CSC311 Final Report

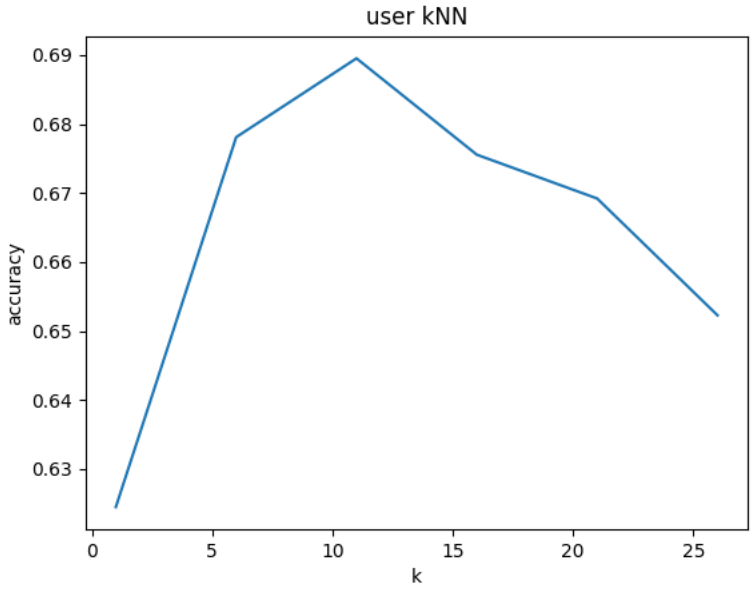
Alfred Mikhael, Tianyi Pan, Jimmy Hao

March 31, 2023

1 kNN

(a)

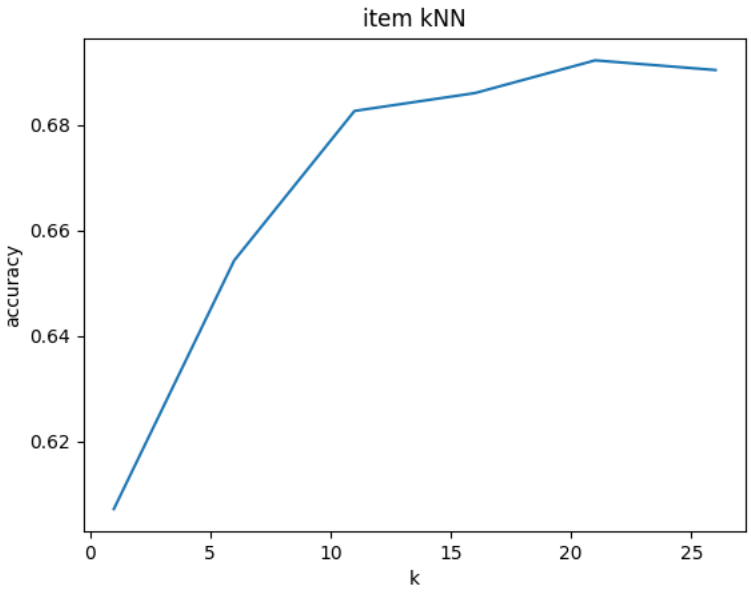
k	1	6	11	16	21	26
accuracy (%)	62.4	67.8	69.0	67.6	66.9	65.2



(b) Choose $k^* = 11$ since it achieves highest validation accuracy. The test accuracy is 68.4%.

(c) Assumption: If two questions are chosen by the same set of students and have the same correctness by each student, they should be the same performance on other students are well.

k	1	6	11	16	21	26
accuracy (%)	60.7	65.4	68.3	68.6	69.2	69.0



$k^* = 21$ achieve maximum validation accuracy and the corresponding test accuracy is 68.2%.

(d) Note that even though kNN by item achieves higher validation accuracy compared to kNN by student, it performs poorly on test accuracy, meaning that it's not generalized well compared to kNN by student. So I believe kNN by student model is better.

(e)

1. Since the dimension of data is large (either from student side or item side), the curse of dimensionality poses a challenge for this classification problem.
2. The training data is a sparse matrix, so it is hard to match with another data point exactly, hence sensitive to noises.
3. The prediction process is costly.
4. The choice of k also might not derive the optimal model, since the cost is large and we have to pick k in a rough sense.

