CSC311 Final Project

Songheng Yin, Haining Tan

December 2020

1 Part A

1. k-Nearest Neighbour

   (a) Accuracy on the validation data:

   k = 1: 0.6244707874682472

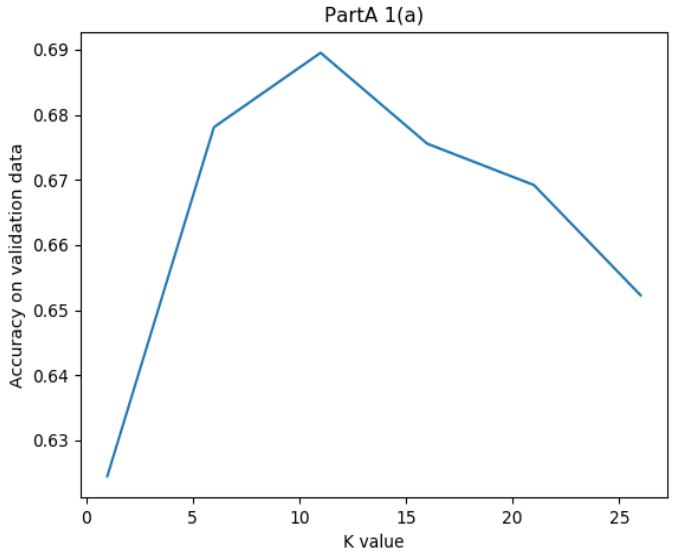
   k = 6: 0.6780976573525261

   k = 11: 0.6895286480383855

   k = 16: 0.6755574372001129

   k = 21: 0.6692068868190799

   k = 26: 0.6522720858029918



   (b)

   User-based: Chosen k = 11

   Final test accuracy is 0.6841659610499576.

(c)

Assumption on item-based collaborative filtering:

If two questions A and B are answered the same (correctly or incorrectly) by other students, then for a specific student, his correctness on these two questions would match.

Accuracy on the validation data:

k = 1: 0.607112616426757

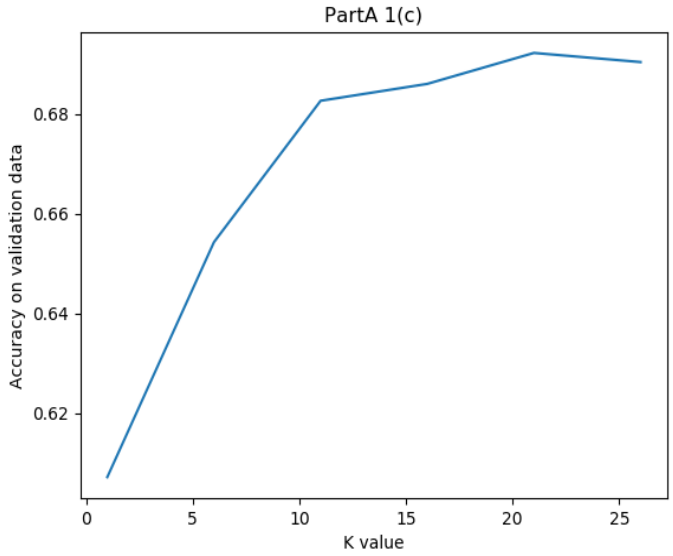
k = 6: 0.6542478125882021

k = 11: 0.6826136042901496

k = 16: 0.6860005644933672

k = 21: 0.6922099915325995

k = 26: 0.69037538808919



Item-based: Chosen k = 21

Final test accuracy is 0.6683601467682755.

(d)

Test performances:

User-based test accuracy: 0.6841659610499576

Item-based test accuracy: 0.6683601467682755

User-based collaborative filtering performs better.

(e) Limitations:

1. Since the number of students and the number of questions are large (542 and 1774), then the dimension of the input data to KNN is high. This will lead to the curse of dimensionality. That is, most data points are approximately the same distance, so the predictions made by KNN might be inaccurate. Also, it will takes a long time to predict.

2. The input data is a sparse matrix, so the questions answered by two students could be different and the groups of students that answer two questions could be different as well. Thus, the metric computed by KNN might not reflect the similarity between two students or questions correctly.

