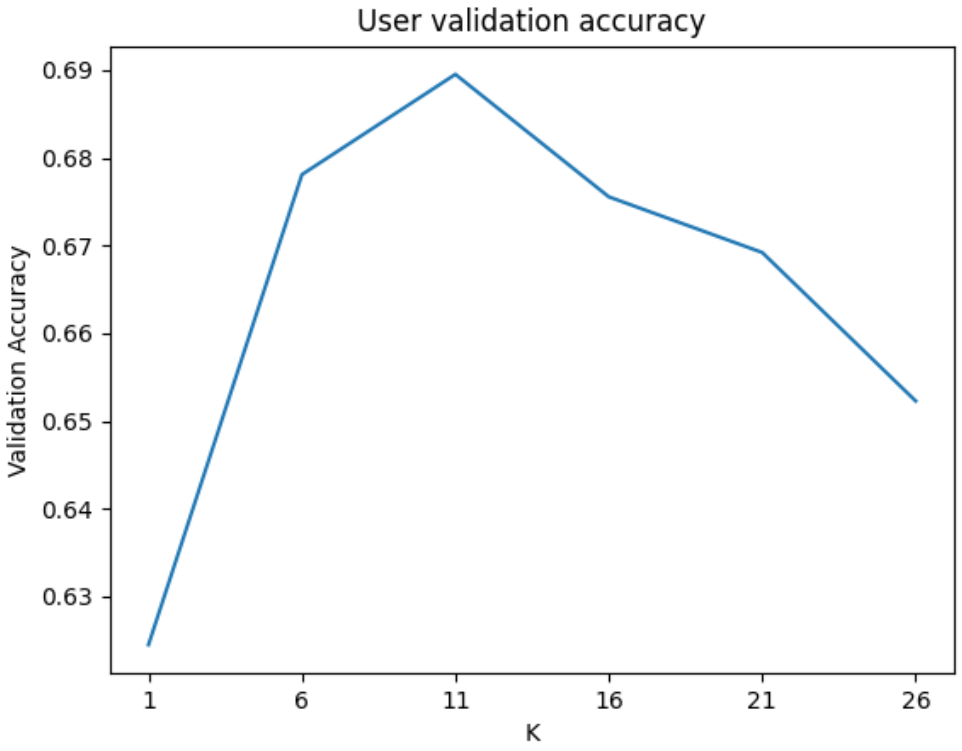
# CSC311 Final Project

Aditya Gulati, Kavin Singh

## *Part A*

### 1. K-Nearest Neighbor

a)




Validation accuracy as a function of K

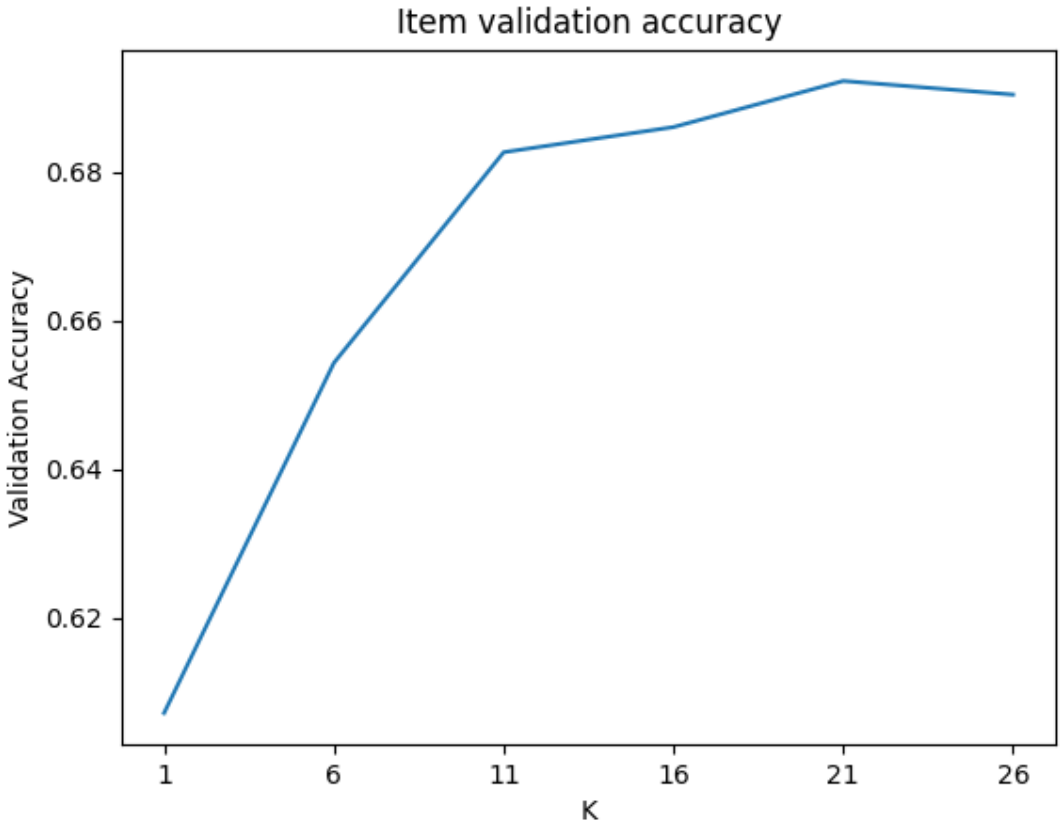
| K | Validation Accuracy |
|---|---|
| 1 | 0.6244707874682472 |
| 6 | 0.6780976573525261 |
| 11 | 0.6895286480383855 |
| 16 | 0.6755574372001129 |
| 21 | 0.6692068868190799 |
| 26 | 0.6522720858029918 |