2 IRT

(a) Let $\log L = \log p(\mathbf{C} \mid \boldsymbol{\theta}, \boldsymbol{\beta})$. Given i, j, if $c_{ij} \neq \text{NaN}$, then

if $c_{ij} = 1$,

$$\begin{aligned}\log p(c_{ij} \mid \theta_i, \beta_j) &= \log \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)} \\ &= (\theta_i - \beta_j) - \log(1 + \exp(\theta_i - \beta_j))\end{aligned}$$

if $c_{ij} = 0$,

$$\begin{aligned}\log p(c_{ij} \mid \theta_i, \beta_j) &= \log \frac{1}{1 + \exp(\theta_i - \beta_j)} \\ &= -\log(1 + \exp(\theta_i - \beta_j))\end{aligned}$$

Hence the probability takes following form:

$$\log p(c_{ij} \mid \theta_i, \beta_j) = \begin{cases} c_{ij} \cdot (\theta_i - \beta_j) - \log(1 + \exp(\theta_i - \beta_j)) & \text{if } c_{ij} \neq \text{ NaN} \\ 0 & \text{if } c_{ij} = \text{NaN} \end{cases}$$

Define $D = \{(i, j) \mid c_{ij} \neq \text{NaN}\}$, $D_i = \{j \mid c_{ij} \neq \text{NaN}\}, D_j = \{i \mid c_{ij} \neq \text{NaN}\}$, then

$$\begin{aligned}\log L &= \log \prod_{(i,j)} p(c_{ij} \mid \theta_i, \beta_j) \\ &= \sum_{(i,j)} \log p(c_{ij} \mid \theta_i, \beta_j) \\ &= \sum_{(i,j) \in D} c_{ij} \cdot (\theta_i - \beta_j) - \log(1 + \exp(\theta_i - \beta_j))\end{aligned}$$

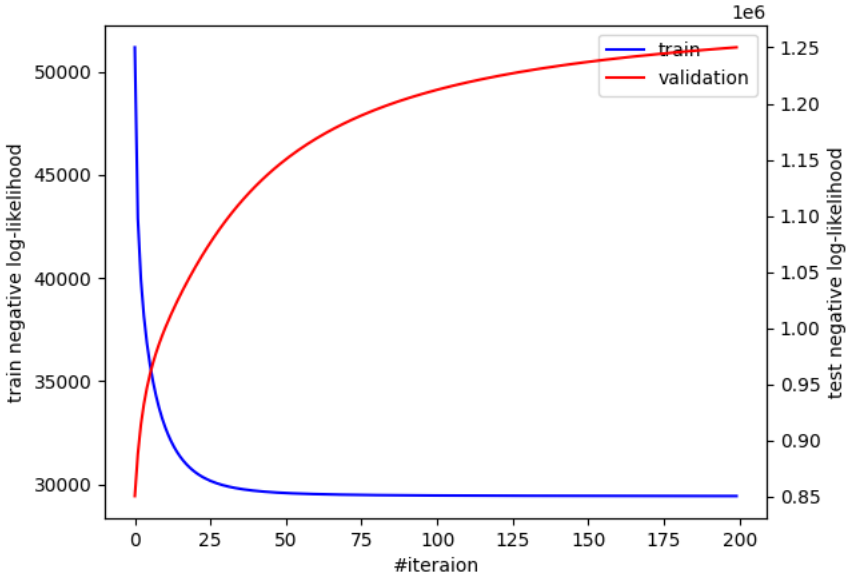
The partial derivatives with respect to θ and β are as follows:

$$\begin{aligned}\frac{\partial \log L}{\partial \theta_i} &= \frac{\partial}{\partial \theta_i} \sum_{j \in D_i} c_{ij} \cdot (\theta_i - \beta_j) - \log(1 + \exp(\theta_i - \beta_j)) \\ &= \left(\sum_{j \in D_i} c_{ij} \right) - \left(\sum_{j \in D_i} \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)} \right)\end{aligned}$$

$$\begin{aligned}\frac{\partial \log L}{\partial \beta_j} &= \frac{\partial}{\partial \beta_j} \sum_{i \in D_j} c_{ij} \cdot (\theta_i - \beta_j) - \log(1 + \exp(\theta_i - \beta_j)) \\ &= -\left(\sum_{i \in D_j} c_{ij} \right) + \left(\sum_{i \in D_j} \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)} \right)\end{aligned}$$

