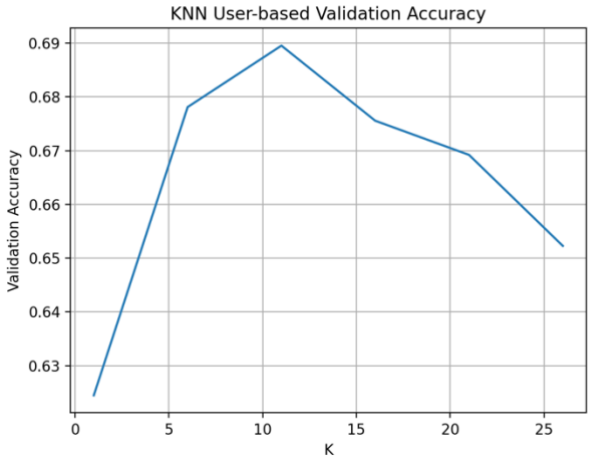
Part A

1. KNN

(a)



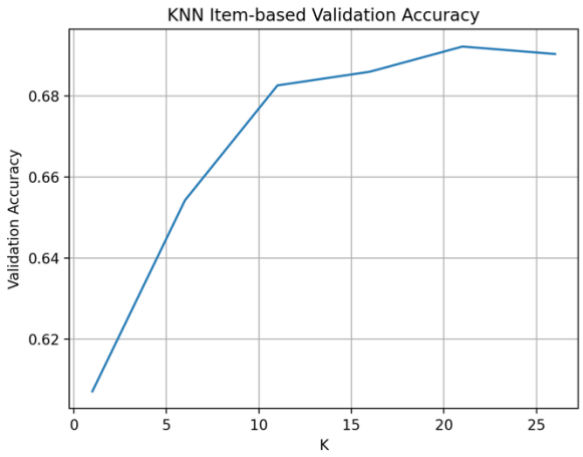

```
Validation Accuracy: 0.6244707874682472
Validation Accuracy: 0.6780976573525261
Validation Accuracy: 0.6895286480383855
Validation Accuracy: 0.6755574372001129
Validation Accuracy: 0.6692068868190799
Validation Accuracy: 0.6522720858029918
```

(b)

```
k* = 11, with test accuracy 0.6841659610499576.
```

(c)

Assumption: If question A and B gets same correctness from other users, then question A's correctness for a specific user is the same as question B's correctness for that user.




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

```
k* = 21, with test accuracy 0.6816257408975445.
```

(d)

Test performance of user-based collaborative filtering is slightly better than item-based collaborative filtering.

(e)

Limitations:

- User-based collaborative filtering cannot find users with similar knowledge (who has similar answers to other questions) accurately when the user is new to the platform (does not have many questions answered). Similarly for item-based collaborative filtering in the case that some questions only get a few answers.
- The test time computation cost is high since we need to calculate distance for each test sample.

2. IRT

2. (a) Assume C is $N \times D$

$$P(C \mid \theta, \beta) = \prod_{i,j} \left[\sigma(\theta_i - \beta_j)\right]^{c_{ij}} \left[1 - \sigma(\theta_i - \beta_j)\right]^{1 - c_{ij}}$$

$$\log P(C \mid \theta, \beta) = \sum_{i,j} c_{ij} \log\left[\sigma(\theta_i - \beta_j)\right] + (1 - c_{ij}) \log\left[1 - \sigma(\theta_i - \beta_j)\right]$$

$$= \sum_{i,j} c_{ij}\left[(\theta_i - \beta_j) - \log(1 + e^{\theta_i - \beta_j})\right] - (1 - c_{ij}) \log(1 + e^{\theta_i - \beta_j})$$

$$= \sum_{i,j} c_{ij}(\theta_i - \beta_j) - \log(1 + e^{\theta_i - \beta_j})$$

$$\frac{\partial \log P(C \mid \theta, \beta)}{\partial \theta_i} = \sum_{j=1}^{D} \left(c_{ij} - \frac{e^{\theta_i - \beta_j}}{1 + e^{\theta_i - \beta_j}}\right)$$

$$\frac{\partial \log P(C \mid \theta, \beta)}{\partial \beta_j} = \sum_{i=1}^{N} \left(-c_{ij} + \frac{e^{\theta_i - \beta_j}}{1 + e^{\theta_i - \beta_j}}\right)$$