b)

$K^*$ is 11 with the Highest Validation Accuracy of 0.6895 based on Students. The Final Test Accuracy 0.6842.

c)

The core underlying assumption is that if question A has the same students with correct or incorrect answers as question B, students' correctness on question A will be the same as question B.




| K | Validation Accuracy |
|---|---|
| 1 | 0.607112616426757 |
| 6 | 0.6542478125882021 |
| 11 | 0.6826136042901496 |
| 16 | 0.6860005644933672 |
| 21 | 0.6922099915325995 |
| 26 | 0.69037538808919 |

$K^*$ is 21 with the Highest Validation Accuracy of 0.6922 based on Questions—the Final Test Accuracy 0.6816.

d)

Item-based $K^*$ = 21 has the better Validation Accuracy than User-based $K^*$ = 11. Although, User-based $K^*$ has slightly better test accuracy than the Item-based $K^*$. Lower User-based K values have better Validation Accuracy than Item-based K values.

e)

1. KNN is very computationally expensive. The algorithm would not be efficient with large datasets.
2. KNN does not work with higher dimensions. If multiple factors are included in the dataset, the algorithm will not give accurate results.
3. KNN treats every data point with similar potential, making looking for trends more demanding among the data points.

## 2. Item Response Theory

a)

Define $A_{ij} = (2c_{ij} - 1)(\theta_i - \beta_j)$

- N = # students
- Q = # questions

$$P(c_{ij} = 1 \mid \theta_i, \beta_j) = \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)} = \sigma(\theta_i - \beta_j)$$

$$\Rightarrow P(c_{ij} = 0 \mid \theta_i, \beta_j) = 1 - \sigma(\theta_i - \beta_j) = 1 - \sigma(-(\theta_i - \beta_j))$$

$$= \sigma(\beta_j - \theta_i) \text{ by property of sigmoid function}$$

General Function:

$$P(c_{ij} \mid \theta_i, \beta_j) = \sigma((2c_{ij} - 1)(\beta_j - \theta_i)) = \frac{e^{A_{ij}}}{1 + e^{A_{ij}}}$$

Since the data is independent we get likelihood function

$$P(c \mid \theta, \beta) = \prod_{i=1}^{N} \prod_{j=1}^{Q} p(c_{ij}; \theta_i, \beta_j) = \prod_{i=1}^{N} \prod_{j=1}^{Q} \frac{e^{A_{ij}}}{1 + e^{A_{ij}}}$$

Log Likelihood is

$$\log P(c; \theta, \beta) = \sum_{i=1}^{N} \sum_{j=1}^{Q} \log\left(\frac{e^{A_{ij}}}{1 + e^{A_{ij}}}\right)$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{Q} A_{ij} - \log(1 + e^{A_{ij}})$$

Derivate w.r.t $\theta_i$ and $\beta_j$

$$\frac{\partial \log(c; \theta, \beta)}{\partial \theta_i} = \sum_{j=1}^{Q} \frac{\partial A_{ij}}{\partial \theta_i} - \frac{\partial}{\partial \theta_i} \log(1 + e^{A_{ij}})$$

$$= \sum_{j=1}^{Q} (2c_{ij} - 1) - (2c_{ij} - 1)\frac{e^{A_{ij}}}{1 + e^{A_{ij}}} \qquad \text{let } A'_{ij} = 2c_{ij} - 1$$

$$= \sum_{j=1}^{Q} A'_{ij} - A'_{ij}\sigma(A_{ij}) = \sum_{j=1}^{Q} A'_{ij}(1 - \sigma(A_{ij}))$$