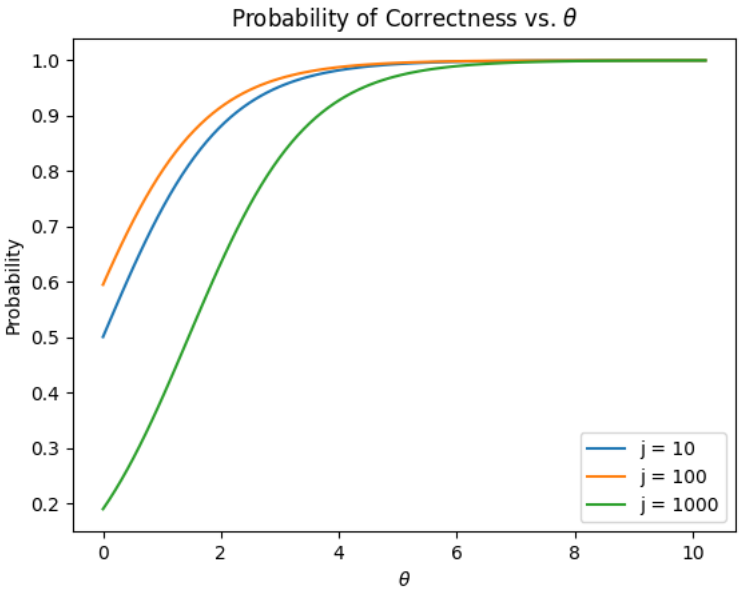
(b) The best hyperparameter are summarized in the following table:

α	iteration
0.01	200



(c) The final validation accuracy is 70.7%, and test accuracy is 70.8%.

(d) I pick $j = [10, 100, 1000]$. The rendering probability plot is as follows:



Each curve is upward sloping because as θ, the student's ability becomes higher, it is more probable that he will get correct. Also note that if one curve is above the other curve, it means that the problem is easier (i.e. smaller β) so it is more probable to get correct given the same student's ability θ.

3 Neural Networks

(a)

1. ALS requires the input data to be in matrix form, while neural networks are more apt to analyze unstructured data like images, text, and audio.
2. ALS is a matrix factorization algorithm while neural networks are a class of algorithms that can be used for more broader classes of tasks, including matrix factorization.
3. ALS can not handle new users and items that are not present in the training data, while neural networks can predict data that were not in the training data.
4. ALS use mean square error as its objective function, while neural network can use many objective functions, including mean square error.

(b) see code
(c) Choose $k^* = 500$, $lr = 0.001$, and $num\_epoch = 50$
(d) Choose $k^* = 500$, the final $valid\_acc = 0.639$, $test\_acc = 0.638$