3. There are too many missing data, which KNN is sensitive to.

2. Item Response Theory:

(a) log-likelihood:

Number of students $N = 542$

Number of questions $M = 1774$

$$\begin{aligned}
\log p(\mathbf{C} \mid \boldsymbol{\theta}, \boldsymbol{\beta}) &= \log \prod_{i=1}^{N} \prod_{j=1}^{M} [(\frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)})^{\mathbb{I}(c_{ij}=1)} (\frac{1}{1 + \exp(\theta_i - \beta_j)})^{\mathbb{I}(c_{ij}=0)}] \\
&= \sum_{i=1}^{N} \sum_{j=1}^{M} \left[\mathbb{I}(c_{ij} = 1) \log \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)} + \mathbb{I}(c_{ij} = 1) \log \frac{1}{1 + \exp(\theta_i - \beta_j)} \right] \\
&= \sum_{i=1}^{N} \sum_{j=1}^{M} [\mathbb{I}(c_{ij} = 1)(\theta_i - \beta_j) - [\mathbb{I}(c_{ij} = 1) + \mathbb{I}(c_{ij} = 0)] \log(1 + \exp(\theta_i - \beta_j))] \\
&= \sum_{i=1}^{N} \sum_{j=1}^{M} [\mathbb{I}(c_{ij} = 1)(\theta_i - \beta_j) - \mathbb{I}(c_{ij} = 0 \text{ or } 1) \log(1 + \exp(\theta_i - \beta_j))]
\end{aligned}$$

Derivative with respect to θ_i, β_j:

$$\frac{\partial \log p(\mathbf{C} \mid \boldsymbol{\theta}, \boldsymbol{\beta})}{\partial \theta_i} = \sum_{j=1}^{M} \left[\mathbb{I}(c_{ij} = 1) - \mathbb{I}(c_{ij} = 0 \text{ or } 1) \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)} \right]$$

$$\frac{\partial \log p(\mathbf{C} \mid \boldsymbol{\theta}, \boldsymbol{\beta})}{\partial \theta_i} = \sum_{i=1}^{N} \left[-\mathbb{I}(c_{ij} = 1) + \mathbb{I}(c_{ij} = 0 \text{ or } 1) \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)} \right]$$

(b)

Hyperparameters selected:

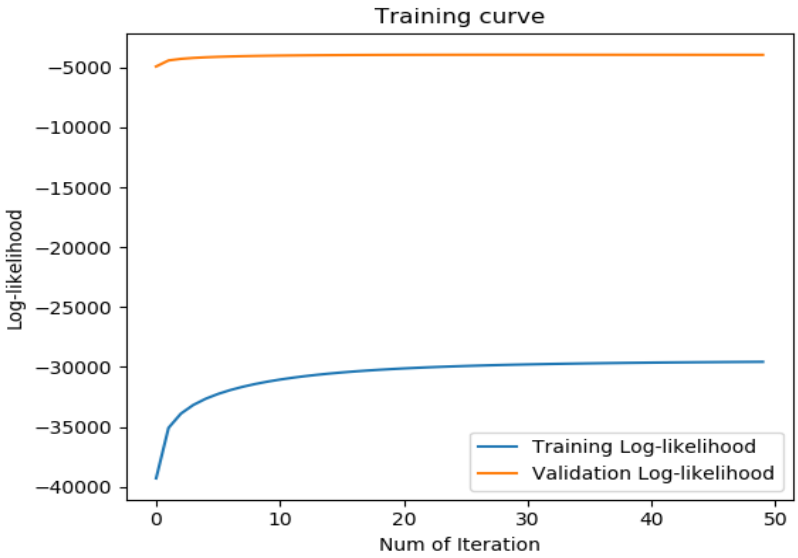
Initialization of $\boldsymbol{\theta}$: Zero Vector

Initialization of $\boldsymbol{\beta}$: Zero Vector

Learning rate: 0.01

Number of iterations: 50

Training curve:



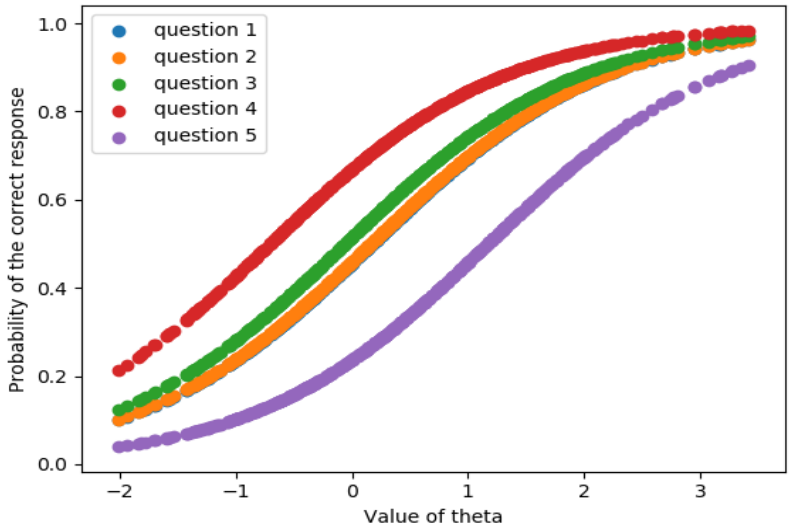
(c)

Final validation accuracy: 0.7060400790290714

Final test accuracy: 0.7070279424216765

(d)

Relationship between the probability of the correct response $p(c_{ij})$ and θ given a question j:



The curves are all S-shaped sigmoid curves with shifts. The shifts are determined by the difficulty of the questions.

