CSC311 Final Project

Shiven Taneja (1005871013),
Jonathan Chung(1005415953),
Yuyang Chen(1003892200)

January 26, 2022

1 k-Nearest Neighbor

1.a

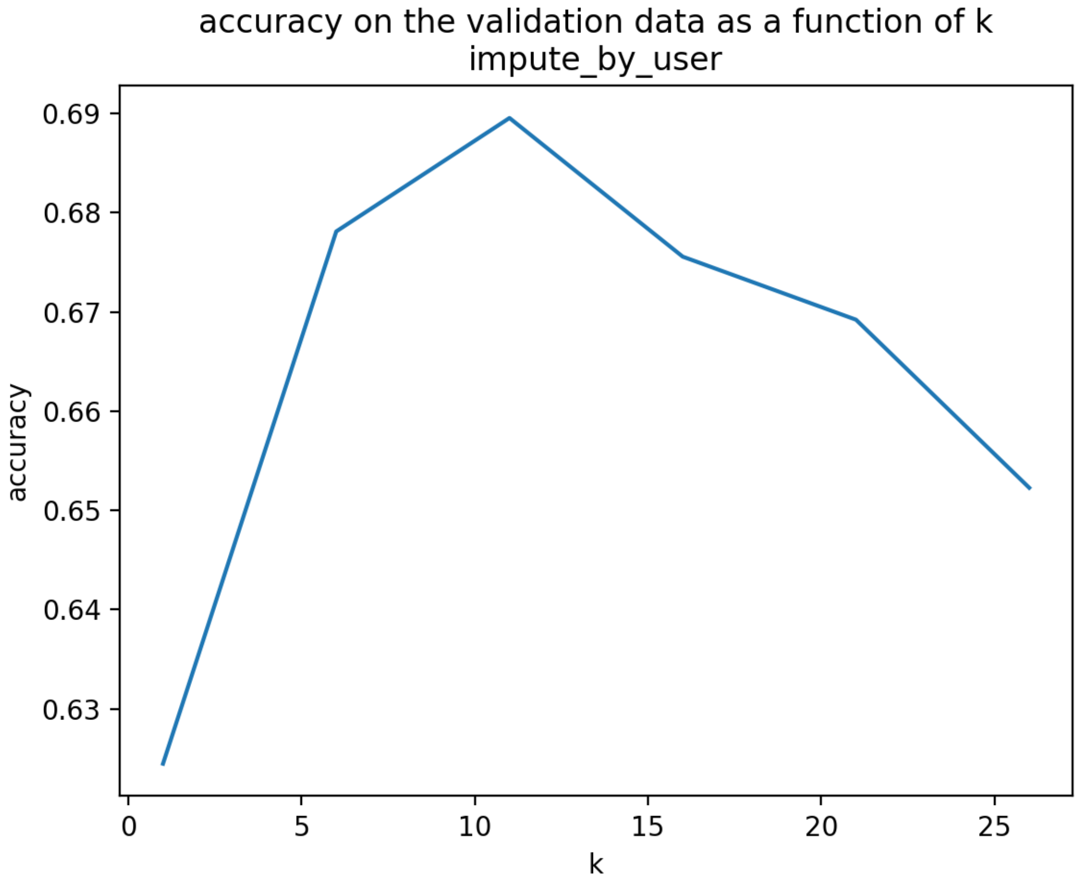


Figure 1: accuracy on the validation data as a function of k

```
accuracy on the validation data as a function of k (user)
[0.6244707874682472, 0.6780976573525261, 0.6895286480383855, 0.6755574372001129, 0.6692068868190799, 0.6522720858029918]
accuracy on the test data as a function of k (user)
[0.6356195314705052, 0.6646909398814564, 0.6841659610499576, 0.6692068868190799, 0.6683601467682755, 0.6528365791701948]
k* for user-based = 11
```

Figure 2: The result of accuracy

Note that the accuracy list above is in the same order as the k list for k $\in \{1, 6, 11, 16, 21, 26\}$.

