CSC311 Fall 2022 Project Final Report

Ruiting Chen, Sirui Chen, Xinran Gong

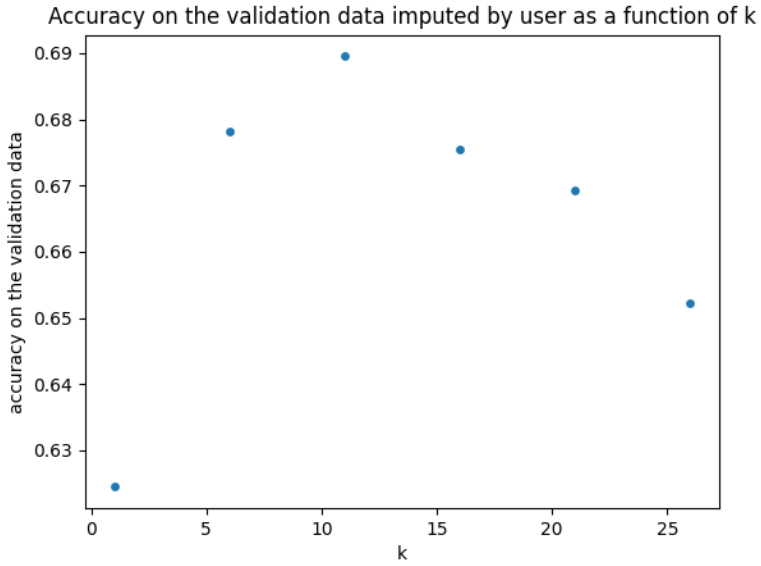
Contribution:

Ruiting Chen: Q1-code, Q2-code, Q2-report, partB-code + report: comparison/demonstration
Sirui Chen: Q1-code, Q1-report, Q4-code, Q4-report, partB-report: description + figure
Xinran Gong: Q3-code, Q3-report, partB-report: limitations

Q1 k-Nearest Neighbor

(a) The accuracy on validation data for $k \in \{1, 6, 11, 16, 21, 26\}$ is as the following:

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



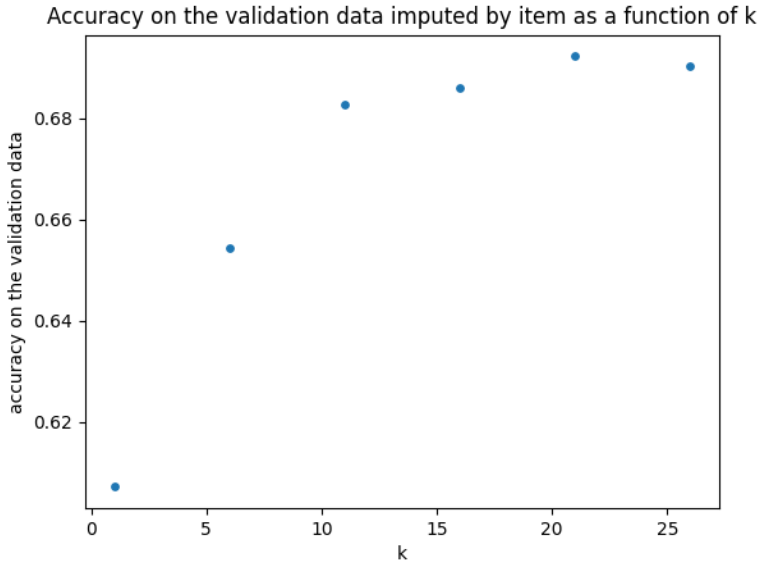
(b) We choose k*=11 for *knn_imputed_by_user* and the final test accuracy is as the following:

```
final test accuracy with k=11
By User Validation Accuracy: 0.6841659610499576
```

(c) For item-based collaborative filtering, the underlying assumption is that if a diagnostic question A has the same correct and incorrect answers from other users as question B, A's correctness from a

specific user matches that of question B.
The accuracy on validation data for $k \in \{1, 6, 11, 16, 21, 26\}$ is as the following:

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



We choose k*=21 for *knn_imputed_by_item* and the final test accuracy is as following:

```
final test accuracy with k=21
By Item Validation Accuracy: 0.6816257408975445
```

(d) The test performance between user- and item- collaborative filtering is quite similar with user-based performs slightly better.
(e) Here are two potential limitation of kNN in this task:

- Computational Cost: For every missing value in the sparse matrix, we need to calculate distance for all other data points, sort the distances and use the nearest k neighbors to impute its value. It cost a lot of computation time and also memory space to do this work, especially when k is large.
- Curse of Dimensionality: When dimension is high, we will have information about a bunch of users and questions. Then when we use user-based collaborative filtering, there will be a lot of users that similarly answered other questions. For item-based collaborative filtering, we will have a lot of questions that has been answered similarly by other users. This will cause the algorithm to choose users/questions that are not the best to impute the missing value.