These curves represent how the probability of the correct response to a given question increases as the ability of the student increases.

3. Neural Networks:

(a) ALS is a matrix factorization algorithm, which is mainly used to find a low-dimensional representation of the data. While neural networks are not, they are used to fitting an unknown function. Generally speaking, NN is more powerful because it can be used widely. ALS is only for doing matrix factorziation.
ALS is an iterative method and may converge after some iterations. There is no such condition for a NN, we have to set a hyperparameter to control the number of iterations.
ALS is also a direct method, say we are decomposing a sparse matrix C s.t. $C \approx U^T Z$. Every time we fix U or Z and optimize the other one directly via algebraic formula. From this perspective, NN is more like a 'black box', how it optimize depends on the chosen optimizer(e.g. SGD, Adam) and the loss function.

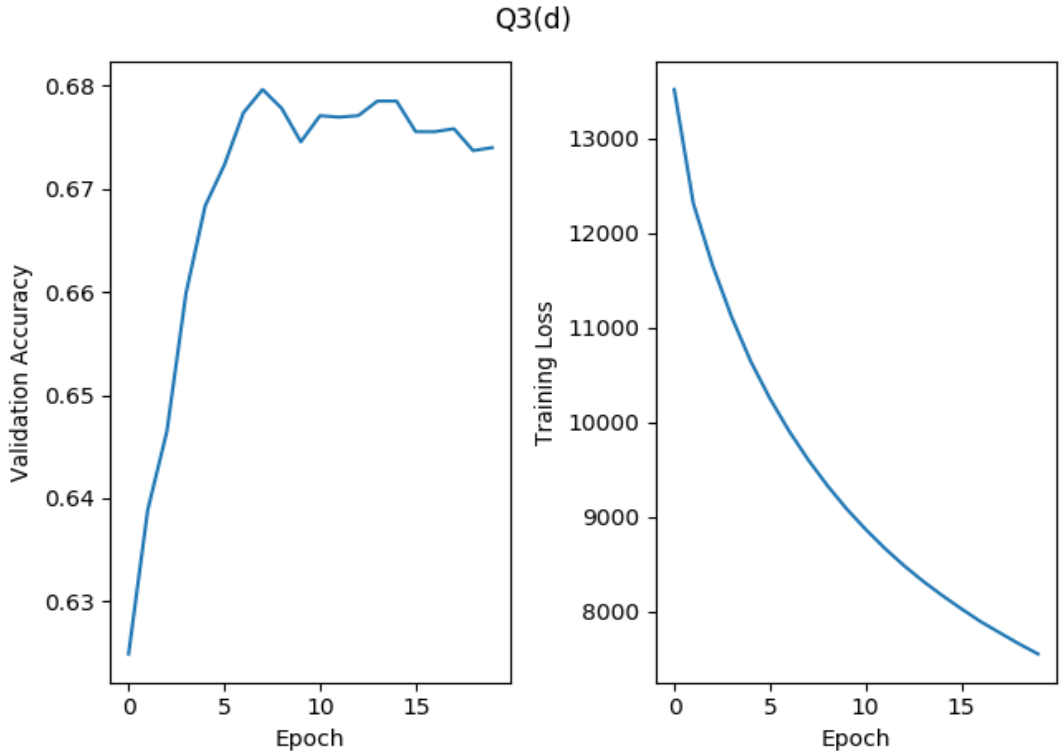
(b) Refer to neural_network.py

(c) The optimal k we selected is 10. ($\alpha = 0.1$)

```
K = 10
Epoch: 0        Training Cost: 13574.730269      Valid Acc: 0.6186847304544172
Epoch: 1        Training Cost: 12370.656125      Valid Acc: 0.6367485182049111
Epoch: 2        Training Cost: 11696.557006      Valid Acc: 0.6508608523849845
Epoch: 3        Training Cost: 11111.308103      Valid Acc: 0.6622918430708439
Epoch: 4        Training Cost: 10599.196796      Valid Acc: 0.6766864239345187
Epoch: 5        Training Cost: 10155.998873      Valid Acc: 0.6809201241885408
Epoch: 6        Training Cost: 9787.055136       Valid Acc: 0.6836014676827548
Epoch: 7        Training Cost: 9472.856223       Valid Acc: 0.6843070843917584
Epoch: 8        Training Cost: 9207.502061       Valid Acc: 0.6854360711261642
Epoch: 9        Training Cost: 8966.861737       Valid Acc: 0.6850127011007621
K = 50
Epoch: 0        Training Cost: 13322.431944      Valid Acc: 0.630962461191081
Epoch: 1        Training Cost: 11912.872389      Valid Acc: 0.6546711826136042
Epoch: 2        Training Cost: 10911.479287      Valid Acc: 0.6682190234264748
Epoch: 3        Training Cost: 10034.412396      Valid Acc: 0.67499294383291
Epoch: 4        Training Cost: 9210.049967       Valid Acc: 0.6810612475303415
Epoch: 5        Training Cost: 8401.886630       Valid Acc: 0.682895850973751
Epoch: 6        Training Cost: 7620.257146       Valid Acc: 0.6810612475303415
Epoch: 7        Training Cost: 6887.349825       Valid Acc: 0.6795088907705334