1.b

From Figure 2, we see that the optimal k value ($k*$) on validation data is 11 for user-based given the accuracy of 0.6895286480383855. Figure 1 shows the accuracy on the validation data for $k \in \{1, 6, 11, 16, 21, 26\}$, the highest point in the plot indicates the optimal k for user-based is 11. The final test accuracy is 0.6841659610499576.

1.c

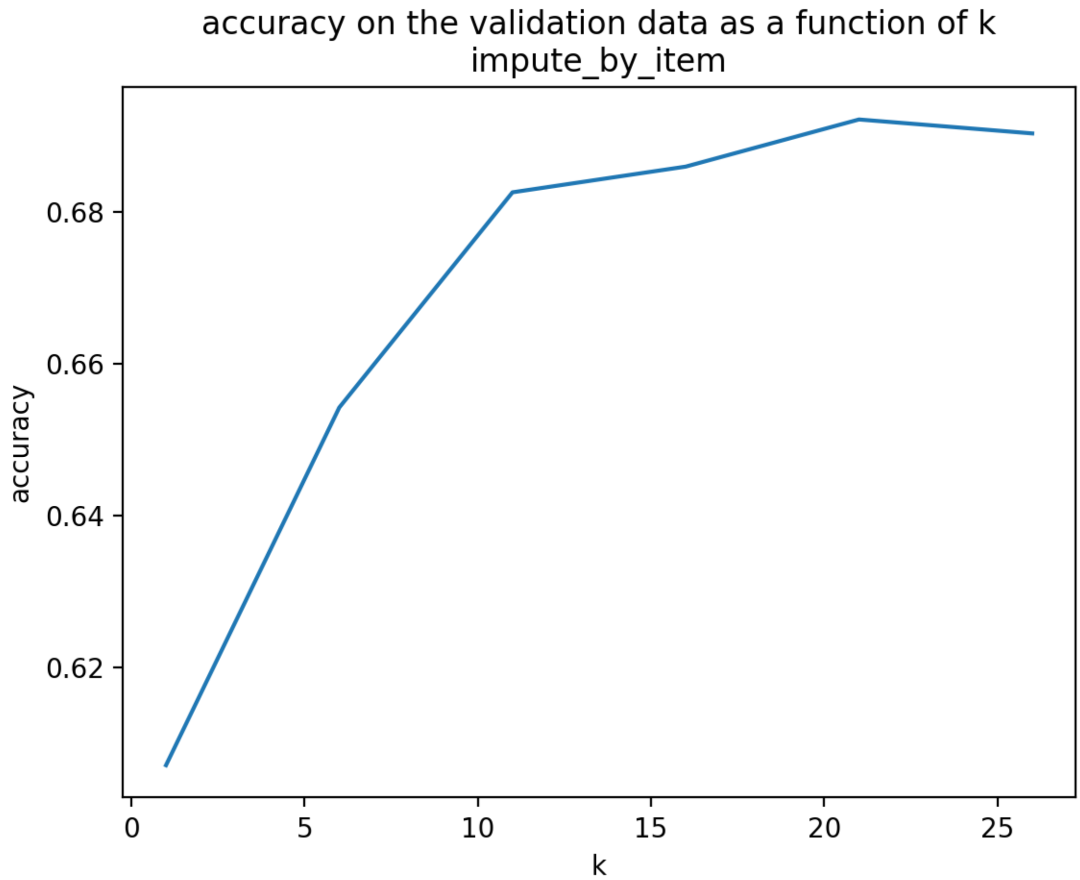


Figure 3: accuracy on the validation data as a function of k

From Figure 4, we see that the optimal k value ($k*$) on validation data is 21 for item-based given the accuracy of 0.6922099915325995. Figure 3 shows the accuracy on the validation data for $k \in \{1, 6, 11, 16, 21, 26\}$, the highest point in the plot indicates the optimal k for user-based is 21. The final test accuracy is 0.6816257408975445.

```
accuracy on the validation data as a function of k (item)
[0.607112616426757, 0.6542478125882021, 0.6826136042901496, 0.6860005644933672, 0.6922099915325995, 0.69037538808919]
accuracy on the test data as a function of k (item)
[0.6057013830087496, 0.6539655659046006, 0.6689246401354784, 0.676545300592718, 0.6816257408975445, 0.6816257408975445]
k* for item-based = 21
```