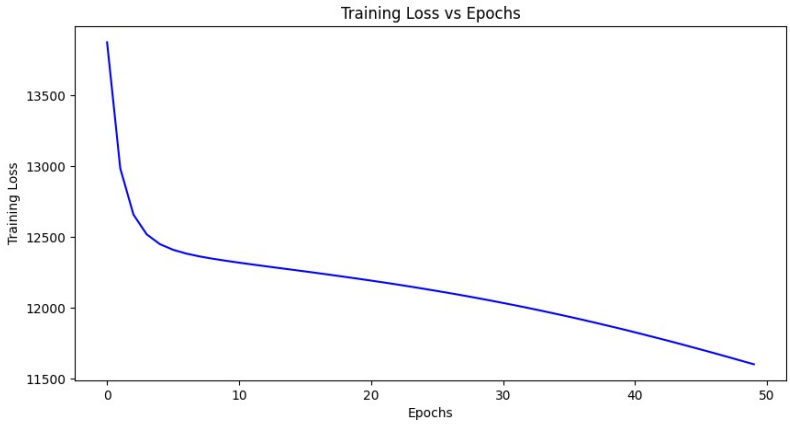


Figure 1: Training Loss Without Regularization

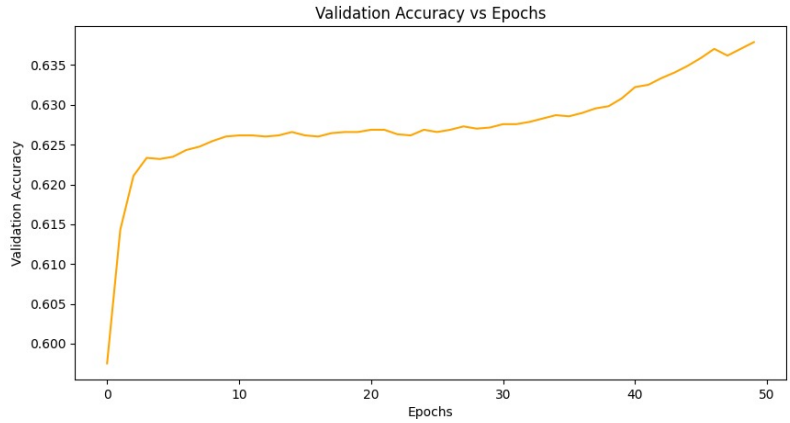


Figure 2: Validation Accuracy Without Regularization

(e) Choose $k^* = 500$, $lr = 0.001$, $num\_epoch = 50$, and $lamb = 0.001$.
The final $valid_acc = 0.638$, $test\_acc = 0.644$. The model performs better with the test data with the regularization penalty but a little worse with the validation data.

4 Ensembling

The ensemble process was implemented as follows:

- Generate N datasets by randomly sampling the training data

- For each algorithm (KNN, IRT, Neural Network), train N / 3 different models, using the same hyperparameters but a different dataset. So overall N / 3 KNN models, N / 3 IRT models, and N / 3 Neural Networks are trained.
- Make a prediction with each model that has been trained
- Take the majority vote of the predictions

The results (cut to 4 decimal places) are displayed below:

	KNN	IRT	NN	Ensemble
Validation Accuracy	0.5719	0.6161	0.6081	0.6217
Test Accuracy	0.5819	0.6138	0.6116	0.6217

Table 1: Validation and test accuracies of the base algorithms and the ensemble

The hyperparameters used were:

- k = 21 for kNN
- learning rate = 0.01 and 200 iterations for IRT
- $k^* = 250$, learning rate = 0.002, $\lambda = 0.001$, and 30 epochs for the neural networks
- N = 9

We notice that the ensemble accuracy is lower than all of the three models trained individually. But since the bagging process (theoretically) decreases the variance of the ensemble by a factor of $\frac{1}{N}$, there are more dominant reasons that can explain for this decline.

For kNN, sampling with replacement cannot cover all students, and will have repeated data point. This will increase bias and making the current `evaluate` function invalid since it is no longer able to keep track of each student correctly. But if we sample without replacement, then it is the same as without sampling at all! The same reasons also applies to IRT and neural network, since we don't know which θ correspond to which students in the validation accuracy evaluation, and which row is which student in NN. However, by running the code, we can see that the ensemble performance did increase given the same data set.

5 Part 2

5.1 Formal Description

We note that the original IRT model gives closer training, validation and test accuracy (74.0%, 70.7% and 70.8%), so we believe the model is underfitting, and

adding more complexity (i.e. more parameter to the model) will gain better performance on prediction. We add our third parameter **a**, guessing, which measures the student's probability of guessing the correct response (which is a kind of ability). We referenced the following article for the new probability of correctness (https://assess.com/three-parameter-irt-3pl-model/).

Such assumption is reasonable: Since diagnostic questions are multiple-choice questions, there is a chance that a student can guess the answer without understanding the concept. We will apply gradient descent on guessing parameter **a**, θ, and β during the training process to maximize the log-likelihood.

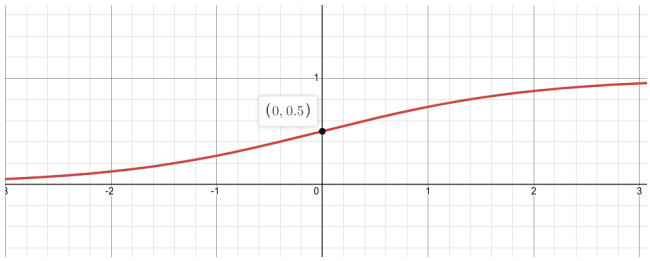


Figure 3: A sample curve of the 2-parameter model, with beta = 0. The average student has a 50% chance of guessing the correct answer and the minimum probability to get the correct answer is 0

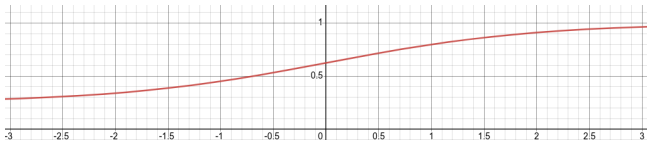


Figure 4: A sample curve of the 3 parameter model, with beta = 0 and guessing parameter = 0.25. Note how the minimum probability of answering correctly is now 0.25 rather than 0

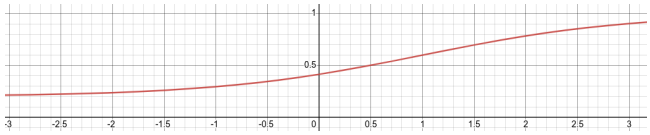


Figure 5: A sample curve of the 3 parameter model, with beta = 1 and guessing parameter = 0.20. It is skewed left relative to the previous curve.