Epoch: 8        Training Cost: 6220.681038       Valid Acc: 0.6776742873271239
Epoch: 9        Training Cost: 5621.702287       Valid Acc: 0.6718882303132938
K = 100
Epoch: 0        Training Cost: 13506.787812      Valid Acc: 0.6281399943550663
Epoch: 1        Training Cost: 12023.424885      Valid Acc: 0.6507197290431838
Epoch: 2        Training Cost: 11000.800465      Valid Acc: 0.6645498165396556
Epoch: 3        Training Cost: 10057.346963      Valid Acc: 0.6731583403895004
Epoch: 4        Training Cost: 9051.321212       Valid Acc: 0.6773920406435224
Epoch: 5        Training Cost: 8002.457235       Valid Acc: 0.6804967541631386
Epoch: 6        Training Cost: 6953.072947       Valid Acc: 0.6788032740615297
Epoch: 7        Training Cost: 5981.731284       Valid Acc: 0.6775331639853232
Epoch: 8        Training Cost: 5118.657616       Valid Acc: 0.6706181202370872
Epoch: 9        Training Cost: 4378.023125       Valid Acc: 0.6686423934518769
```

```
K = 200
Epoch: 0        Training Cost: 13945.662182      Valid Acc: 0.6206604572396275
Epoch: 1        Training Cost: 12364.171479      Valid Acc: 0.642111205193339
Epoch: 2        Training Cost: 11328.833280      Valid Acc: 0.6560824160316117
Epoch: 3        Training Cost: 10481.509228      Valid Acc: 0.6686423934518769
Epoch: 4        Training Cost: 9581.361656       Valid Acc: 0.6742873271239063
Epoch: 5        Training Cost: 8541.488857       Valid Acc: 0.6755574372001129
Epoch: 6        Training Cost: 7400.962617       Valid Acc: 0.677109793959921
Epoch: 7        Training Cost: 6266.682148       Valid Acc: 0.6738639570985041
Epoch: 8        Training Cost: 5225.156892       Valid Acc: 0.6714648602878917
Epoch: 9        Training Cost: 4320.640509       Valid Acc: 0.6679367767428732
K = 500
Epoch: 0        Training Cost: 15351.067042      Valid Acc: 0.6059836296923511
Epoch: 1        Training Cost: 13447.441891      Valid Acc: 0.6234829240756421
Epoch: 2        Training Cost: 12350.607979      Valid Acc: 0.6337849280270956
Epoch: 3        Training Cost: 11498.382158      Valid Acc: 0.6474738921817669
Epoch: 4        Training Cost: 10743.373648      Valid Acc: 0.6562235393734124
Epoch: 5        Training Cost: 9932.686412       Valid Acc: 0.6668077900084673
Epoch: 6        Training Cost: 8982.091026       Valid Acc: 0.6707592435788879
Epoch: 7        Training Cost: 7880.051351       Valid Acc: 0.6737228337567034
Epoch: 8        Training Cost: 6702.163766       Valid Acc: 0.6741462037821055
Epoch: 9        Training Cost: 5579.373962       Valid Acc: 0.6696302568444821
```

(d)