Figure 4: The result of accuracy on the validation data

1.d

In terms of the test performance for $(k*)$, the accuracy for user-based is 68.42%, whereas the accuracy for item-based is 68.16%. As a result, the performance of the user-based method is better than the performance of the item-based method.

1.e

Limitations of kNN:

- Relatively high computational cost. It takes O(nd) for Euclidean distance calculation and O(n log n) for distance sorting where n is data size of dimension d, potentially extra time to replace the missing data. As a result, the kNN method is time consuming.
- looking at the performances for the two methods (item-based and user-based), kNN algorithm has its limit to predict the current data as it yields low accuracy of predictions.

2 Item Response Theory

2.a

a) We know that:

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

Thus we know:

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

Therefore:

$$\log(p(c \mid \theta, \beta)) = \log\left(\prod_{i=1}^{N}\prod_{j=1}^{M}\left[\left(\frac{\exp(\theta_i - \beta_j)}{1+\exp(\theta_i - \beta_j)}\right)^{\mathbb{I}(c_{ij}=1)} \cdot \left(\frac{1}{1+\exp(\theta_i - \beta_j)}\right)^{\mathbb{I}(c_{ij}=0)}\right]\right)$$

$$= \sum_{i=1}^{N}\sum_{j=1}^{M}\left[\mathbb{I}(c_{ij}=1) \cdot \log\left(\frac{\exp(\theta_i - \beta_j)}{1+\exp(\theta_i - \beta_j)}\right) + \mathbb{I}(c_{ij}=0) \log\left(\frac{1}{1+\exp(\theta_i - \beta_j)}\right)\right]$$

$$\log(P(c \mid \theta, \beta)) = \sum_{i=1}^{N}\sum_{j=1}^{M}\left[\mathbb{I}(c_{ij}=1) \cdot (\theta_i - \beta_j) - \mathbb{I}(c_{ij} \in \{0,1\}) \cdot \log(1+\exp(\theta_i - \beta_j))\right]$$

Finding the derivative W.R.T. θ_i:

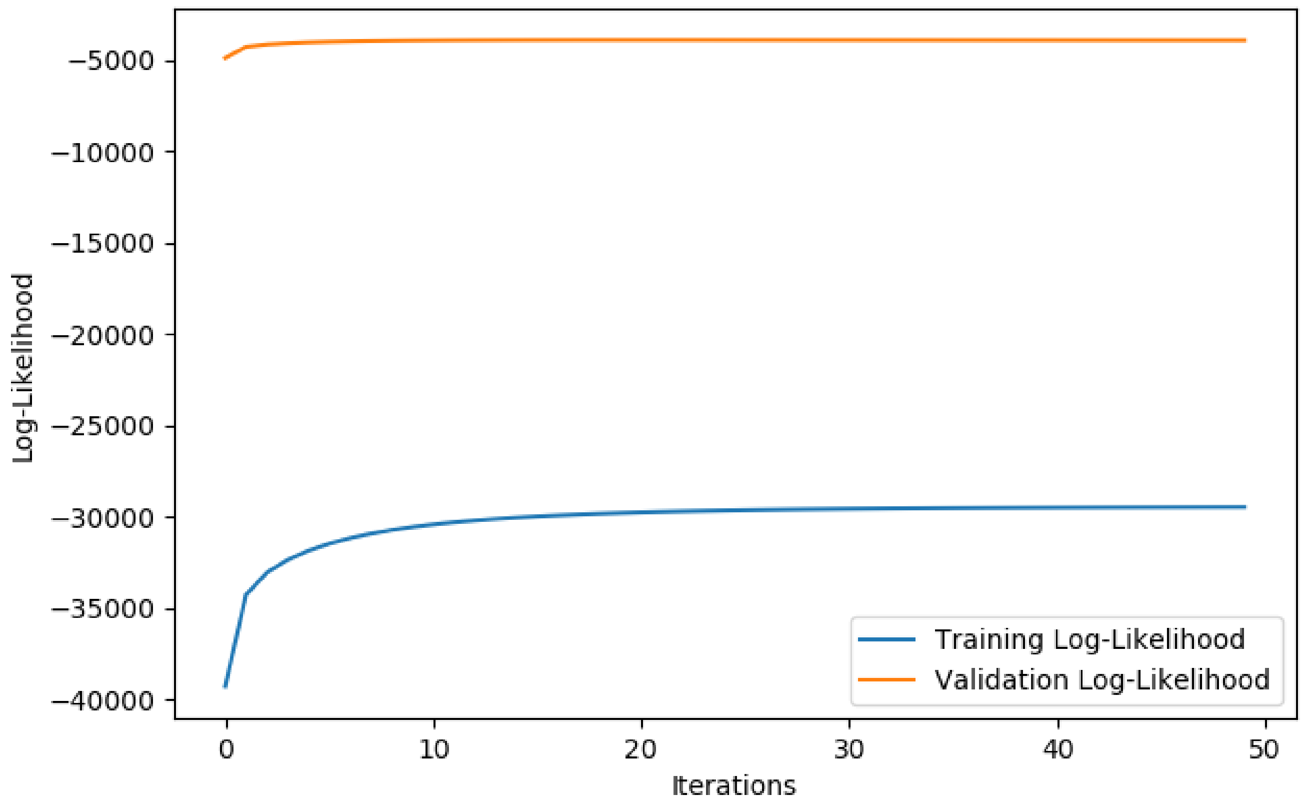
$$\frac{\partial(\log(P(c \mid \theta, \beta)))}{\partial \theta_i} = \sum_{j=1}^{M}\left[\mathbb{I}(c_{ij}=1) - \mathbb{I}(c_{ij} \in \{0,1\}) \cdot \frac{\exp(\theta_i - \beta_j)}{1+\exp(\theta_i - \beta_j)}\right]$$

Finding the derivative W.R.T. β_j:

$$\frac{\partial(\log(P(c \mid \theta, \beta)))}{\partial \beta_j} = \sum_{i=1}^{N}\left[-\mathbb{I}(c_{ij}=1) + \mathbb{I}(c_{ij} \in \{0,1\}) \frac{\exp(\theta_i - \beta_j)}{1+\exp(\theta_i - \beta_j)}\right]$$