5.1.1 Gradient calculations for 3 parameter model

The new probability of correctness is defined as

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

Now taking log on both sides gives

$$\begin{aligned}\log(p(c_{ij} = 1|a_i, \theta_i, \beta_j)) &= \log\left(a_i + (1 - a_i)\frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)}\right) \\ &= \log(a_i \cdot (1 + \exp(\theta_i - \beta_j)) + (1 - a_i) \cdot \exp(\theta_i - \beta_j)) \\ &- \log(1 + \exp(\theta_i - \beta_j)) \\ &= \log(a_i + \exp(\theta_i - \beta_j)) - \log(1 + \exp(\theta_i - \beta_j))\end{aligned}$$

$$\begin{aligned}\log(p(c_{ij} = 0|a_i, \theta_i, \beta_j)) &= \log\left(1 - a_i + (a_i - 1)\frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)}\right) \\ &= \log((1 - a_i) \cdot (1 + \exp(\theta_i - \beta_j)) + (a_i - 1) \cdot exp(\theta_i - \beta_j)) \\ &- \log(1 + \exp(\theta_i - \beta_j)) \\ &= \log(1 - a_i) - \log(1 + \exp(\theta_i - \beta_j))\end{aligned}$$

Now we can calculate $\log L = \log(p(C|\mathbf{a}, \theta, \beta))$ as

$$\begin{aligned}\log L &= \log\left(\prod_{i,j} p(c_{ij} = 1|a_i, \theta_i, \beta_j)^{c_{ij}} p(c_{ij} = 0|a_i, \theta_i, \beta_j)^{1-c_{ij}}\right) \\ &= \sum_{i,j} c_{ij} \log(p(c_{ij} = 1|a_i, \theta_i, \beta_j)) + (1 - c_{ij}) \log(p(c_{ij} = 0|a_i, \theta_i, \beta_j)) \\ &= \sum_{i,j} c_{ij} \cdot (\log(a_i + \exp(\theta_i - \beta_j)) - \log(1 + \exp(\theta_i - \beta_j))) \\ &+ (1 - c_{ij}) \cdot (\log(1 - a_i) - \log(1 + \exp(\theta_i - \beta_j))) \\ &= \sum_{i,j} c_{ij} \cdot \log(a_i + \exp(\theta_i - \beta_j)) + (1 - c_{ij}) \cdot \log(1 - a_i) - \log(1 + \exp(\theta_i - \beta_j))\end{aligned}$$

Now we can calculate the gradient with respect to each of the parameters.

$$\frac{\partial}{\partial \theta_i} \log(p(C|\mathbf{a}, \theta, \beta)) = \sum_j \left(\frac{c_{ij} \exp(\theta_i - \beta_j)}{a_i + \exp(\theta_i - \beta_j)} - \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)}\right)$$

$$\frac{\partial}{\partial \beta_j} \log(p(C|\mathbf{a}, \theta, \beta)) = \sum_i \left(\frac{-c_{ij} \exp(\theta_i - \beta_j)}{a_i + \exp(\theta_i - \beta_j)} + \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)}\right)$$

$$\frac{\partial}{\partial a_i} \log(p(C|\mathbf{a}, \theta, \beta)) = \sum_j \left(\frac{c_{ij}}{a_i + \exp(\theta_i - \beta_j)} - \frac{1 - c_{ij}}{1 - a_i}\right)$$

5.1.2 Algorithm

Algorithm

$\theta \leftarrow$ list of Gaussian random number between 0 and 1
$\beta \leftarrow$ list of Gaussian random number between 0 and 1
$\mathbf{a} \leftarrow$ list of 0's
γ, γ_a learning rate
for $i = 1, \ldots, 100$ **do**
    $\theta \leftarrow \theta + \gamma \frac{\partial J}{\partial \theta}$
    $\beta \leftarrow \beta + \gamma \frac{\partial J}{\partial \beta}$
    if $i \geq 30$ **then**
        $\mathbf{a} \leftarrow \mathbf{a} + \gamma_a \frac{\partial J}{\partial \mathbf{a}}$

5.2 Figure or Diagram

Our analysis in previous part should be a clear and full illustration of our idea, and the modification is hard to describe using figures or diagrams, so we decide not to put those in this section.