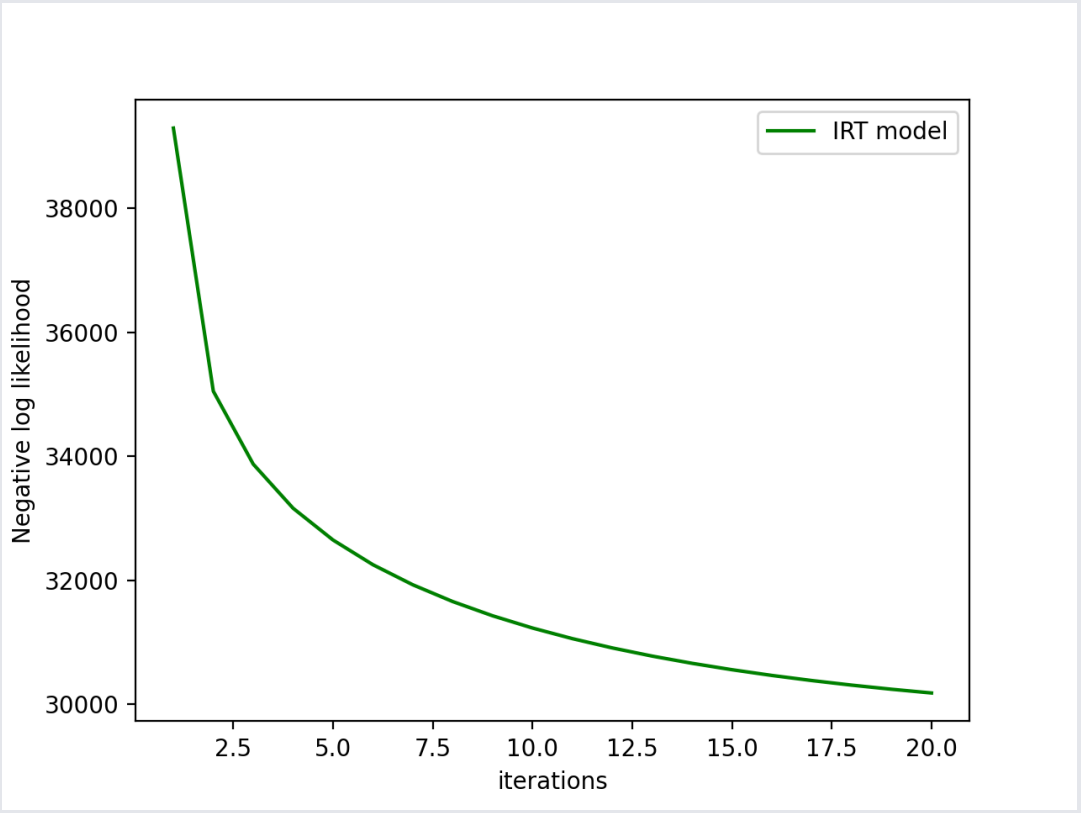
Similarly,

$$\frac{\partial \log(c; \theta, \beta)}{\partial \beta_j} = \sum_{i=1}^{N} -(2c_{ij} - 1)(1 - \sigma(A_{ij}))$$

b)

The hyperparameter:

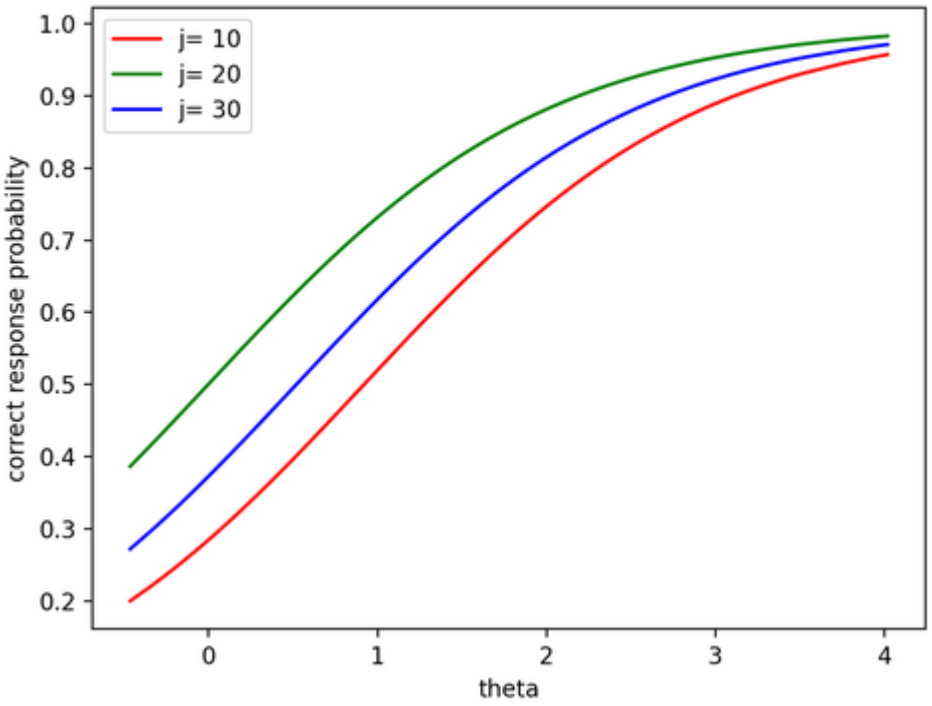
- **Number of epochs:** 20
- **Learning rate:** 0.01




c)