Q2 Item Response Theory

(a) Let n = total number of students. Let m = total number of questions. The log-likelihood is derived as following under the Naive Bayes assumption:

$$\begin{aligned}
p(C|\theta,\beta) &= p(c_{11}, c_{12}, \ldots, c_{nm}|\theta,\beta) \\
&= \prod_{i=1}^{n}\prod_{j=1}^{m} p(c_{ij}=1|\theta_i,\beta_j)^{c_{ij}} \cdot (1-p(c_{ij}=1|\theta_i,\beta_j))^{(1-c_{ij})} \\
&= \prod_{i=1}^{n}\prod_{j=1}^{m} \sigma(\theta_i-\beta_j)^{c_{ij}} \cdot (1-\sigma(\theta_i-\beta_j))^{(1-c_{ij})}
\end{aligned}$$

$$\begin{aligned}
log(p(C|\theta,\beta)) &= log\left(\prod_{i=1}^{n}\prod_{j=1}^{m} \sigma(\theta_i-\beta_j)^{c_{ij}} \cdot (1-\sigma(\theta_i-\beta_j))^{(1-c_{ij})}\right) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij} \cdot log(\sigma(\theta_i-\beta_j)) + (1-c_{ij}) \cdot log(1-\sigma(\theta_i-\beta_j))
\end{aligned}$$

Now we calculate the partial derivative of the log-likelihood with respect to each θ_i and β_j

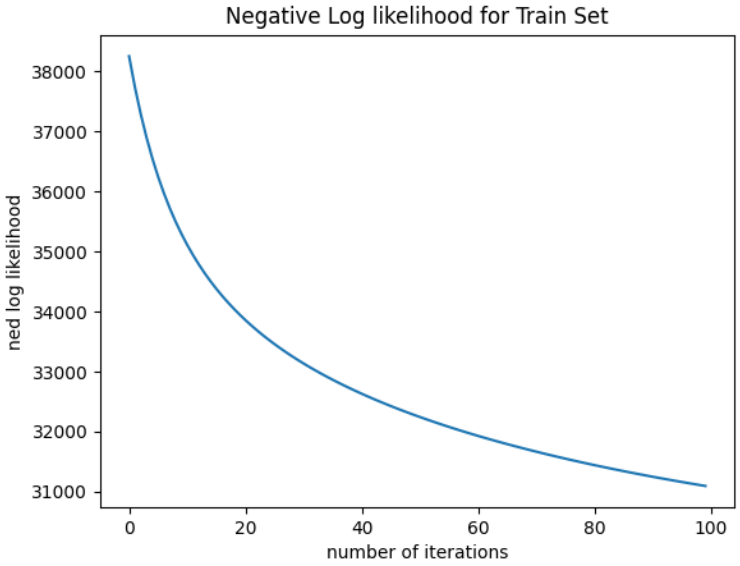
$$\begin{aligned}
\frac{\partial log(p(C|\theta,\beta))}{\partial \theta_i} &= \sum_{j=1}^{m} \frac{c_{ij}}{\sigma(\theta_i-\beta_j)} \cdot \frac{\partial \sigma(\theta_i-\beta_j)}{\partial(\theta_i-\beta_j)} \cdot \frac{\partial(\theta_i-\beta_j)}{\partial\theta_i} \\
&\quad + \frac{1-c_{ij}}{1-\sigma(\theta_i-\beta_j)} \cdot \frac{\partial(1-\sigma(\theta_i-\beta_j))}{\partial(\theta_i-\beta_j)} \cdot \frac{\partial(\theta_i-\beta_j)}{\partial\theta_i} \\
&= \sum_{j=1}^{m} \frac{c_{ij}}{\sigma(\theta_i-\beta_j)} \cdot \sigma(\theta_i-\beta_j) \cdot (1-\sigma(\theta_i-\beta_j)) \cdot 1 \\
&\quad + \frac{1-c_{ij}}{1-\sigma(\theta_i-\beta_j)} \cdot -\sigma(\theta_i-\beta_j) \cdot (1-\sigma(\theta_i-\beta_j)) \cdot 1 \\
&\qquad\qquad \text{(We used } \sigma'(x) = \sigma(x)(1-\sigma(x))\text{)} \\
&= \sum_{j=1}^{m} c_{ij} \cdot (1-\sigma(\theta_i-\beta_j)) - (1-c_{ij}) \cdot \sigma(\theta_i-\beta_j) \\
&= \sum_{j=1}^{m} c_{ij} - c_{ij} \cdot \sigma(\theta_i-\beta_j) - \sigma(\theta_i-\beta_j) + c_{ij} \cdot \sigma(\theta_i-\beta_j) \\
&= \sum_{j=1}^{m} c_{ij} - \sigma(\theta_i-\beta_j)
\end{aligned}$$