(b)

```
# iterations: 10
Train_NLLK: 31232.652972957872   Train_Score: 0.7263971210838273   Val_NLLK: 4015.5689713346444   Val_Score: 0.7060400790298714
# iterations: 15
Train_NLLK: 30568.2361425465528   Train_Score: 0.7327123986294101   Val_NLLK: 3966.1438771180287   Val_Score: 0.7085802991814846
# iterations: 25
Train_NLLK: 29959.02322125384   Train_Score: 0.7366991250352808   Val_NLLK: 3937.00010203143714   Val_Score: 0.7064634490544736
# iterations: 40
Train_NLLK: 29653.35963237899   Train_Score: 0.7391158622636184   Val_NLLK: 3937.14559979269294   Val_Score: 0.7056167090036692
# iterations: 60
Train_NLLK: 29528.2570883555258   Train_Score: 0.7392922664408693   Val_NLLK: 3946.875341282798   Val_Score: 0.7063223257126728
# iterations: 85
Train_NLLK: 29478.401184274746   Train_Score: 0.7399096810612475   Val_NLLK: 3955.30876853082O5   Val_Score: 0.7056167090036692
```

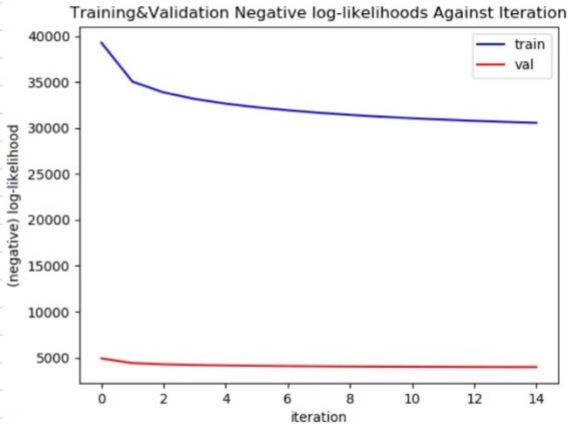
```
Learning Rate: 0.1
Train_NLLK: 82328.12147659516   Train_Score: 0.6938858312164832   Val_NLLK: 10933.159293081908   Val_Score: 0.6685012701100762
Learning Rate: 0.05
Train_NLLK: 45628.63880671573   Train_Score: 0.7103090601185436   Val_NLLK: 6104.693889162354   Val_Score: 0.6850127011007621
Learning Rate: 0.01
Train_NLLK: 30185.23613056377   Train_Score: 0.7352173299463731   Val_NLLK: 3945.26498941895   Val_Score: 0.7060400790298714
Learning Rate: 0.005
Train_NLLK: 31180.363211819552   Train_Score: 0.726926333361558   Val_NLLK: 4011.6146184657O3   Val_Score: 0.7067456957380751
Learning Rate: 0.001
Train_NLLK: 34271.803838984144   Train_Score: 0.7057401919277448   Val_NLLK: 4319.240897169442   Val_Score: 0.6951735817104149
```

we tries a set hyperparameters, and select

$$\begin{cases} \text{learning rate} = 0.01 \\ \text{number of iteration} = 15 \end{cases}$$

As the best hyperparameters.

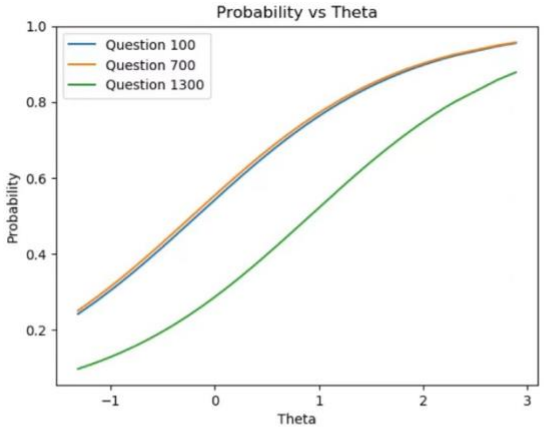
training curve:



(c) `Final Validation Acc: 0.7077335591306803 Test Acc: 0.703076488851256`

final validation & test are 70.77% & 70.31% respectively

(d)



The shape of all curves are sigmoid-like. They have similar "slope".

These curves show that given the difficulty of a question (β), the stronger a student's ability (θ) is, the higher the propability that the student will answer the question correctly. The righter the curve shifts, the higher its difficulty.