- Final validation accuracy: **0.7060400790290714**
- Test Accuracy: **0.7050522156364663**

d)




$j_1 = 10, j_2 = 20, j_3 = 30$

The curves follow the shape of the sigmoid function since our probability density function is a sigmoid function. They represent the students' probability of answering correctly given their ability (i.e., theta). The x-axis is the theta, i.e., the student performance, and the y-axis is the predicted probability of the student answering the question correctly.

# 3. Neural Networks

a)

1. In ALS, the stochastic gradient is used, but we use backpropagation to optimize the ensure neural networks.
2. In neural networks, we minimize the objective function, but in ALS, we reduce the respective loss function.
3. In neural networks, we computed the composition of activator functions to get the score, but in ALS, we use the dot product of two-parameter matrices.

b) We implemented the AutoEncoder code.

c)

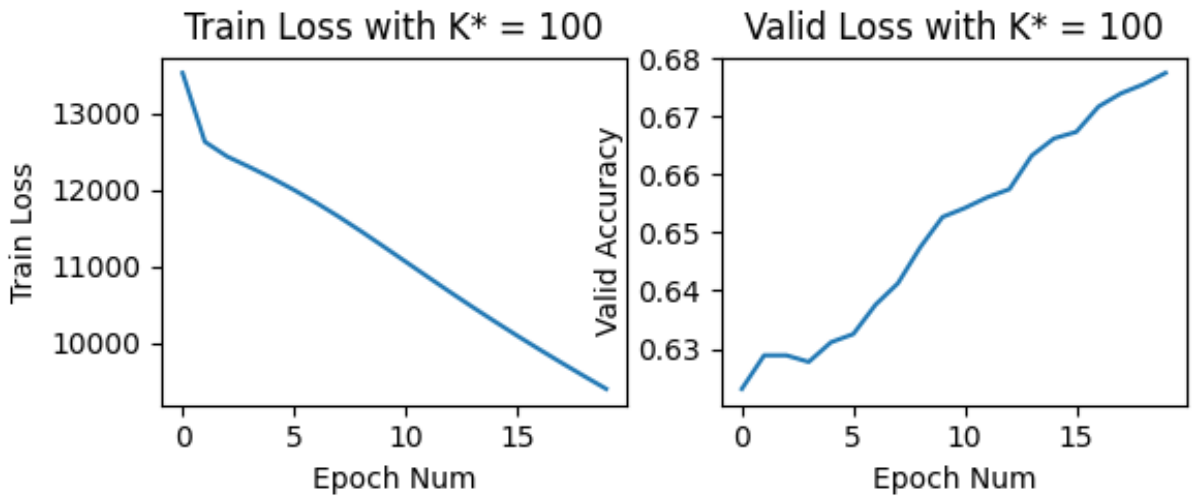
The hyperparameters are:

- Learning Rate: 0.01
- Epoch Number: 20

| K | Validation Accuracy |
|---|---|
| 10 | 0.6504374823595823 |
| 50 | 0.6751340671747107 |
| 100 | 0.6790855207451313 |
| 200 | 0.676545300592718 |
| 500 | 0.6690657634772792 |

*K* = *100* has the best validation accuracy of 0.6791 hence we choose $K^*$ = 100.

d)




$K^*$ = 100 with the final Validation Accuracy is 0.6791
$K^*$ = 100 with Test Accuracy is 0.6729

e) The values of $\lambda$ with their validation accuracy with $K^* = 100$

| $\lambda$ | Validation Accuracy |
|---|---|
| 0.001 | 0.6766864239345187 |
| 0.01 | 0.6713237369460909 |
| 0.1 | 0.6268698842788597 |
| 1 | 0.6244707874682472 |

$\lambda$ = 0.001 has the highest validation of 0.6767 among other $\lambda$s. The final Validation Accuracy with regularization is 0.6767 and without penalty is 0.6791, and test, Accuracy with regularization is 0.6726 and without penalty is 0.6729. The algorithm with regularization has slightly worse results than with no penalty.