2.b

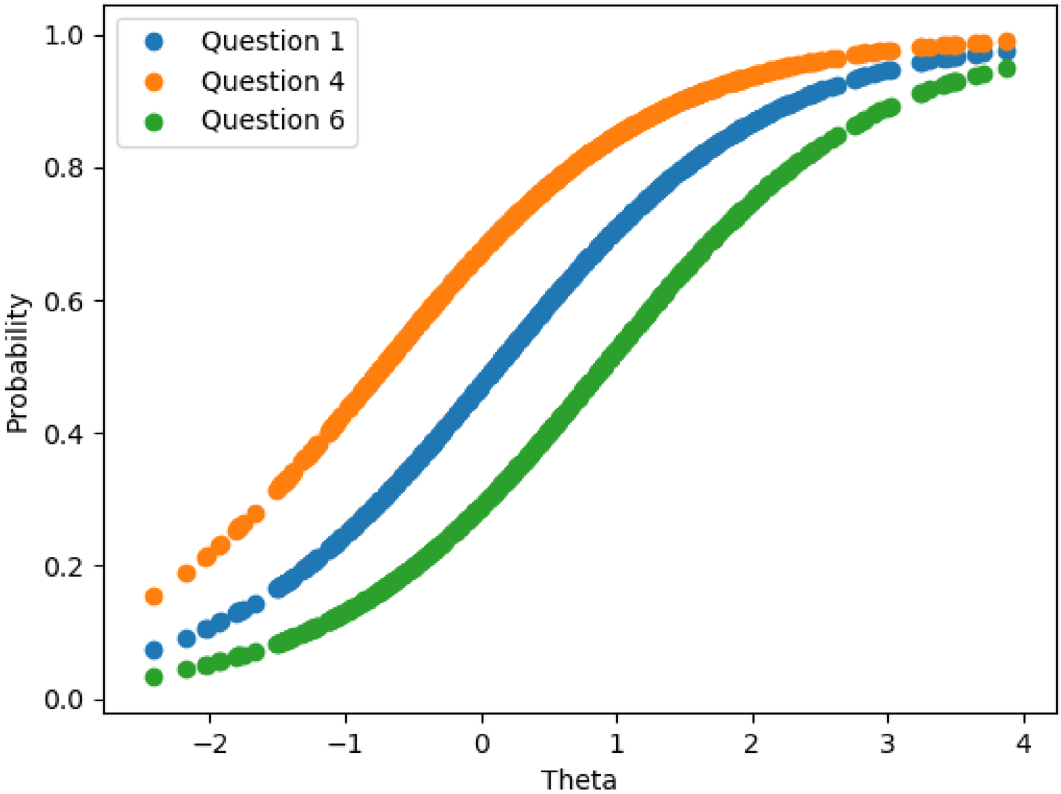
Hyper parameters:
Learning Rate: 0.015
Iterations: 50
Θ initialization: 0
β initialization: 0



2.c

```
Validation Accuracy:  0.7060400790290714
Test Accuracy: 0.7070279424216765
```

2.d



Each curve is a shifted sigmoid curve by the difficulty of each question. It tells us how the probability that a student gets the right answer based on their level (theta).

3 Neural Network

(a) Three differences between ALS and neural networks:

- Neural networks can adequately approximate complex nonlinear Data, but ALS can only fit linear data.
- ALS is based on simple calculations and has a shorter learning time, but Neural networks require a large number of parameters and long learning time.
- Neural Network works as supervise learning, but ALS is unsupervise learning.