3. Matrix Factorization

3. (a)

```
k: 1    acc: 0.6428168219023427
k: 2    acc: 0.6579170194750211
k: 4    acc: 0.6577758961332204
k: 8    acc: 0.6603161162856337
k: 16   acc: 0.6558001693480101
```

the best $k = 8$

```
final val acc: 0.6603161162856337
final test acc: 0.65961049957663
```

final validation accuracy is 66.03%
final test accuracy is 65.96%

(b) Since we can't perform SVD on sparse matrix, we filled the missing values (NaN) with the average on the current item (question). This could cause significant inaccuracy in some cases. Imagine a question that is only answered by one student and he/she answered correctly.

	q_0
n_0	NaN
n_1	NaN
n_2	NaN
n_3	1
n_4	NaN
n_5	NaN

The average to be filled is 1, but it can happens that this question is extremely difficult, so the value to be filled should be close to 0.
However, our SVD ignores this kind of issue, thus the resulting latent factor model is inaccurate.

(c) (d)

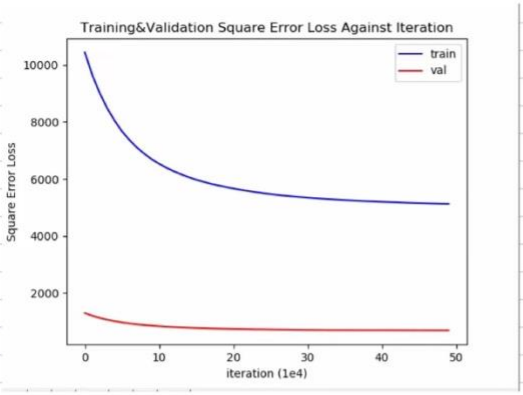
```
# iterations: 50000 acc: 0.5108664973186565
# iterations: 100000    acc: 0.5983629692351115
# iterations: 500000    acc: 0.7003951453570421
# iterations: 1000000   acc: 0.7006773920406435
# iterations: 5000000   acc: 0.6816257408975445
```

chosen learning rate: 0.01
chosen # iterations: 500000
$k^* = 4$

```
lr: 0.1 acc: 0.6618684730454417
lr: 0.05    acc: 0.6867061812023709
lr: 0.01    acc: 0.7033587355348575
lr: 0.005   acc: 0.6826136042901496
lr: 0.001   acc: 0.5009878633926051
```

```
k: 1    acc: 0.7009596387242449
k: 2    acc: 0.6989839119390348
k: 4    acc: 0.7053344623200677
k: 8    acc: 0.7027942421676545
k: 16   acc: 0.7011007620660458
```

(e)




```
final val acc: 0.7006773920406435
final test acc: 0.6923511148744003
```

final validation & test accuracy are 70.07% & 69.24% respectively.

4. Ensemble

Selected models:
3 item response models with learning rate = 0.01 and iterations = 15.

Ensemble process:
- Generate 3 resamples from the training set.
- Train the 3 models using different resamples.
- For each IRT model, get the sigmoid function output for training set, validation set and test set.
- For each dataset, average over the sigmoid function output from the 3 models; use threshold = 0.5 to get the final prediction.
- Calculate accuracy for each dataset.

Result:

Model	Validation accuracy	Test accuracy
IRT model (Data = training set, learning rate = 0.01, iterations = 15	0.7077	0.7031
Ensemble model (as descripted above)	0.7032	0.6977

Compared to the IRT model with original training set and the same learning rate and iteration, our ensemble model has a worse performance.

Analysis:
- Since each base model take a dataset that may contain repetitive data, it puts more weight on the data that repeats more times and affects our gradient decent process. These weights are put randomly and may affect our base models' performance.
- Since the times of resampling is small, it limits the upper bounds of the amount of the variance that bagging can reduce. When we increase the times of resampling, our ensemble model performs better.

Model	Validation accuracy	Test accuracy
IRT model (Data = training set, learning rate = 0.01, iterations = 15	0.7077	0.7031
Ensemble model (m = 3)	0.7032	0.6977
Ensemble model (m = 20)	0.7083	0.7017
Ensemble model (m = 40)	0.7086	0.7031

Part B
3-PL IRF (Idea from Columbia Public Health. See Reference)

Yiyi Zhang 1006298962
Tao Zhang 1006057696

Introduction

We will show our approach of improving our IRT model in this section. The IRT model we used in Q2 has two sets of parameters that represent students' ability and questions' difficulty. In this new version, we implement the 3 – Parameter logistic model and introduce to our model a new set of parameters for discrimination of each question, and a guessing parameter.