The final test accuracy is 0.676.

```
Test Accuracy = 0.6762630539091166.
```

(e) The optimal λ we selected is 0.01.

```
lambda = 0.001.
Epoch: 0        Training Cost: 13539.524803      Valid Acc: 0.6185436071126165
Epoch: 1        Training Cost: 12357.145674      Valid Acc: 0.6409822184589331
Epoch: 2        Training Cost: 11652.399471      Valid Acc: 0.6566469093988145
Epoch: 3        Training Cost: 11038.013710      Valid Acc: 0.6696302568444821
Epoch: 4        Training Cost: 10538.885708      Valid Acc: 0.6782387806943269
Epoch: 5        Training Cost: 10131.943629      Valid Acc: 0.6826136042901496
Epoch: 6        Training Cost: 9802.113089       Valid Acc: 0.6821902342647473
Epoch: 7        Training Cost: 9505.529339       Valid Acc: 0.68077900084674
Epoch: 8        Training Cost: 9251.169859       Valid Acc: 0.6843070843917584
Epoch: 9        Training Cost: 9021.227876       Valid Acc: 0.6843070843917584
lambda = 0.01.
Epoch: 0        Training Cost: 13652.424121      Valid Acc: 0.6181202370872142
Epoch: 1        Training Cost: 12471.224669      Valid Acc: 0.6370307648885125
Epoch: 2        Training Cost: 11814.461834      Valid Acc: 0.6487440022579735
Epoch: 3        Training Cost: 11237.683411      Valid Acc: 0.6597516229184307
Epoch: 4        Training Cost: 10753.570690      Valid Acc: 0.6682190234264748
Epoch: 5        Training Cost: 10350.517769      Valid Acc: 0.6735817104149027
Epoch: 6        Training Cost: 10023.256874      Valid Acc: 0.6785210273779283
Epoch: 7        Training Cost: 9746.599776       Valid Acc: 0.681343494213943
Epoch: 8        Training Cost: 9500.292364       Valid Acc: 0.6848715777589613
Epoch: 9        Training Cost: 9276.848368       Valid Acc: 0.6840248377081569
lambda = 0.1.
Epoch: 0        Training Cost: 14328.765959      Valid Acc: 0.6268698842788597
Epoch: 1        Training Cost: 13286.620254      Valid Acc: 0.6414055884843353
Epoch: 2        Training Cost: 12745.163933      Valid Acc: 0.6539655659046006
Epoch: 3        Training Cost: 12318.287537      Valid Acc: 0.6666666666666666
Epoch: 4        Training Cost: 11991.638709      Valid Acc: 0.672311600338696
Epoch: 5        Training Cost: 11762.169687      Valid Acc: 0.6766864239345187
Epoch: 6        Training Cost: 11583.626853      Valid Acc: 0.6783799040361276
Epoch: 7        Training Cost: 11446.036993      Valid Acc: 0.6809201241885408
Epoch: 8        Training Cost: 11325.768469      Valid Acc: 0.6837425910245555
Epoch: 9        Training Cost: 11227.104949      Valid Acc: 0.6845893310753599
```

```
lambda = 1.
Epoch: 0        Training Cost: 17229.468884      Valid Acc: 0.6081004798193621
Epoch: 1        Training Cost: 13805.054391      Valid Acc: 0.6217894439740334
Epoch: 2        Training Cost: 13320.796388      Valid Acc: 0.6232006773920407
Epoch: 3        Training Cost: 13041.192936      Valid Acc: 0.6244707874682472
Epoch: 4        Training Cost: 12870.943589      Valid Acc: 0.6234829240756421
Epoch: 5        Training Cost: 12763.813358      Valid Acc: 0.6243296641264465
Epoch: 6        Training Cost: 12694.519478      Valid Acc: 0.6243296641264465
Epoch: 7        Training Cost: 12647.281445      Valid Acc: 0.6255997742026531
Epoch: 8        Training Cost: 12614.181872      Valid Acc: 0.624611910810048
Epoch: 9        Training Cost: 12589.959116      Valid Acc: 0.6240474174428451
```

Final validation accuracy: 0.6764

Final test accuracy: 0.6797

```
Validation Accuracy = 0.6764041772509173.
Test Accuracy = 0.6796500141123342.
```

Compared with (d), the model performs better to some extent after regularization.

4. Ensemble

In this section, we trained three Item Response Theory models with bootstrapping the training set.

Ensemble process implemented (referring to ensemble.py):

1. Function **bootstrap**(data) takes a dictionary data {user_id: list, question_id: list, is_correct: list} and returns a bootstrap sample of the training set. The returned data has the same sample size as the original data. This is implemented by randomly selecting data points from the original data with replacement. (The pairs user_id, question_id, is_correct are preserved.)

