Alpha and Beta (end of Alg)

Figure 16: Grouping IRT flowchart

2.5 Comparison and Demonstration

2.5.1 Models required by project

Comparison of Accuracies		
Model	Train Accuracy (%)	Test Accuracy (%)
KNN_item	Not assessed	68.1
KNN_user	Not assessed	68.4
Ensemble with 3 Baseline IRT	72.6	69.9
Baseline NN	80.7	68.7
Improved NN	**75.3**	**70.2**
Baseline IRT	73.3	70.3
Optimized IRT without grouping	74.1	70.8
Optimized IRT with grouping	**74.1**	**70.9**

1. **KNN Models Versus Other models**
   Since the baseline KNN model uses KNN imputer package from sklearn, positions of student-question that are not NAN are not predicted, we didn't assess the training accuracy for those baseline KNN models. Evidence of KNN models are not suitable for the given context can be found in **Section 2.2**.

2. **Ensemble with 3 Baseline IRT Versus Baseline IRT**
   See explanations in **Section 1.4** in Part A.

3. **Baseline_NN Versus Improved_NN**
   As shown in the table, the training accuracy from baseline NN model reduced from 80.7% to 75.3% in the improved NN model. At the same time, the test accuracy from baseline NN model increased from 68.7% to 70.2%. This provides enough evidence that the improved NN model less overfits to the training data and has better optimization. We believe the improvements are due to the fact that we added reasonable features to the input vector in the first layer and better hidden features are learned by the added hidden layers.

4. **Baseline_IRT Versus Improved_IRT**
   From the above table, our optimized IRT model with grouping increased the test accuracy for 0.6% and the training accuracy also increased slightly, which is consistent with our hypothesis(can be found in **Section 2.4.1**).

2.5.2 Pick final model for Engineering/Deployment

In conclusion, for the purpose of engineering/deployment, we would pick improved_IRT as our final model because it achieved highest accuracy on test data set while it does not overfit training set based on its moderated training accuracy.

2.6 Limitations and Extension

2.6.1 General limitations that applies to all existing models

- If a student that in test set have not done any of the question or a question that have not been worked by any of the student, which leads to all NAs horizontally or vertically, none of the models could predict well for such pair of student-question.

- As we calculated in Exploratory Data Analysis, only 6% of interactions between a question and a student are available for learning. If we could get more training data, better performance of the models is expected.

2.6.2 Limitations and extensions applies to our selected model (IRT)

- **Better choice of g:** According to **Formula 17** on page 16, the parameter g_j, which represents the probability that a student with low ability may guess correctly on question j, has strong influence on the final prediction, yet there aren't any additional data available in the question metadata to update g. Therefore, our current algorithm is to pass a hyperparameter g value as a constant, meaning that all the questions will have the same guessing index. Ideally, the guessing index should be used as a trait to describe each question. To improve, one feasible solution is to update g using gradient ascent like the other in model parameters.
- **IRT Assumption:** In our selected model, there is an underlying assumption which assumes sample independence and is very likely to be violated in practice. For instance, if the correctness of a student's score depends on that of another student, then our IRT model would not be able to predict accurately.

2.6.3 Next Step

One of the extensions we attempted is to covert the entire problem into a binary classification problem. As shown in the graph below, we tried to construct a new training matrix only based on the features of a student and features of a question.

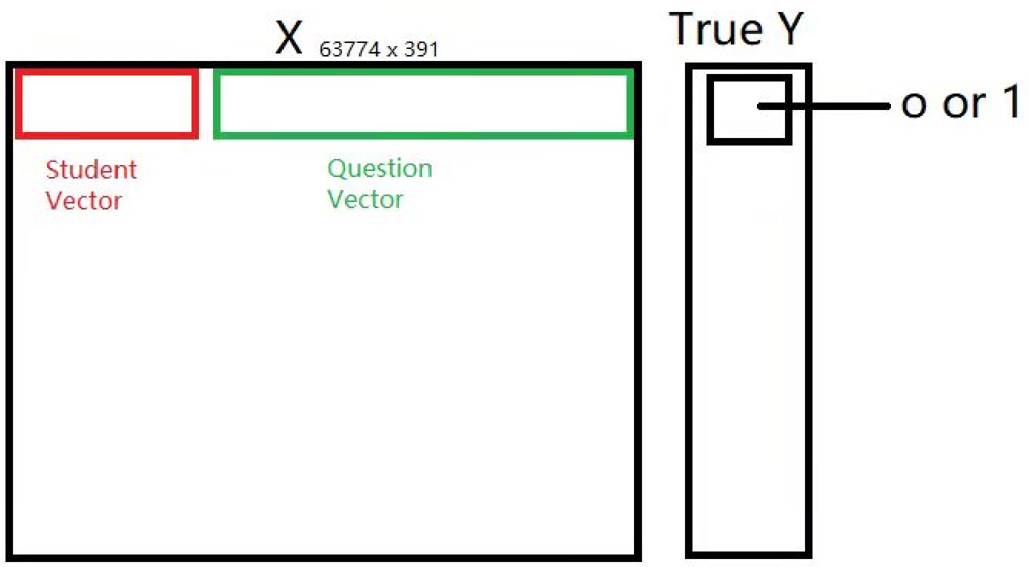


Figure 17: New Training Matrix

The red vector is a student vector with 3 elements which are gender, age and premium_pupil.

The green vector is a question vector with 388 elements where each element is a binary value and 1 only shows up at the index of the subject type this question belongs to.

Combines the observation from “train_data.csv” and “valid_data.csv”, we have total of 63774 observations of pairs of student and question and its corresponding binary correctness.

The benefit of this approach is that we could minimize the effect of NAs in the training data and the underly assumption is that there existing relationship between raw features of student-question pair and probability that the student could get correct on this question. We tested our hypothesis with following three models but none of them performs better than the ones we tried in Part A.

Model	Train Accuracy	Test Accuracy
Decision Tree	65.7%	59.18%
Logistic Regression	60.3%	60.17%
Support Vector Machine	60.4%	60.11%

If time permitted, we would investigate more on those models and explore the possibility to ensemble them with our models in part A.

References

[1] An, Xinming, Yung, Yiu-Fai. *Item Response Theory: What It Is and How You Can Use the IRT Procedure to Apply It.* SAS support. (n.d.).

[2] Wang, Zichao, et al. *Diagnostic Questions: The NeurIPS 2020 Education Challenge.* arXiv preprint arXiv:2007.12061(2020)