# 4. Ensemble

We have heard of KNNs and Neural Networks ensembles but not probabilistic models. Hence, we decided to investigate the performance of a choir composed of only the baseline item response theory (IRT) model as described in question 2. We trained three baseline IRT models with different parameters to validate and hopefully improve the accuracy of the base models. This comprised of a two-step process:

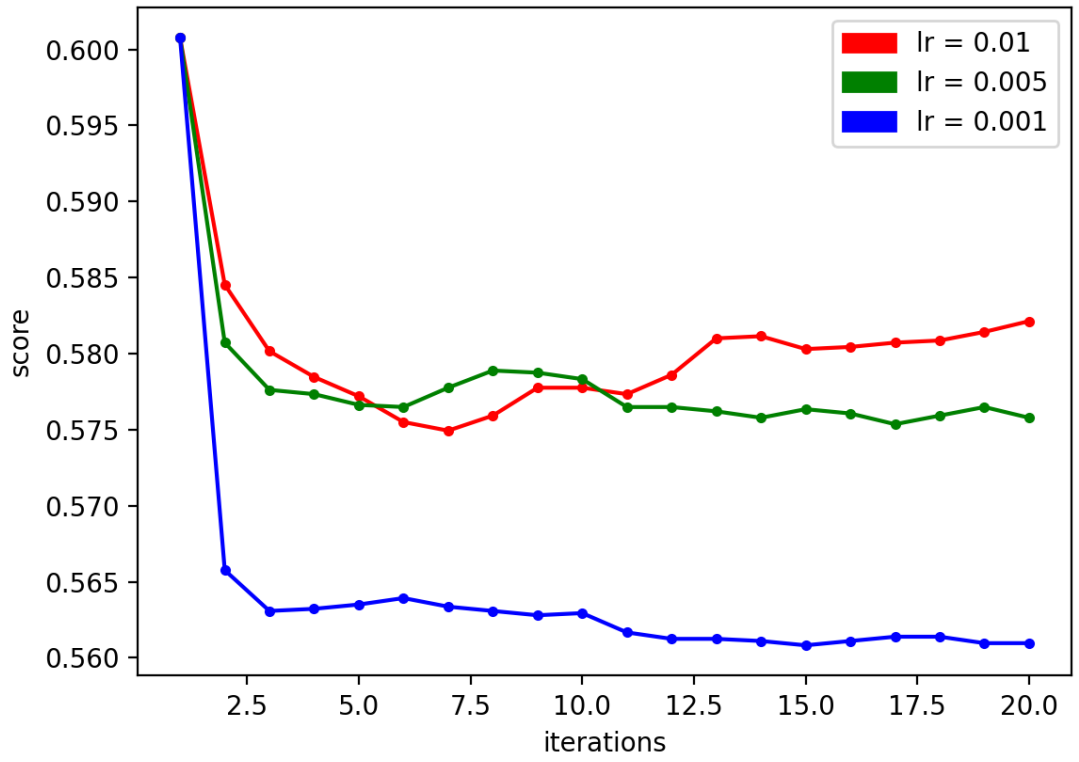
**Bootstrapping:** First, we implemented a bootstrap function that resampled the N students from the training dataset with replacement to provide three separate training sets for the base models. We hypothesize that doing this would reduce the variation in the accuracy.

**Training & Testing:** We trained the baseline models after retrieving the resampled dataset and forming three new training datasets. Once the models are trained, we evaluate the models on the testing data set and compute the average for the accuracy as our final prediction.

The hyperparameters for the three models are:

- Learning rate = 0.01 *(Model 1)*, 0.005 *(Model 2)*, 0.001 *(Model 3)*
- Epochs = 20

Here are the results:



| Model | Validation Accuracy | Test Accuracy |
|---|---|---|
| Model 1 | 0.5821337849280271 | 0.5949760090318938 |
| Model 2 | 0.5757832345469941 | 0.5828394016370307 |
| Model 3 | 0.560965283657917 | 0.5670335873553486 |
| Ensemble | 0.57296076771098 | 0.58161633267476 |

Firstly, we see that the performance of each model is worse than before training the model. This explains the poor performance of the ensemble as well. This could be because the resampling in the dataset has caused the dataset not to be independent anymore. We also see that the model with a lower learning rate performs better than a model with a higher learning rate. This means that the learning rate starts negatively affecting the performance as it gets lower than 0.01. Interestingly, despite the negative log-likelihood of all the models consistently decreasing over the 20 epochs, the implementation only worsens.

# Part B

## 1. Formal Description

We are trying to improve the Item Response Theory(IRT) algorithm from Part A.

We made a coefficient called a difficulty factor($D_j$) for questions. The difficulty factor determines the difficulty of the question. The more students responded to the question precisely, the easier the question.

The difficulty factor helps in adjusting $\beta_j$ more realistically than IRT. It is calculated by taking the average of all the sum of questions and dividing it by the sum of the question $j^{th}$ was answered correctly.

$$D_j = \frac{\frac{\sum_{i=1}^{N}\sum_{k=1}^{Q} c_{ik}}{N}}{\sum_{i=1}^{N} c_{ij}}$$

When the sum of the question is closer to the mean of the sum of questions, then the value of $D_j$ is around one, i.e., the $j^{th}$ the question is of average difficulty, if the sum is lesser than the mean, then the value of, i.e., The question is tricky. If the sum is more than the mean value, the question is easy to solve.