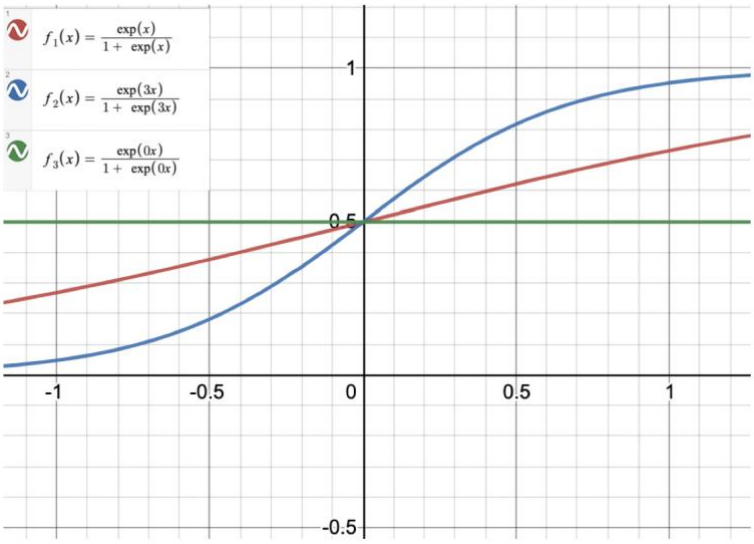
Proposed Method - Discrimination Parameter

In Q2, we used sigmoid functions with the same slope to predict the probability for a student with ability θ_i to give the correct answer for a question with difficulty β_j, no matter what kinds of question we are considering. This means that if the difference between the student's ability and the question's difficulty is the same (i.e., $\theta_i - \beta_j$ are the same), the model always gives the same prediction for correctness.

This is a problem because questions may have different degrees of sensitivity to the difference between a student's ability and a question's difficulty. A question can be insensitive (i.e., with low discrimination). For example, for a multiple-choice question with no description and two options, students with any ability have a 50 percent chance of giving the right answer. For questions that are more discriminative than others, a student with higher ability than the question's difficulty has a better chance of getting the right answer, and a student with lower ability than the question's difficulty has a lower chance of getting the right answer; both cases compared to questions that are less discriminative.

This characteristic of questions can be captured using a set of discrimination parameter α, one for each question. A larger value of α can be used to represent a question that is more discriminative. As shown in the following figure (Figure 1).

$$p_{i,j}\left(\theta_{i,}\beta_{j,}\alpha_{j}\right) = \frac{\exp\left(\alpha_j\left(\theta_i - \beta_j\right)\right)}{1 + \exp\left(\alpha_j\left(\theta_i - \beta_j\right)\right)}$$



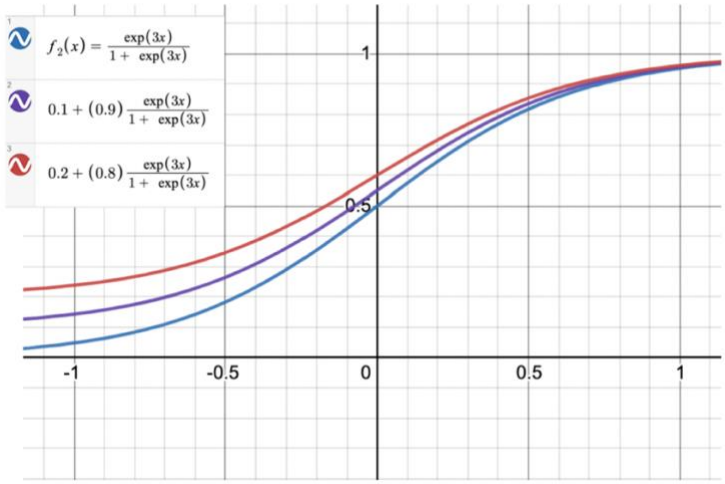
(Figure 1)

Proposed Method - Guessing Parameter

Another thing to notice is that despite how large the gap between the student's ability and the question's difficulty is, the probability for the student to the correct answer should not be close to 0. A student can always guess the answer and has a certain probability of getting the right answer.