$$
\begin{aligned}
\frac{\partial log(p(C|\theta,\beta))}{\partial \beta_j} &= \sum_{i=1}^{n} \frac{c_{ij}}{\sigma(\theta_i-\beta_j)} \cdot \frac{\partial \sigma(\theta_i-\beta_j)}{\partial(\theta_i-\beta_j)} \cdot \frac{\partial(\theta_i-\beta_j)}{\partial \beta_j} \\
&\quad + \frac{1-c_{ij}}{1-\sigma(\theta_i-\beta_j)} \cdot \frac{\partial(1-\sigma(\theta_i-\beta_j))}{\partial(\theta_i-\beta_j)} \cdot \frac{\partial(\theta_i-\beta_j)}{\partial \beta_j} \\
&= \sum_{i=1}^{n} \frac{c_{ij}}{\sigma(\theta_i-\beta_j)} \cdot \sigma(\theta_i-\beta_j) \cdot (1-\sigma(\theta_i-\beta_j)) \cdot -1 \\
&\quad + \frac{1-c_{ij}}{1-\sigma(\theta_i-\beta_j)} \cdot -\sigma(\theta_i-\beta_j) \cdot (1-\sigma(\theta_i-\beta_j)) \cdot -1 \\
&\qquad\qquad \text{(We used } \sigma'(x) = \sigma(x)(1-\sigma(x))\text{)} \\
&= \sum_{i=1}^{n} -c_{ij} \cdot (1-\sigma(\theta_i-\beta_j)) + (1-c_{ij}) \cdot \sigma(\theta_i-\beta_j) \\
&= \sum_{i=1}^{n} -c_{ij} + c_{ij} \cdot \sigma(\theta_i-\beta_j) + \sigma(\theta_i-\beta_j) - c_{ij} \cdot \sigma(\theta_i-\beta_j) \\
&= \sum_{i=1}^{n} -c_{ij} + \sigma(\theta_i-\beta_j)
\end{aligned}
$$

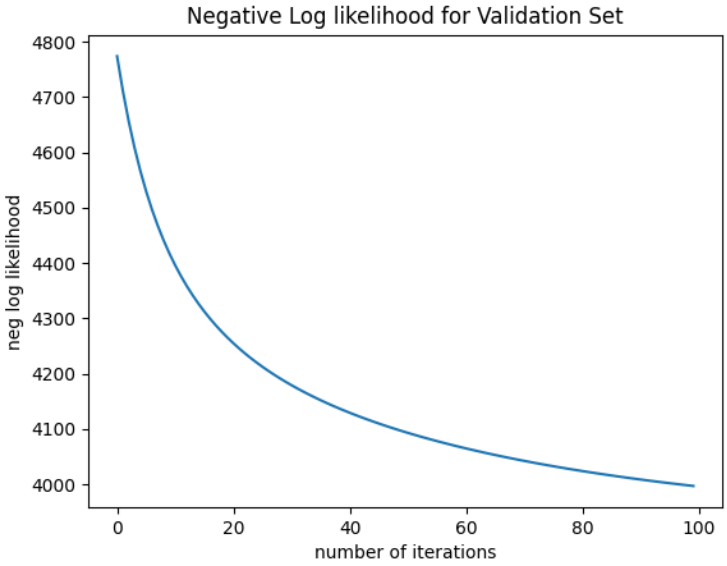
(b) Hyperparameters selected for irt:

```
hyperparameters:
 num_iteration = 100, learning rate = 0.001
```

Negative log-likelihood in each interaction for the training set:



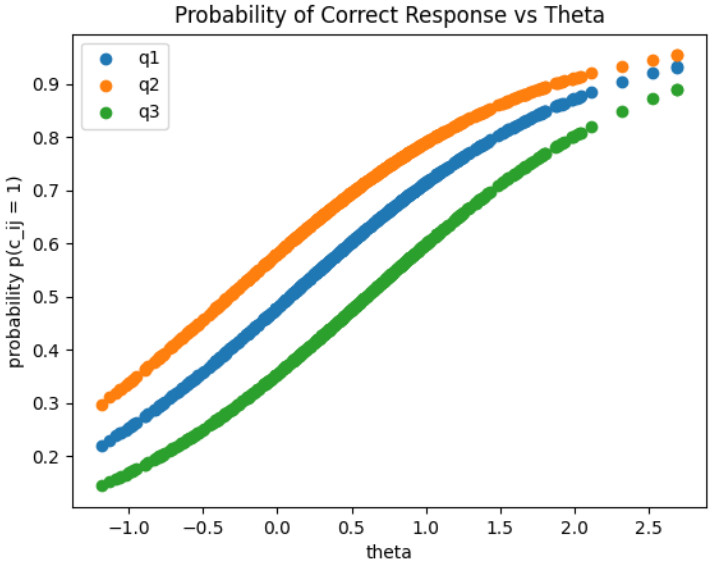
Negative log-likelihood in each interaction for the validation set:



(c) Final validation and test accuracy:

```
val accuracy:
0.7074513124470787
test accuracy:
0.6965848151284222
```

(d) Plot for the probability of getting correct response vs theta for 3 questions:



From this graph, we can see that when theta is small, the probability of correct responses is low and as theta increases the probability of correct responses also increases. This makes sense in the

context of the problem since theta represents a student's ability. When a student has low ability they are less likely to answer a question right, whereas a student with high ability is more likely to get a question right. The 3 curves in the graph roughly follow an S-shape since we used the sigmoid function $\sigma(\theta - \beta)$ to calculate the probability. As θ increase, the input to sigmoid, $\theta - \beta$, also increase and since sigmoid is monotonically increasing with an S-shape with respect to its input, the probability will roughly follow an S-shape. The difference in heights(vertical displacement) of the 3 curves could represent the different difficulty levels of questions. That is, in questions with lower difficulty, all students (regardless of ability) is more likely to get it right and in questions with higher difficulty, all students (regardless of ability) is more likely to get it wrong.

Q3 Neural Networks

(a)

1. ALS and Neural Networks have different training processes. ALS updates parameters alternatingly by fixing one and updating another, whereas neural network updates all parameters at once through a backward pass.

2. ALS and Neural Networks minimize different objectives. From tutorial: ALS is set up such that a student i is being assigned a vector u_i and a question j is being assigned a vector v_j. Then the values of these vectors are trained to minimize $\Sigma_{(i,j)\in O}(C_{ij} - u_i \cdot v_j)^2$, where C_{ij} represents whether the student i answers question j correctly or not. In contrast, the neural networks approach represent each student by a vector $v \in S$ of dimension num_questions, and then project the student vectors onto a lower dimensional space so that similar students are clustered together (source: Nov.16 tutorial), where noise is reduced and important features extracted. Eventually, the lower-dimensional vectors are reconstructed back to the original input dimensions. The objective to be minimized is the difference between the input vectors v and their reconstructions $f(v)$, given by $\Sigma_{v\in S}|v - f(v)|^2$.

3. ALS has a closed form solution update rule, whereas neural networks do not.

(b)

See neural_network.py codes.

(c)

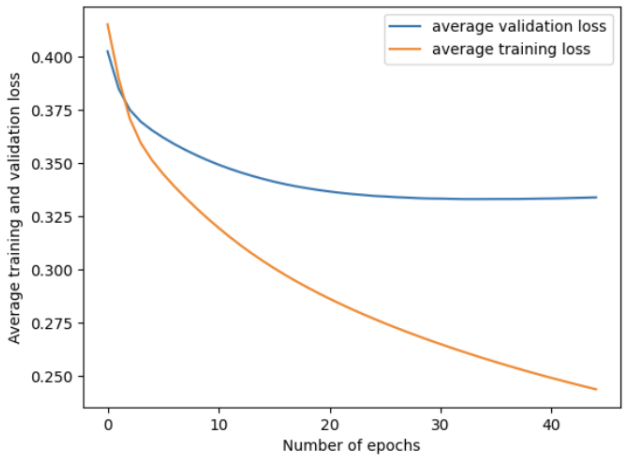
We first tune the hyper-parameters learning rate and number of epochs, which was accomplished by fixing one parameter and tuning the other. After trying out num_epochs $= \{10, 20, 30, 50, 80, 100\}$, we noticed that for num_epochs ≥ 50 the validation accuracy will begin to decrease due to overfitting and the optimal num_epochs $= 45$. We also tried $lr \in \{0.001, 0.005, 0.01, 0.025, 0.05, 0.1\}$ and chose $lr = 0.025$ as the optimal learning rate because it showed promising convergence time and the validation accuracy decreased without any fluctuation (unlike some larger learning rates). After fixing learning rate and number of epochs, we trained the neural network using $k \in \{10, 50, 100, 200, 500\}$. We did notice that different k values cause the validation accuracy to converge at different rates (for example, $k = 10$ needs about 50 epochs to reach maximum validation accuracy, whereas $k = 200$ needs only about 15 epochs before validation accuracy reaches maximum and starts decreasing due to overfitting). The best result (compared across the maximum validation accuracy achieved for different k) was $k* = 10$, with a final validation accuracy around 0.685.

(d)

After fixing $lr = 0.025, k* = 10, num\_epochs = 45$, we plotted the training and validation objectives (a.k.a loss) for each epoch in Figure 1. Because the training and validation datasets have different number of non-zero inputs, we decided to follow the TA's advice and plotted the *average* training and validation loss instead. This was accomplished through dividing the loss by the number of non-zero inputs in that dataset.