We are trying to adjust the difference of $\theta_i - \beta_j$ with the difficulty of questions.

The difficulty factor decreases or increases the weight of $A_{ij}$. The purpose of difficulty is to assume that if the questions tend to be difficult, there is less chance that the student can answer them correctly. The more complex the question is, the more unlikely the student can solve it.

We have a similar method for deriving the updates for $\theta_i$ and $\beta_j$ from part A. We are changing it by just multiplying $D_j$ with $\theta_i - \beta_j$

Define $A_{ij} = (2c_{ij} - 1)\, D_j (\theta_i - \beta_j)$

- $N$ = # students
- $Q$ = # questions

$$P(c_{ij} = 1 \mid \theta_i, \beta_j) = \frac{\exp(D_j(\theta_i - \beta_j))}{1 + \exp(D_j(\theta_i - \beta_j))} = \sigma(D_j(\theta_i - \beta_j))$$

$$P(c_{ij} = 0 \mid \theta_i, \beta_j) = 1 - \sigma(D_j(\theta_i - \beta_j)) = 1 - \sigma(-D_j(\theta_i - \beta_j))$$

$$= \sigma(D_j(\beta_j - \theta_i)) \quad \text{by property of sigmoid function}$$

General Function

$$P(c_{ij} \mid \theta_i, \beta_j) = \sigma((2c_{ij} - 1) D_j(\beta_j - \theta_i)) = \frac{e^{A_{ij}}}{1 + e^{A_{ij}}}$$

Since the data is independent we get likelihood function

$$P(C \mid \theta, \beta) = \prod_{i=1}^{N} \prod_{j=1}^{Q} P(c_{ij}, \theta_i, \beta_j) = \prod_{i=1}^{N} \prod_{j=1}^{Q} \frac{e^{A_{ij}}}{1 + e^{A_{ij}}}$$

Log Likelihood

$$\log P(C; \theta, \beta) = \sum_{i=1}^{N} \sum_{j=1}^{Q} \log\left(\frac{e^{A_{ij}}}{1 + e^{A_{ij}}}\right)$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{Q} A_{ij} - \log(1 + e^{A_{ij}})$$

Derivate w.r.t $\theta_i$ and $\beta_j$

$$\frac{\partial \log(C; \theta, \beta)}{\partial \theta_i} = \sum_{j=1}^{Q} \frac{\partial A_{ij}}{\partial \theta_i} - \frac{\partial}{\partial \theta_i} \log(1 + e^{A_{ij}})$$

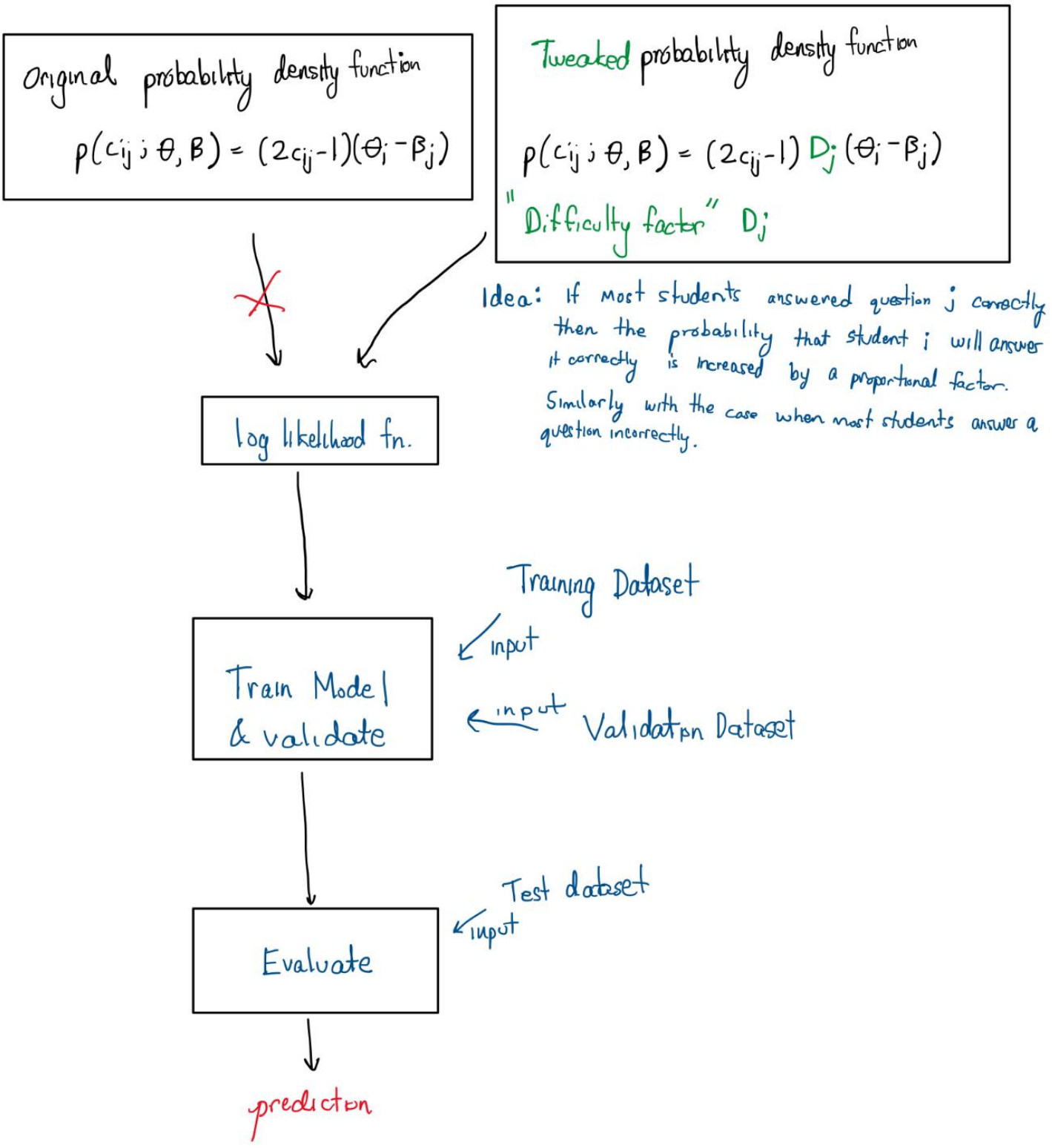
$$= \sum_{j=1}^{Q} (2c_{ij} - 1) D_j - (2c_{ij} - 1) \frac{e^{A_{ij}}}{1 + e^{A_{ij}}} \qquad \text{let } A'_{ij} = (2c_{ij} - 1) D_j$$