5.3 Comparison or Demonstration

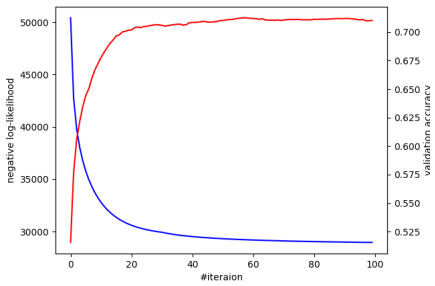


Figure 6: Training loss and validation accuracy of the 3-parameter model. Red: Validation Accuracy, Blue: Training Loss

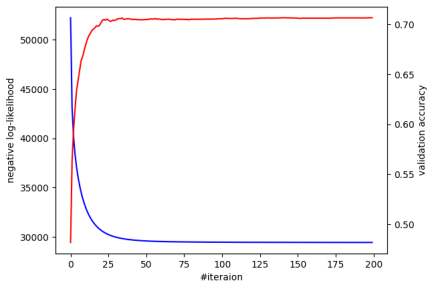


Figure 7: Training loss and validation accuracy of the 2-parameter model. Red: Validation Accuracy, Blue: Training Loss

The new training, validation and test accuracy are 74.4%, 71.0%, and 71.1%, which goes up by 0.3% compared to the original IRT model. In the following paragraphs, we will explain (1) why the model improves, and (2) why the model only improves slightly.

First, since we are adding more parameters, the hypothesis space is larger (setting $a_i = 0$ will return to the original model). Therefore, under proper optimiza-

tion procedure, the training loss should be lower, given the same loss function. Another possible reason for the improved performance is that to restrict the behavior of a_i, we decide to first iterate 30 times for θ and β, making them relatively stable, then start updating $\mathbf{a}$ as well. We also set a lower learning rate for $\mathbf{a}$ and initialize it to all zeros instead of Gaussian distribution. If not (this is by trial and error), the performance would drop to 60%. One can refer to Limitations for more details.

However, the results are far from desirable. First, one may note that some a_i is becoming negative (even as low as -0.75). This is because the gradient descent update want to "penalize" those who always get wrong and "award" those who gets correct, making a_i negative. But the assumption of a should be a probability, which is bounded between 0 and 1. For further attempts we take, please refer to Limitations.

5.4 Limitations

First, we encounter serious problems in optimization. Because the probability is a linear function $\mathbf{a}$ with respect to a, so its value become unreasonably large in magnitude if setting the same learning rate. We also update a_i's after some initial iteration to make θ and β stabilize to achieve an interpretable result. We have tried to 3 methods to solve this issue:

- initialize in standard normal distribution or uniform $[-1, 1]$ distribution;
- replace $\mathbf{a}$ by $\eta \cdot \sigma(\mathbf{a})$ so as to control the range of $\mathbf{a}$ by η;
- in each parameter update, replace a_i's that exceeds some threshold to the threshold.

But none of them gives statisfactory result. A method to solve this optimization problem is to add more iterations (the update is too slow to wait), and restrict the range of $\mathbf{a}$ to make the problem convex (if not now).

Second, In this modification, we still see the main problem, underfitting is not solved: the accuracy gap between train and test are still small. So a possible modification would be add more parameter on the exponential part of the probability of correctness, or using other metric like AUC and MSE.

Third, We also expect that the model will perform poorly on free-response questions and questions that do not have a clear-cut binary answer. For free response questions, the chance of guessing correctly becomes very low, so the guessing parameter becomes obsolete, and it would be better to use the 2-parameter model in order to reduce computational load during gradient descent.

One extension would be to build a generative model and assign partial credits based on the grade distribution of the question instead of making binary classification.

6 Contribution

Alfred Mikhael: Part A (d), Part B report and testing.
Tianyi Pan: Part A (a), (b), Part B code.
Jimmy Hao: Part A (c), Part B report and testing.