Final validation accuracy: 0.685

Final test accuracy: 0.678




```
Epoch: 30, Training loss: 8915.643048405647, Validation Loss: 1425.082674376603
Epoch: 31, Training loss: 8854.964206695557, Validation Loss: 1425.022189602206
Epoch: 32, Training loss: 8796.11555325985, Validation Loss: 1424.6134399131638
Epoch: 33, Training loss: 8739.037480711937, Validation Loss: 1425.227509259231
Epoch: 34, Training loss: 8684.076188087463, Validation Loss: 1425.181150791531
Epoch: 35, Training loss: 8628.827282071114, Validation Loss: 1424.964001087122
Epoch: 36, Training loss: 8576.051241755486, Validation Loss: 1425.953527430843
Epoch: 37, Training loss: 8525.289509773254, Validation Loss: 1425.655049382334
Epoch: 38, Training loss: 8474.798636555672, Validation Loss: 1426.045781356184
Epoch: 39, Training loss: 8426.385057210922, Validation Loss: 1427.290795800358
Epoch: 40, Training loss: 8378.680794477463, Validation Loss: 1428.330543042745
Epoch: 41, Training loss: 8332.114373803139, Validation Loss: 1428.764126509417
Epoch: 42, Training loss: 8286.188021540642, Validation Loss: 1428.871143660118
Epoch: 43, Training loss: 8241.352742552757, Validation Loss: 1430.411757678858
Epoch: 44, Training loss: 8198.628517746925, Validation Loss: 1430.843061406020
```

Figure 1: Average training and validation loss over number of epochs (without regularization). Notice that there seem to be a slight overfitting during the last 10 epochs since validation loss started to increase a bit.

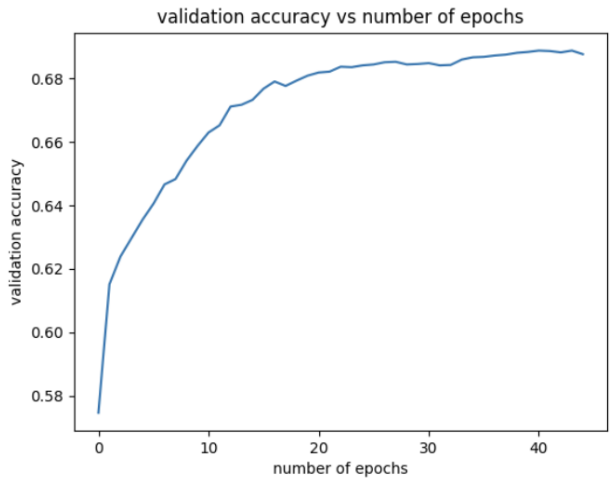
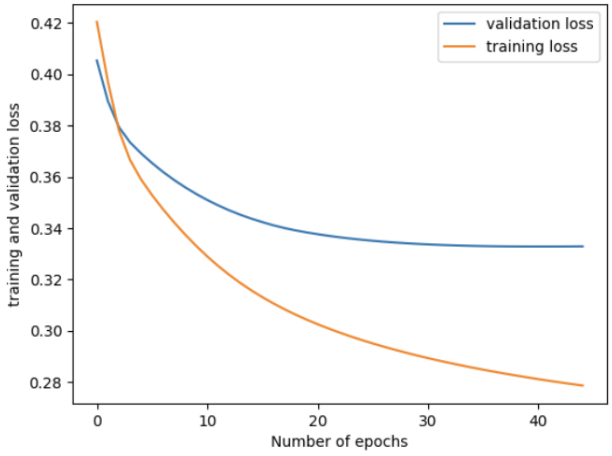


Figure 2: Change in validation accuracy over number of epochs (without regularization).

(e)

We noticed that the validation loss shows a very small increasing trend after about 30 epochs, which is likely due to slight overfitting of the data. In order to address this issue, we added a L^2 regularization term after the loss function and tried different λ amongst {0.001, 0.01, 0.1,1}. Except for 0.001, all

of the other λ values result in worse validation accuracy than achieved without regularization. With $\lambda = 0.001$, however, the problem of overfitting seemed to be resolved since the average validation loss continued to decrease even after 40 epochs (see in Figure 3). In addition, the final validation accuracy increased to 0.687. Thus, we conclude that the L^2 regularization improves the model by reducing overfitting.
Final validation accuracy: 0.687
Final test accuracy: 0.681




```
Epoch: 31, Training loss: 9796.03313779831, Validation Loss: 1414.263363629732
Epoch: 32, Training loss: 9763.340292930603, Validation Loss: 1413.54041417372
Epoch: 33, Training loss: 9732.11213350296, Validation Loss: 1412.921267414747
Epoch: 34, Training loss: 9702.250502586365, Validation Loss: 1412.39362007175
Epoch: 35, Training loss: 9673.666892290115, Validation Loss: 1411.94694328533
Epoch: 36, Training loss: 9646.281157970428, Validation Loss: 1411.57207404890
Epoch: 37, Training loss: 9620.019968032837, Validation Loss: 1411.26074730600
Epoch: 38, Training loss: 9594.814890384674, Validation Loss: 1411.00636027966
Epoch: 39, Training loss: 9570.60101723671, Validation Loss: 1410.804661954815
Epoch: 40, Training loss: 9547.315816640854, Validation Loss: 1410.65282001892
Epoch: 41, Training loss: 9524.904635429382, Validation Loss: 1410.54431553432
Epoch: 42, Training loss: 9503.330454587936, Validation Loss: 1410.46791469174
Epoch: 43, Training loss: 9482.566520929337, Validation Loss: 1410.41866147012
```