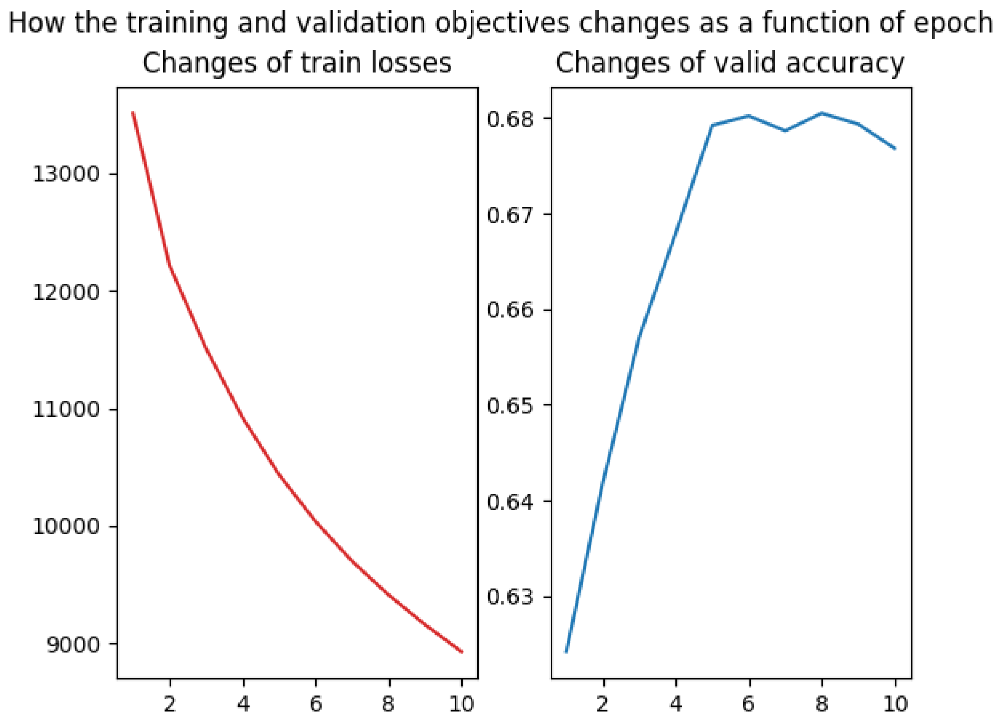
(c) We trained the autoencoder using different values of k and other hyperparameters. Finally, we set k = 10, learning rate = 0.1 and the number of iterations is 10, to get the highest accuracy. The codes showing the different k values are below:

```
K =  10 lr =  0.1
Epoch: 0    Training Cost: 13520.999304  Valid Acc: 0.619813717188823
Epoch: 1    Training Cost: 12279.361857  Valid Acc: 0.6383008749647192
Epoch: 2    Training Cost: 11563.442075  Valid Acc: 0.655376799322608
Epoch: 3    Training Cost: 10967.544756  Valid Acc: 0.6676545300592718
Epoch: 4    Training Cost: 10492.964518  Valid Acc: 0.6737228337567034
Epoch: 5    Training Cost: 10101.995681  Valid Acc: 0.6790855207451313
Epoch: 6    Training Cost: 9764.879256   Valid Acc: 0.6806378775049393
Epoch: 7    Training Cost: 9469.625344   Valid Acc: 0.6826136042901496
Epoch: 8    Training Cost: 9206.353725   Valid Acc: 0.682895850973751
Epoch: 9    Training Cost: 8971.117660   Valid Acc: 0.6841659610499576
K =  50 lr =  0.1
Epoch: 0    Training Cost: 13327.520209  Valid Acc: 0.62884561106407
Epoch: 1    Training Cost: 11911.142947  Valid Acc: 0.656929156082416
Epoch: 2    Training Cost: 10903.371549  Valid Acc: 0.6666666666666666
Epoch: 3    Training Cost: 10021.558885  Valid Acc: 0.674428450465707
Epoch: 4    Training Cost: 9173.808334   Valid Acc: 0.6802145074795372
Epoch: 5    Training Cost: 8350.771845   Valid Acc: 0.6817668642393452
Epoch: 6    Training Cost: 7562.539849   Valid Acc: 0.6840248377081569
Epoch: 7    Training Cost: 6832.822338   Valid Acc: 0.6806378775049393
Epoch: 8    Training Cost: 6171.797293   Valid Acc: 0.6780976573525261
Epoch: 9    Training Cost: 5574.938293   Valid Acc: 0.6778154106689246
```

```
K =  100 lr =  0.1
Epoch: 0    Training Cost: 13510.366057  Valid Acc: 0.6287044877222693
Epoch: 1    Training Cost: 12043.604995  Valid Acc: 0.6501552356759808
Epoch: 2    Training Cost: 11025.708289  Valid Acc: 0.6673722833756703
Epoch: 3    Training Cost: 10095.620643  Valid Acc: 0.6703358735534858
Epoch: 4    Training Cost: 9117.125078   Valid Acc: 0.6766864239345187
Epoch: 5    Training Cost: 8093.672920   Valid Acc: 0.6819079875811459
Epoch: 6    Training Cost: 7059.935860   Valid Acc: 0.6814846175557437
Epoch: 7    Training Cost: 6091.676214   Valid Acc: 0.6817668642393452