We can introduce a guessing parameter γ to our model. A larger γ gives a higher chance for students to guess the correct answer.

$$p_{i,j}\left(\theta_{i,}\beta_{j,}\alpha_{j,}\gamma\right) = \gamma + (1-\gamma)\frac{\exp\left(\alpha_j\left(\theta_i-\beta_j\right)\right)}{1+\exp\left(\alpha_j\left(\theta_i-\beta_j\right)\right)}$$



(Figure 2)

Implementation of the Methods (Formal Description)

Most of our implementation stays the same. We changed our formula for the possibility of getting the correct answer by including more given parameters. Then, we calculate the partial derivative for each parameter, and apply gradient descent to get the optimal parameters.

The following is the detailed calculation.

PART B Assume C is $N \times D$

3-PL:

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

$$P(c_{ij}=0 \mid \theta_i, \gamma_i, \beta_j, \alpha_j) = \frac{1-\gamma_i}{1+exp(\alpha_j(\theta_i-\beta_j))}$$

$$P(C \mid \theta, \gamma, \beta, \alpha) = \prod_{i,j}\left[\frac{\gamma_i + exp(\alpha_j(\theta_i-\beta_j))}{1+exp(\alpha_j(\theta_i-\beta_j))}\right]^{c_{ij}} \cdot \left[\frac{1-\gamma_i}{1+exp(\alpha_j(\theta_i-\beta_j))}\right]^{1-c_{ij}}$$

$$\log P(C \mid \theta, \gamma, \beta, \alpha) = \sum_{i,j} c_{ij}\log(\gamma_i + exp(\alpha_j(\theta_i-\beta_j))) - \log(1+exp(\alpha_j(\theta_i-\beta_j))) + (1-c_{ij})\log(1-\gamma_i)$$

$$\frac{\partial \log P(C \mid \theta, \gamma, \beta, \alpha)}{\partial \theta_i} = \sum_{j=1}^{D} \frac{\alpha_j c_{ij} \cdot exp(\alpha_j(\theta_i-\beta_j))}{\gamma_i + exp(\alpha_j(\theta_i-\beta_j))} - \frac{\alpha_j exp(\alpha_j(\theta_i-\beta_j))}{1+exp(\alpha_j(\theta_i-\beta_j))}$$

$$\frac{\partial \log P(C \mid \theta, \gamma, \beta, \alpha)}{\partial \gamma_i} = \sum_{j=1}^{D} \frac{c_{ij}}{\gamma_i + exp(\alpha_j(\theta_i-\beta_j))} - \frac{1-c_{ij}}{1-\gamma_i}$$

$$\frac{\partial \log P(C \mid \theta, \gamma, \beta, \alpha)}{\partial \beta_j} = \sum_{i=1}^{N} \frac{-\alpha_j c_{ij} \cdot exp(\alpha_j(\theta_i-\beta_j))}{\gamma_i + exp(\alpha_j(\theta_i-\beta_j))} + \frac{\alpha_j exp(\alpha_j(\theta_i-\beta_j))}{1+exp(\alpha_j(\theta_i-\beta_j))}$$

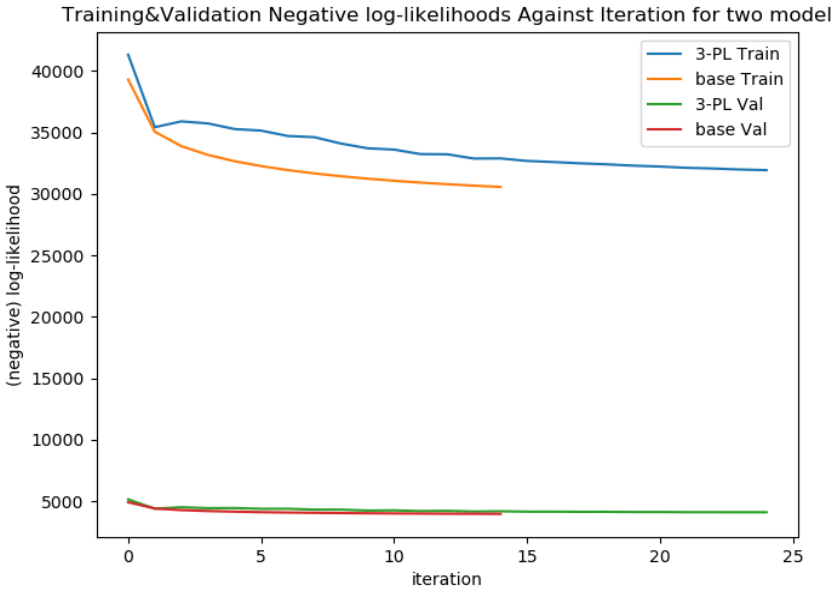
$$\frac{\partial \log P(C \mid \theta, \gamma, \beta, \alpha)}{\partial \alpha_j} = \sum_{i=1}^{N} \frac{(\theta_i-\beta_j) c_{ij} \cdot exp(\alpha_j(\theta_i-\beta_j))}{\gamma_i + exp(\alpha_j(\theta_i-\beta_j))} - \frac{(\theta_i-\beta_j) exp(\alpha_j(\theta_i-\beta_j))}{1+exp(\alpha_j(\theta_i-\beta_j))}$$