Figure 3: Average training and validation loss over number of epochs (with L^2 regularization). Notice that the validation loss continues to decrease for the last 10 epochs.

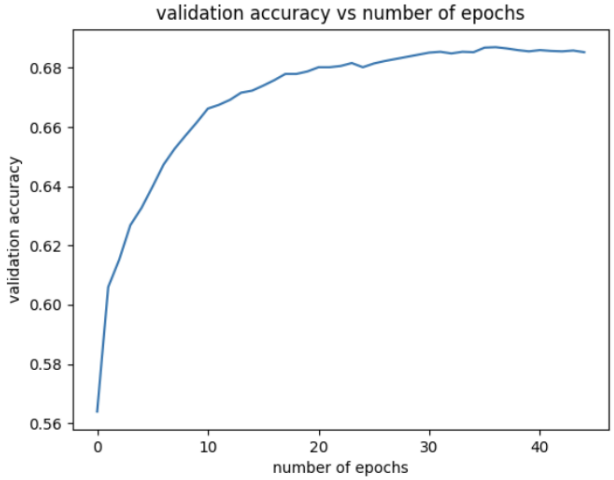


Figure 4: Validation accuracy with L^2 regularizatio of $\lambda = 0.001$.

Q4 Ensemble

- **Explain the ensemble process you implemented.**
  For this part, we choose the Item Response Theory algorithm to implement the bagging ensemble and improve the stability and accuracy of our base models. Basically, we do the following steps:
  Step 1: Resample to get 3 new training sets from the given dataset. Each training set has the same number of samples randomly chosen from the given dataset with replacement.
  Step 2: Compute the prediction $\mathbf{y}_i$ using the training set i.
  Step 3: Compute the average prediction $\mathbf{y} = \frac{1}{3}\sum_{i=1}^{3}\mathbf{y}_i$. Since this task is a binary classification (whether user i answer question j correct or not), we apply a threshold of 0.5 to get the final prediction. $\mathbf{y}_{bagged} = \mathbb{I}(\frac{1}{3}\sum_{i=1}^{3}\mathbf{y}_i > 0.5)$
  Step 4: Compare with validation data and calculate accuracy of ensemble.
- **Report the final validation and test accuracy.**
  We use hyperparameter: num_iteration = 280 and learning rate = 0.0005.

```
The validation accuracy of ensemble is 0.7071690657634773
The test accuracy of ensemble is 0.6965848151284222
```

- **Do you obtain better performance using the ensemble? Why or why not?**
  The performance on validation set is slightly lower but the performance on test set is the same. Notice that the difference between validation and test accuracy of ensemble is smaller than the original model. This means that the ensemble model is more stable on prediction than the original model.
  By using bagging ensemble, the model averages multiple noises. It reduces the variance but does not change the bias and Bayes error. By theory, the ensemble model should improve accuracy. However, in our case there are only 3 base models. The amount of base models might not be enough to remarkably improve accuracy. As the result, our test accuracy does not increase.

Part B

Formal description:
For part B, we choose the Item Response Theory algorithm to implement extension.

- *Extension 1: Add parameters α_j that describe how discriminative the question is*
  Notice that in the original model, it made a simplified assumption that the correct answer probability only depends on two parameters θ_i, the ability of student i, and β_j, the difficulty of question j. However, all the questions are treated to be equally discriminative, which will probably not be the case in real life situation.
  Therefore, we decide to add parameters α_j that describe how discriminative each question is. Figure 5 in **Figure / Diagram** section shows the general idea of how different α values affects the probability. When x is negative, the blue line is lower than the red line; while x is positive, the blue line is higher than the red line. This indicates that students with lower ability will be less likely to get the correct answer on question that is more discriminative, and vice versa.
  We add α to the model, calculate the derivative of the log-likelihood with respect to α_j and perform gradient descent on α just like θ and β to maximize the log-likelihood. We also set a constraint on α to be between (0,2), since that is a more realistic range as suggested by Columbia Public Health. For further references see Item Response Theory.
- *Extension 2: Add hyperparameter c that describe the probability of getting the correct answer by random guess*
  We assume all the questions to be multiple choice questions with 4 options (just like the example diagnostic question provided in the handout). Consider there are possibility where students don't know how to do the question and randomly choose an option, we decide to also include a hyperparameter c in the model. We tuned the hyperparameter c and get the optimal value c = 0.25.