$$= \sum_{j=1}^{Q} A'_{ij} - A'_{ij} \sigma(A_{ij}) = \sum_{j=1}^{Q} A'_{ij} (1 - \sigma(A_{ij}))$$

Similarly,

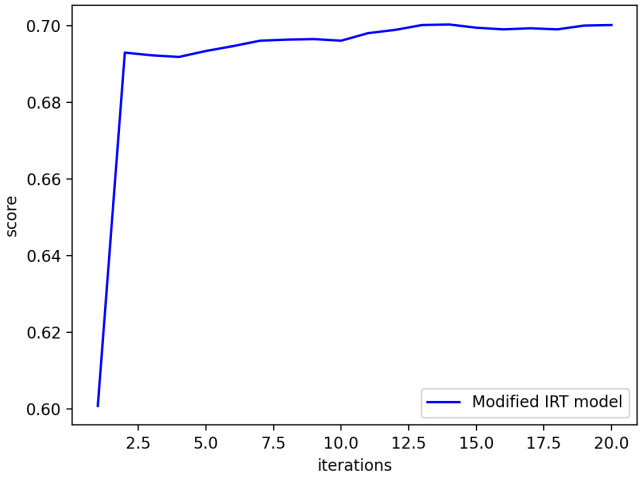
$$\frac{\partial \log(C; \theta, \beta)}{\partial \beta_j} = \sum_{i=1}^{N} -2(c_{ij} - 1) D_j (1 - \sigma(A_{ij}))$$

2. Diagram

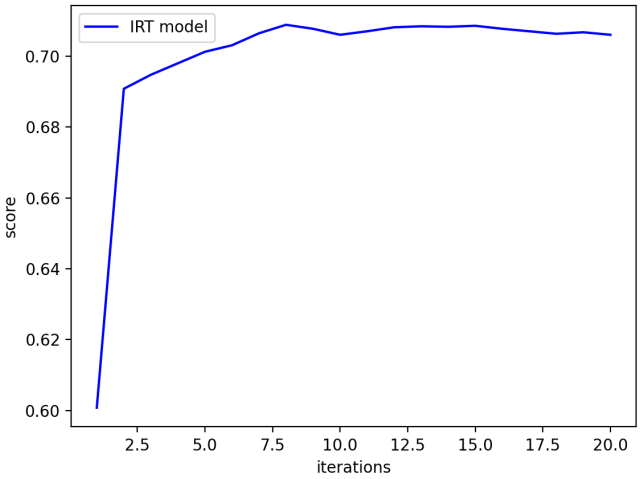

original probability density function
$p(c_{ij}; \theta, B) = (2c_{ij}-1)(\theta_i - \beta_j)$
Tweaked probability density function
$p(c_{ij}; \theta, B) = (2c_{ij}-1)\, D_j\, (\theta_i - \beta_j)$
"Difficulty factor" $D_j$
Idea: If most students answered question j correctly then the probability that student i will answer it correctly is increased by a proportional factor. Similarly with the case when most students answer a question incorrectly.
log likelihood fn.
Training Dataset
input
Train Model & validate
input
Validation Dataset
Test dataset
input
Evaluate
prediction