Comparison & Demonstration

Model	Validation Accuracy	Test Accuracy
IRT	**70.77%**	**70.31%**
3-PL IRT	**70.17%**	**69.04%**
kNN	**68.95% (user-based) / 69.22% (item-based)**	**68.42% (user-based) / 68.16 (item-based)**
Matrix Factorization	**66.03%**	**65.96%**
Ensemble (IRT)	**70.32%**	**69.77%**

In general, our IRT based model (IRT, 3-PL IRT, ensemble (IRT)) has better performance (higher validation and higher test accuracy) than the kNN model and the matrix factorization model.

Our 3-PL IRT model introduces discrimination parameter and guessing parameter to try to improve our IRT model; however, the performance does not improve, the validation accuracy and the test accuracy are both lower than our IRT model in Q2. In addition, compared our 3-PL IRT model to another attempt of improving our IRT model in Q4 that uses bagging ensemble, the performance for our 3-PL IRT model is also worse than the ensemble (IRT) model. Possible reasons for this result are shown in the limitation section.



(Plot 1)

Comparing Training Process:

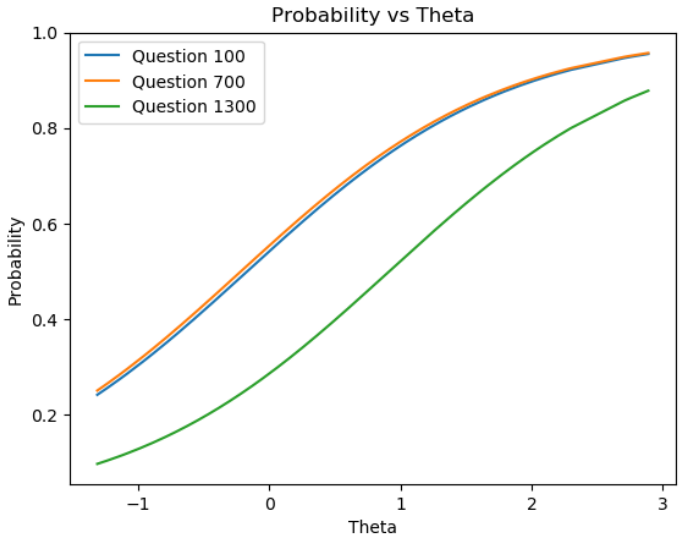
Plot 1 shows how training & validation negative log-likelihoods for two model changes during the training process. We can see from the plot that for the validation set, two models behave almost identical. But for the training set, 3-PL IRF converges more slowly than base IRF, and the final log-likelihood is outperformed by base IRF. We can infer from it that 3-PL IRF experiences less overfitting issues than base IRF. And this is quite surprising! Note that 3-PL IRF has more parameters than base IRF. We infer that it is because of the way we update our parameters: in each update process, we give two interval constraints on gamma and alpha respectively (see Figure 3). The reason for gamma's constraint is because it is an asymptote for the probability, or it cannot be out of (0, 1). Why we have alpha's constraint is because of two reasons, one is because the prior experience concluded by researchers that even though alpha can go from negative infinity to positive infinity, it only makes sense to be in (0, 2) in practice. The second reason is that big alpha will cause np.exp() overflow, so for "safety", we limit it to this interval. Thus, we can think that these two constraints work just like regularization, so at least gamma and alpha will not cause overfitting to the training set.