Epoch: 8    Training Cost: 5223.232293   Valid Acc: 0.678238780694 3269
Epoch: 9    Training Cost: 4468.753368   Valid Acc: 0.6759808072255151
K =  200 lr =  0.1
Epoch: 0    Training Cost: 13955.322071  Valid Acc: 0.6182613604290149
Epoch: 1    Training Cost: 12396.859047  Valid Acc: 0.6408410951171324
Epoch: 2    Training Cost: 11362.677554  Valid Acc: 0.655376799322608
Epoch: 3    Training Cost: 10480.909684  Valid Acc: 0.663279706463449
Epoch: 4    Training Cost: 9555.542014   Valid Acc: 0.6717471069714931
Epoch: 5    Training Cost: 8494.661980   Valid Acc: 0.67499294383291
Epoch: 6    Training Cost: 7345.332489   Valid Acc: 0.6727349703640982
Epoch: 7    Training Cost: 6195.071176   Valid Acc: 0.672311600338696
Epoch: 8    Training Cost: 5145.183527   Valid Acc: 0.6676545300592718
Epoch: 9    Training Cost: 4240.620482   Valid Acc: 0.6652554332486593
```

```
K =  500 lr =  0.1
Epoch: 0    Training Cost: 15337.506256  Valid Acc: 0.6059836296923511
Epoch: 1    Training Cost: 13420.328105  Valid Acc: 0.6250352808354502
Epoch: 2    Training Cost: 12308.320481  Valid Acc: 0.6359017781541066
Epoch: 3    Training Cost: 11452.393335  Valid Acc: 0.6497318656505786
Epoch: 4    Training Cost: 10707.548518  Valid Acc: 0.6559412926898109
Epoch: 5    Training Cost: 9910.402828   Valid Acc: 0.6675134067174711
Epoch: 6    Training Cost: 8981.515978   Valid Acc: 0.6718882303132938
Epoch: 7    Training Cost: 7890.396397   Valid Acc: 0.6696302568444821
Epoch: 8    Training Cost: 6724.641678   Valid Acc: 0.6683601467682755
Epoch: 9    Training Cost: 5610.500118   Valid Acc: 0.6683601467682755
Epoch: 0    Training Cost: 13714.736742  Valid Acc: 0.6226361840248377
Epoch: 1    Training Cost: 12513.231706  Valid Acc: 0.6343494213942986
Epoch: 2    Training Cost: 11907.630283  Valid Acc: 0.6512842224103866
Epoch: 3    Training Cost: 11414.528177  Valid Acc: 0.6651143099068586
Epoch: 4    Training Cost: 11026.444844  Valid Acc: 0.6738639570985041
Epoch: 5    Training Cost: 10731.623390  Valid Acc: 0.6775331639853232
Epoch: 6    Training Cost: 10501.716925  Valid Acc: 0.6837425910245555
Epoch: 7    Training Cost: 10310.312519  Valid Acc: 0.6844482077335591
Epoch: 8    Training Cost: 10155.264232  Valid Acc: 0.6833192209991532
Epoch: 9    Training Cost: 10024.926041  Valid Acc: 0.6836014676827548
```