2. If a data point is selected more than once, then it will be counted that many times. This is done by using a scipy **csc_matrix** to transform the dictionary data into a sparse matrix in the implementation of IRT (question 2). When transforming the data into a sparse matrix, the values at the same position (user_id, question_id) will be added together. This is equivalent to re-weight the data by how many times each one is selected in bootstrap.

3. Function **bootstrap_evaluate**(data, theta_list, beta_list) is used to predict the ensemble correctness. theta_list and beta_list include the three base IRT models trained with bootstrap. To make the prediction, each model will predict a conditional probability z_i ranged in [0, 1]. For each query data, the prediction is made by averaging the probabilities given by the three models. That is,

$$y_{\text{ensemble}} = \mathbb{I}(\frac{1}{3}(z_1 + z_2 + z_3) \geq 0.5)$$

Result we get:

Validation accuracy for each single base model:

[0.6953147050522156, 0.6978549252046289, 0.696725938470223]

Test accuracy for each single base model:

[0.703076488851256, 0.7036409822184589, 0.6971493084956252]

(Final) Ensemble validation accuracy is: 0.7015241320914479

(Final) Ensemble test accuracy is: 0.7104149026248942

Following is some findings according to the result:

1. The ensemble performance always tends to be better than that of each single base model.

2. Compared to the IRT model trained without bootstrap, sometimes we can get a better performance and sometimes we can not. This is due to the randomness of the ensemble process. But in general, there is no obvious improvement using the ensemble. The reason could be that the bootstrapped samples generated by the training set are correlated. The variance might not get reduced and bootstrap does not reduce the bias in nature, so there tends to be no obvious improvement in performance using the ensemble.

2 Part B

In this part, we will modify the Item Response Theory algorithm to get a better performance.

2.1 Description

Extension:

In general, we introduce a new set of parameters $\boldsymbol{\gamma}$ to the model.

Recall that the baseline model in part A defines the conditional probability as

$$p(c_{ij} = 1 \mid \boldsymbol{\theta}, \boldsymbol{\beta}) = \text{sigmoid}(\theta_i - \beta_j)$$

Here we modify the conditional probability model to

$$p(c_{ij} = 1 \mid \boldsymbol{\theta}, \boldsymbol{\beta}, \boldsymbol{\gamma}) = \text{sigmoid}(\theta_i - \beta_j - \sum_{k \in \text{subject}(j)} \gamma_k)$$

where subject(j) is the set of all the subjects that question j belongs to.
For simplicity, following we may use s(j) for subject(j).

Interpretation:
We measure the difficulty of a question as the sum of two parts:
the difficulties induced by the subjects it belongs to, and the difficulty of itself.
That is:

$$\text{difficulty of question } j = \sum_{\text{subject } k \text{ of } j} \text{difficulty of } k + \text{difficulty of } j \text{ itself}$$

The set of parameters $\boldsymbol{\gamma}$ is introduced to represent the difficulty of each subject.
There are 387 subjects in total, so the dimensional of parameter $\boldsymbol{\gamma}$ is 387.

Recompute the log-likelihood:

$$\log p(\mathbf{C} \mid \boldsymbol{\theta}, \boldsymbol{\beta}, \boldsymbol{\gamma}) = \sum_{i=1}^{N} \sum_{j=1}^{M} \left[\mathbb{I}(c_{ij} = 1)(\theta_i - \beta_j - \sum_{k \in \text{s}(j)} \gamma_k) - \mathbb{I}(c_{ij} = 0 \text{ or } 1) \log(1 + \exp(\theta_i - \beta_j - \sum_{k \in \text{s}(j)} \gamma_k)) \right]$$

Gradient used in the update rule for γ_k:

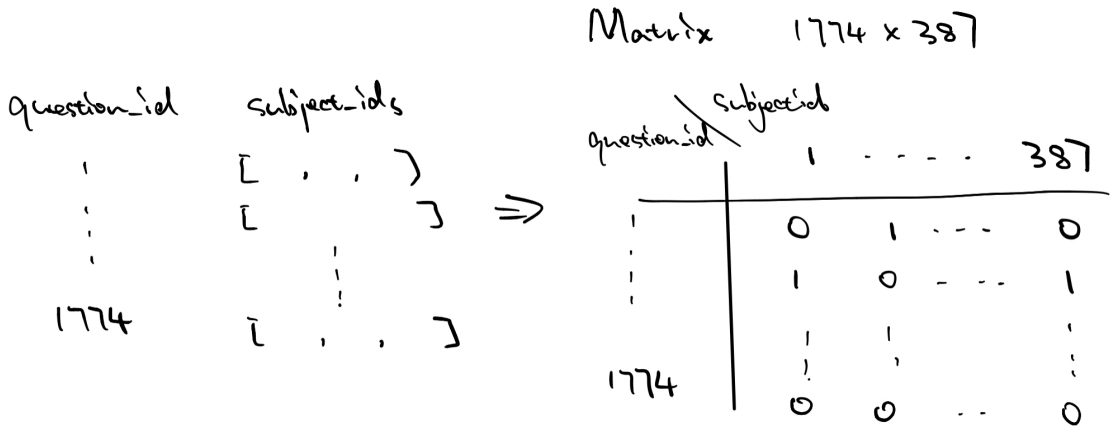
$$\frac{\partial \log p(\mathbf{C} \mid \boldsymbol{\theta}, \boldsymbol{\beta}, \boldsymbol{\gamma})}{\partial \gamma_k} = \sum_{i=1}^{N} \sum_{j : k \in s(j)} \left[-\mathbb{I}(c_{ij} = 1) + \mathbb{I}(c_{ij} = 0 \text{ or } 1) \text{sigmoid}(\theta_i - \beta_j - \sum_{k' \in s(j)} \gamma_{k'}) \right]$$

Note that this is a sum over the questions (columns) that belongs to a specific subject γ_k.

Extra data:

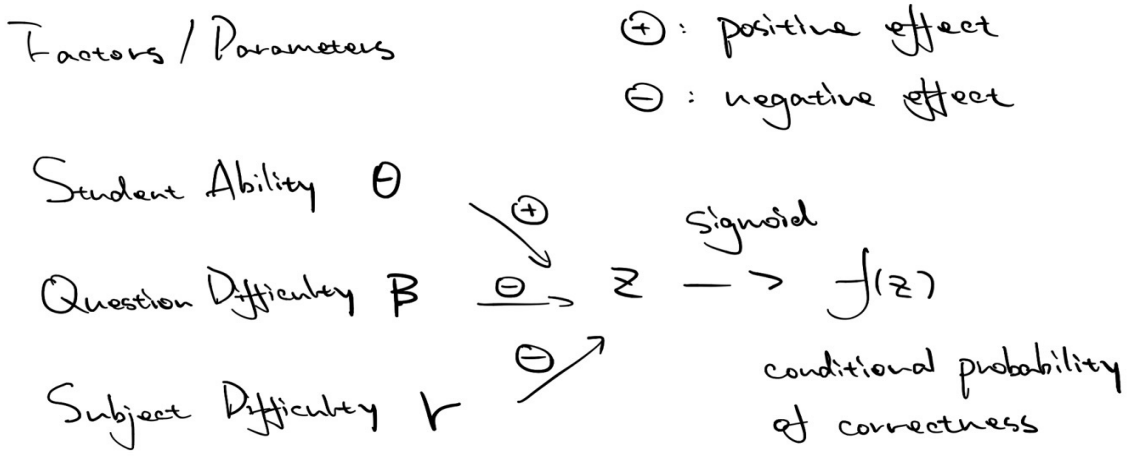
We introduce the question_meta data to our algorithm, which includes the lists of subjects each question belongs to.

Matrix representation for the question-subject data:



We use a 2D-binary matrix (1774 * 387) M to represent the subjects each question belongs to. We have:
$M[j,k] = 1$ if question j belongs to subject k
$M[j,k] = 0$ otherwise.

2.2 Model Diagram



In general, we introduce a set of new parameters γ which aims to improve the optimization of the baseline model.

Weight the two part of the difficulties:

We hope that the overall difficulty of a question is mainly contributed by *beta*, and the difficulty of the subjects it belongs to only contribute a small part. Otherwise, because of the large number of subjects (high dimension of $\boldsymbol{\gamma}$), the model will diverge with unexpected behavior.

We implement this weighting process in the update function. In each iteration, we update $\boldsymbol{\gamma}$ with a smaller scale.

The algorithm:

---

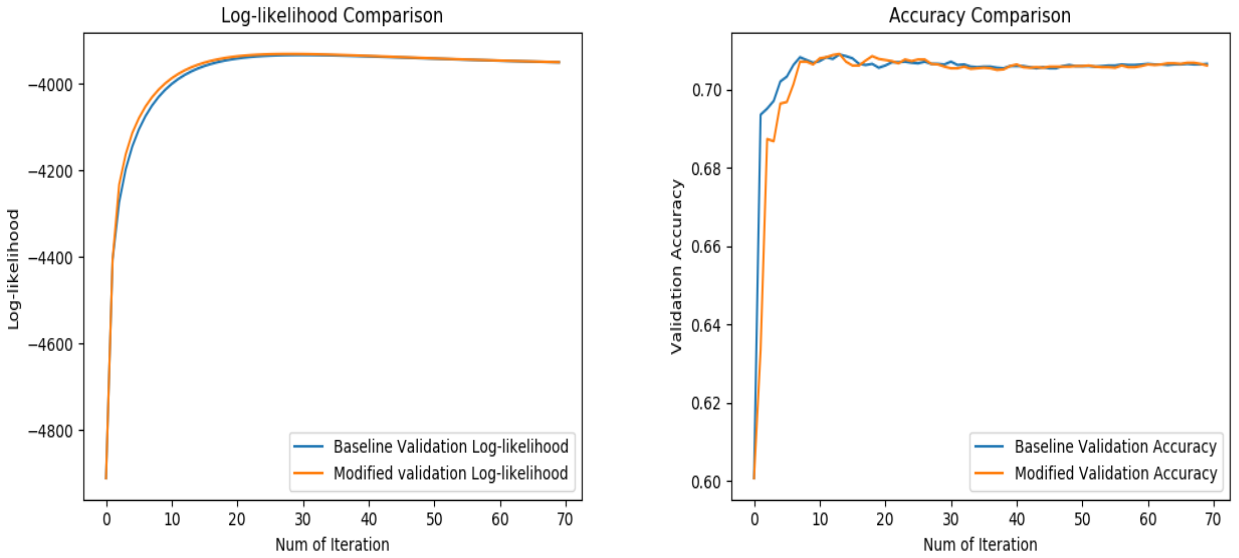
procedure ALTERNATING GRADIENT DESCENT(data, $\boldsymbol{\theta}, \boldsymbol{\beta}, \boldsymbol{\gamma}$)
    Calculate gradient of $\boldsymbol{\theta}$
    $\boldsymbol{\theta} = \boldsymbol{\theta}$ - gradient($\boldsymbol{\theta}$) * learning rate

    Reset the parameters
    Calculate gradient of $\boldsymbol{\beta}$
    $\boldsymbol{\beta} = \boldsymbol{\beta}$ - gradient($\boldsymbol{\beta}$) * learning rate

    Reset the parameters
    Calculate gradient of $\boldsymbol{\gamma}$
    $\boldsymbol{\gamma} = \boldsymbol{\gamma}$ - gradient($\boldsymbol{\gamma}$) * learning rate * **small weight** (less than 1)
    return $\boldsymbol{\theta}, \boldsymbol{\beta}, \boldsymbol{\gamma}$

Refers to part_b/partb.py for the entire implementation (including data processing, log-likelihood, irt, evaluate).

2.3 Comparison



The final test accuracy of the modified model is 0.707874682472481, and sometimes can get over 0.709 with a good weight and learning rate setting.

There is a 0.1 - 0.2% improvement in test accuracy than the baseline model.

We can also see there is also a small improvement in the log-likelihood on the validation set during the training process.

The main modification for our model in part B is to introduce a new set of parameters. As the number of parameters increases, the complexity of the model increases. Thus, the model is expected to get a better optimization and to make a better prediction.

We have tried a large number of experiments on the baseline model without the new parameter $\boldsymbol{\gamma}$ and our modified model. We keep all the hyperparameter settings (learning rate, number of iterations, parameters initialization) the same for the two models and train them at the same time. We find that most of the time there is an obvious improvement on the test accuracy for our modified model. This agrees with our hypothesis that the introduction of the new parameters would help to improve the optimization of the model.

2.4 Limitations

- If we give the parameter $\boldsymbol{\gamma}$ a higher weight, that is to say the difficulty of each question is mainly contributed by the difficulty of the subjects it belongs to, then our model could not converge properly. This might because that the number (dimension) of the subjects is high, so there will be conflicts if they are updated in the same scale with β (the influence of γ is far large than that of β).

- In terms of the convergence, another limitation is the way of updating the parameters. As the number of parameters increases, the method Alternating Gradient Descent might not perform that well to find the local maximum. Although it might still be good for two sets of parameters, the convergence becomes more complicated when we introduce the new set of parameters. This gives much more limitations to the setting of hyperparameters. A possible extension would be to use a more advanced gradient descent method.

- The time complexity of the algorithm would increase linearly as the the number of subjects increases. So when the number of subjects get larger, the algorithm could become time inefficient. A possible extension in the future could be to replace the data structures with some efficient ones, for example using a 3-D array to store the data.

- According to the training curve on the validation set, the log-likelihood seems to have a small decrease when the number of iterations gets large. This might indicate overfitting to some extent, so maybe we can also apply some methods like early stopping to avoid this.