# 3. Results & Analysis

**IRT model with a difficulty factor**




**Original IRT model (part A, question 2)**




We see a steep jump in the modified model after a few iterations. In contrast, in the original IRT, the steep jump of validation score was noticed after the 7th iterations onwards.
We see a positive linear trend with a gradually decreasing slope similar to the original IRT.
We see that the peak of updated IRT is reached at around the 13th iteration, but still, it could go more with more iteration as it has started decreasing. In contrast, the peak of the original IRT reached around 8-10th iteration. If we run both the updated IRT and original IRT with more iterations, we could see better results from the updated IRT.

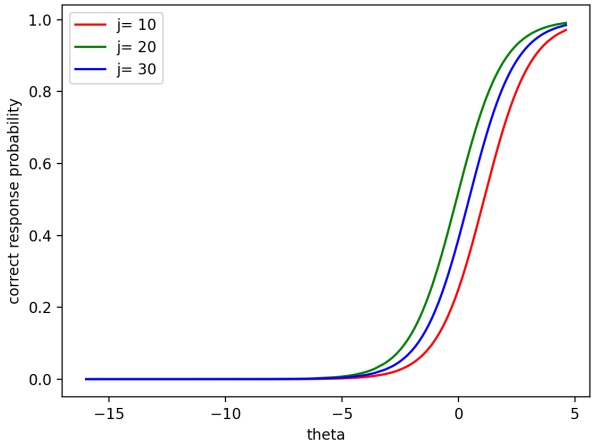
| **Model** | **Validation Accuracy** | **Test Accuracy** |
|---|---|---|
| Original IRT | 0.7060400790290714 | 0.7050522156364663 |
| IRT model w/ difficulty factor | 0.7001128986734406 | 0.6926333615580017 |

The modified IRT model performs very similarly to the original IRT in part A question 2 in terms of the validation accuracy. To make sure that the comparison is accurate we trained both the models with the same learning rate of 0.01. Our goal was to improve the accuracy by tuning the likelihood function with the “difficulty factor.” This new factor increases the likelihood of an easy question being answered correctly and a tricky question being answered incorrectly for an arbitrary student. We expected the modified algorithm to perform better than the original model, but that is not the case as we see. One possible explanation is that the difficulty parameter (beta) already captures the question’s inherent difficulty, making the “difficulty factor” negligible to the performance.

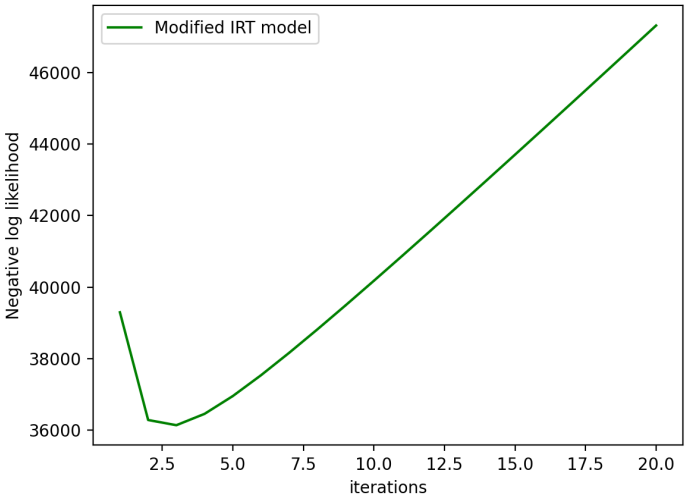
# 4. Limitations

1) Some of the questions were difficult, i.e., only a few students could answer them. This makes the difficulty factor significant. And so, when we are updating theta in the alternating gradient descent, it can make it go in the negatives. This is why I see theta's in the negative in the graph. This is just a byproduct of our implementation.

   We tried to put a checker on the $D_j$ so that value does not get huge. This approach made the accuracy of IRT significantly less.




2) The negative log-likelihood of the function first decreased then increased. If the value of the difficulty factor becomes big, this makes the negative log-likelihood big.




3) We use np.nansum to take the sum of the $j_th$ question. If students left a particular question blank, then the question's sum is small. There is a good chance that the specific question is not tricky, but we assumed it is a difficult question because of the low sum value. We would need a better

# Contributions:

**Aditya Gulati:**

1) Worked on the KNN model and Neural Network.
2) Wrote the formal description and limitations of Part B

**Kavin Singh:**

1) Worked on IRT and Ensemble model.
2) Made the diagram of the Updated IRT model and wrote the result and analysis of the model.