After the extension, our probability function becomes:

$$\mathcal{P}(c_{ij}|\theta_i, \beta_j) = c + [1 - c] \times \text{sigmoid}(\alpha_j(\theta_i - \beta_j))$$

Our extended model consider two additional factors: the discrimination of questions and students' random guess, which adds more complexity and can better capture the characteristics of the dataset. Therefore, we expect our extended model to improve the optimization and get a better test accuracy.

Figure / Diagram

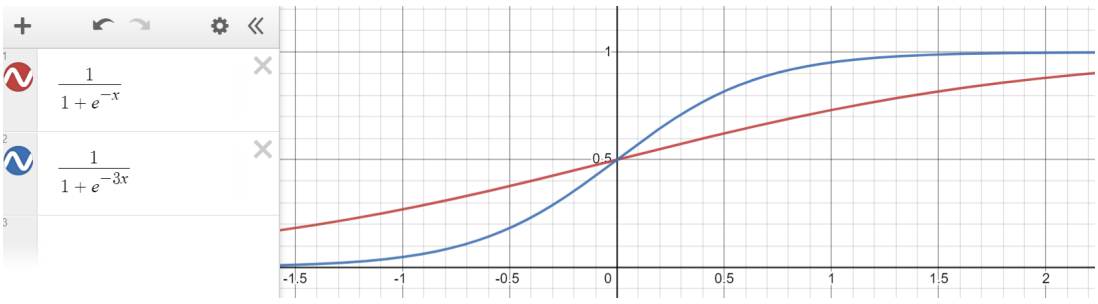


Figure 5: Probability of correct response for different α value

Comparison / Demonstration:

Recall that our hypothesis is newly added parameter α and c will help improve performance because it increases the complexity of the model to capture more characteristics of the dataset. To test our hypothesis, we will implement the modified model as described above and run on the dataset to see its performance. (The code for this section is in \part_a\part2_irt.py)

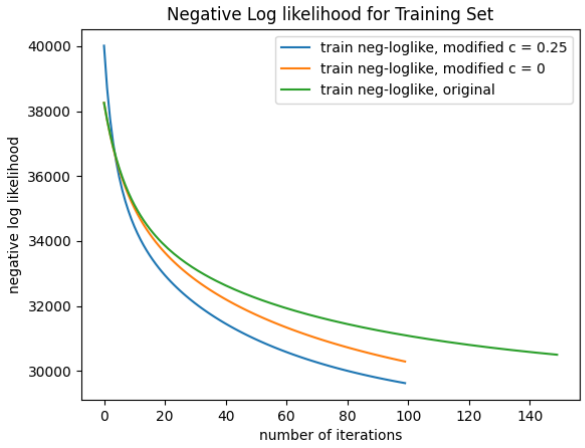
We chose 3 models, the original irt model, modified irt model with c = 0 (representing no random guess probability), and modified irt model with c = 0.25(25 percent chance of getting a question right through random guess), to compare the performance of modified algorithm.

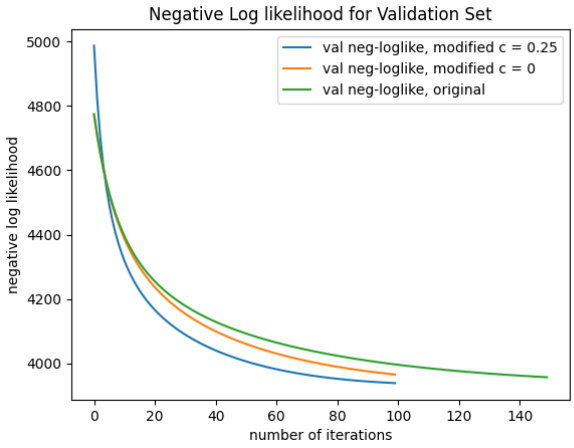
For a set of 9 different of hyperparameters, which is the combination of loop iterations $\{100, 150, 200\}$ and learning $\{0.0025, 0.001, 0.0005\}$, we will find the best validation accuracy each model can achieve. The resulting validation accuracies are summarized in the table below.