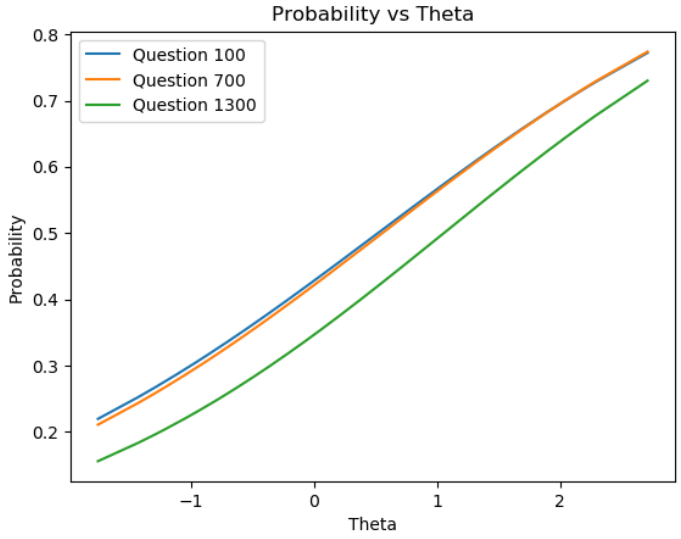
```
gamma[gamma >= 1] = 0.99
gamma[gamma <= 0] = 0.01
alpha[alpha <= 0] = 0.01
alpha[alpha >= 2] = 1.99
```

(Figure 3)

Comparing Question Curves:

As we did in Part A 2(d), we do the same thing for 3-PL IRF with the new discrimination parameter α involves. (See Plot 2) The upper figure is for base IRF, the bottom figure is for 3-PL IRF. We can see that the two results are similar. One slight difference is the slopes of question 100 and question 700. In the base IRF, the two questions' slopes are identical, but in 3-PL IRF, the two questions' slopes are different. This is because of α, which indicates the discrimination ability of each question, it makes the question curve more realistic.


Probability vs Theta
Question 100
Question 700
Question 1300
Probability
Theta
1.0
0.8
0.6
0.4
0.2
−1
0
1
2
3


Probability vs Theta
Question 100
Question 700
Question 1300
Probability
Theta
0.8
0.7
0.6
0.5
0.4
0.3
0.2
−1
0
1
2

(Plot 2)

Limitations

1. Naive Bayes:
   Since we use naive bayes to construct our model, we assume that each data point is independent of each other. But in reality, this assumption usually does not hold. Imagine there is a set of extremely hard questions of similar pattern. If we assume that they are independent, then students should have a high probability of answering them incorrectly without distinction. But in practice, once the student figures out how to solve one of them correctly, the rest of them should be also solved correctly with ease. Even though naive bayes will give us inaccuracy, it is hard to extend it. The only ways to improve it may be using a more advanced probabilistic model or change to a discriminative model.

2. Data sparsity:
   When our training data is relatively small, some patterns in our training data may not represent general characteristics of samples well. For example, we may have a small training set containing a large percentage of questions that are easy to guess, so our γ parameter may be set to a large value, which may not apply to our testing set. A way to improve in this situation is to introduce a prior that provides some base knowledge for our study domain.

3. Upper bound:
   By introducing the guessing parameter γ， we give the prediction probability a lower bound. But we did not give it an upper bound. Consider the sigmoid function: when the student's ability is big to a certain extent, his/her probability of answering a question correctly so close to 100% that it can be treated as 100%. But imagine two cases: Case 1: a student has an extremely strong ability on answering any question correctly, but he/she just deliberately wants to choose the incorrect answer or randomly choose an answer, then of course his/her will not get everything correct. The extension to resolving this limitation is just introducing an upper bound parameter.

4. Training speed:
   We observed that the training time increased significantly. Since we introduced D + 1 (number of questions + 1) parameters, the computation for the partial derivatives for the original parameters becomes more complicated, and we need to optimize a lot more parameters using gradient descent. Since we are using gradient descent, one way we can improve the training speed is to set a smaller iteration number, since we may have already converged before we stopped iterating. Another way is using mini-batch that apply gradient descent based on several training points instead of the whole training set. We may also find relations between parameters to further simplify the set of parameters needed.

Reference

Item Response Theory, Population Health Methods, Columbia University Mailman School of Public Health. https://www.publichealth.columbia.edu/research/population-health-methods/item-response-theory