	Iterations								
	100			150			200		
learning rate	original irt	modified irt, c = 0	modified irt, c = 0.25	original irt	modified irt, c = 0	modified irt, c = 0.25	original irt	modified irt, c = 0	modified irt, c = 0.25
0.0025	0.70759	0.70745	**0.70900**	0.70590	0.70787	0.70759	0.70562	0.70687	0.70703
0.001	0.70745	**0.70886**	0.69433	**0.70802**	0.70844	0.70447	0.70675	0.70773	**0.70900**
0.0005	0.69785	0.69574	0.67556	0.70745	0.70590	0.68882	0.70745	0.70872	0.69461

As we can see from this table, the maximum validation accuracy achieved by the modified model with c = 0.25 (0.70900) is greater than the maximum accuracy achieved by the modified model with c = 0 (0.70886). And the maximum accuracy achieved by the two modified models is both greater than the maximum validation accuracy of the original model (0.70802).

This shows that adding parameter α and c does help improve performance in some sense.





By looking at the change in negative log-likelihood over iterations for the training set and the validation set, we can see the correspondence between the negative log-likelihood curves and validation accuracies for different models in the table. For example, the negative log-likelihood for the modified model with c = 0.25 is lower than the two other curves most of the time, indicating the likelihood for the prediction of the modified model(c = 0.25) to be correct is higher. This is consistent with the modified model(c = 0.25) having the highest validation accuracy.

The test accuracies for the 3 models also indicated an improvement in performance for the modified model(c = 0.25).

```
hyperparameters used: num iteration = 100, learning rate = 0.0025
val accuracy modified c = 0.25: 0.7090036692068868
test accuracy modified c = 0.25: 0.7044877222692634
hyperparameters used: num iteration = 100, learning rate = 0.001
val accuracy modified c = 0: 0.7088625458650861
test accuracy modified c = 0: 0.6957380750776179
hyperparameters used: num iteration = 150, learning rate = 0.001
val accuracy original: 0.7080158058142817
test accuracy original: 0.6996895286480384
```

However, we also see that the increase in validation accuracy is not significant.
For the modified model(c = 0.25), the little increase in accuracy could indicate that our assumption of all questions are multiple choice with four options was not a good one. Using a uniform c value could have limited the accuracy to become higher as different questions may have a different number of choices.

Limitations and Further Extensions

Although we have achieved a good validation and test accuracies with our extended IRT model, there are some limitations that cause it to perform poorly under certain circumstances. Some of the limitations are summarized below with possible extensions to resolve these issues.

1. In this project, we have no information in regards to how many options are present in each multiple-choice question. We assumed that all questions are multiple-choice questions with only one single correct answer out of 4 options. However, this may not always be the case, which makes our assumption of a uniform c value ($c = 0.25$ for 4 options) not necessarily correct. Because c is a hyper-parameter here, a poor choice of c may cause the data to underfit for situations in which different questions have different numbers of options. Hence, a good way to avoid this problem is to use a vector $\mathbf{c} = (c_1, ..., c_{N_{questions}})$ to account for the probability randomly guessing a problem correct. For example. c_j would represent the probability of guessing the question j correctly.

2. Since we have trained all four models (KNN, IRT, NN, and ensemble) on a dataset of multiple choice questions, we would naturally expect all of these models to perform poorly on datasets involving other form of questions, such as short-answer or fill-in-the-blank type of questions. Instead of assuming a dichotomous outcome (correct/incorrect) for all questions, a possible extension of the project is to accommodate for polytomous outcomes - in which every response receives a different score within a range, such as being rated on a scale from 1-5 (source: Item Response Theory Wikipedia page).

3. In our current IRT model, we assign a single ability value θ_i for a student i. We predict whether they can answer a question j correctly solely based on 1) their ability θ_i,2)question j's difficulty β_j, and 3) how discriminative question j is (α_j). The problem with this approach is its failure to account for a student's variable ability in different topis and tasks. For example student i may be excellent in every other subject, but fails to grasp certain concepts in multivariable calculus. But due to their good performance on other subjects, they still get a very high ability value θ_i, which would likely to cause both the original and extended IRT model to overestimate this student's probability of getting a multivariable calculus question correct. One possible improvement to resolve this issue is to construct an ability matrix θ to represent each students' individual abilities in different subjects,

where $\theta_{i,n}$ represents student i's ability in subject n. This matrix can be constructed by using the question_meta.csv file as follows: 1) Initialize an all-zero θ matrix of shape $(N_{students}, N_{subjects})$; 2) create a dictionary called question_subject that maps each question to an array to subjects they belong to; 3) for each student i, look at all the questions they answered correctly, iterate through question_subject for each question, and add 1 to $\theta_{i,n}$ for each subject n these correctly-answered questions belong to. We will normalize $\theta_{i,n}$ by dividing the previously computed $\theta_{i,n}$ by total number of questions in subject n. The updated probability that a student i answers question j correctly should be given by:

$$p(c_{ij}|\theta, \beta) = \sigma(\frac{1}{|N|}\Sigma_{n \in N}(\theta_{i,n}) - \beta_j)$$

Here, N is the set of subjects that question j belongs to, and σ is the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$. We believe this extension will increase the accuracy of the IRT model on predicting the correctness of any (student, question) pair unseen before.