(d) We choose k = 10, lr = 0.1, number of iterations is 10, below is the reports of how the train losses and valid accuracy changes as a function of epoch. The final test accuracy is 67.796%



(e) We use the hyperparameters selected from part (d), and we finally choose $\lambda = 0.001$
The codes showing the different λ values are below:

```
lamb =  0.001
Epoch: 0    Training Cost: 13640.410563  Valid Acc: 0.6316680779000847
Epoch: 1    Training Cost: 12434.919973  Valid Acc: 0.6404177250917301
Epoch: 2    Training Cost: 11850.622645  Valid Acc: 0.6541066892464014
Epoch: 3    Training Cost: 11380.859469  Valid Acc: 0.6679367767428732
Epoch: 4    Training Cost: 11019.160830  Valid Acc: 0.6758396838837144
Epoch: 5    Training Cost: 10750.421609  Valid Acc: 0.6775331639853232
Epoch: 6    Training Cost: 10521.500075  Valid Acc: 0.6783799040361276
Epoch: 7    Training Cost: 10332.571808  Valid Acc: 0.6831780976573525
Epoch: 8    Training Cost: 10175.709025  Valid Acc: 0.6836014676827548
Epoch: 9    Training Cost: 10046.590349  Valid Acc: 0.6831780976573525
Final test Accuracy is  0.6827547276319503
Final valid Accuracy is  0.6831780976573525
```

```
lamb =  0.01
Epoch: 0    Training Cost: 14126.851028  Valid Acc: 0.68077900084674
Epoch: 1    Training Cost: 13050.539594  Valid Acc: 0.6735817104149027
Epoch: 2    Training Cost: 12630.059137  Valid Acc: 0.6725938470222975
Epoch: 3    Training Cost: 12445.849261  Valid Acc: 0.6686423934518769
Epoch: 4    Training Cost: 12374.213453  Valid Acc: 0.6682190234264748
Epoch: 5    Training Cost: 12327.976950  Valid Acc: 0.6642675698560542
Epoch: 6    Training Cost: 12287.517870  Valid Acc: 0.6652554332486593
Epoch: 7    Training Cost: 12254.277429  Valid Acc: 0.66539655659046
Epoch: 8    Training Cost: 12239.916571  Valid Acc: 0.665961049957663
Epoch: 9    Training Cost: 12219.350476  Valid Acc: 0.6651143099068586
Final test Accuracy is  0.6694891335026814
Final valid Accuracy is  0.6651143099068586
```

```
lamb =  0.1
Epoch: 0    Training Cost: 13516.206206  Valid Acc: 0.6116285633643804
Epoch: 1    Training Cost: 12680.307165  Valid Acc: 0.6147332768839966
Epoch: 2    Training Cost: 12559.280609  Valid Acc: 0.6196725938470223
Epoch: 3    Training Cost: 12497.143413  Valid Acc: 0.6240474174428451
Epoch: 4    Training Cost: 12463.670854  Valid Acc: 0.6243296641264465
Epoch: 5    Training Cost: 12443.763682  Valid Acc: 0.6237651707592435
Epoch: 6    Training Cost: 12431.066277  Valid Acc: 0.6241885407846458
Epoch: 7    Training Cost: 12422.483208  Valid Acc: 0.6243296641264465
Epoch: 8    Training Cost: 12416.389147  Valid Acc: 0.6239062941010444
Epoch: 9    Training Cost: 12411.881547  Valid Acc: 0.6236240474174428
Final test Accuracy is  0.6260231442280553
Final valid Accuracy is  0.6236240474174428
```

```
lamb =  1
Epoch: 0    Training Cost: 12409.520616  Valid Acc: 0.6240474174428451
Epoch: 1    Training Cost: 12407.641020  Valid Acc: 0.6240474174428451
Epoch: 2    Training Cost: 12406.325869  Valid Acc: 0.6239062941010444
Epoch: 3    Training Cost: 12405.258508  Valid Acc: 0.6239062941010444
Epoch: 4    Training Cost: 12404.372724  Valid Acc: 0.6239062941010444
Epoch: 5    Training Cost: 12403.623495  Valid Acc: 0.6239062941010444
Epoch: 6    Training Cost: 12402.979293  Valid Acc: 0.6239062941010444
Epoch: 7    Training Cost: 12402.417243  Valid Acc: 0.6239062941010444
Epoch: 8    Training Cost: 12401.920736  Valid Acc: 0.6239062941010444
Epoch: 9    Training Cost: 12401.477654  Valid Acc: 0.6239062941010444
Final test Accuracy is  0.6248941574936494
Final valid Accuracy is  0.6239062941010444
```

When $\lambda = 0.001$, we get the highest accuracy:
Final test Accuracy is 0.6827547276319503
Final valid Accuracy is 0.6831780976573525
The model performs slightly better with the regularization penalty

4 Ensemble

First, we randomly sampled the training data with np.random.randint with replacement three times. Note that the tuned hyperparameters from q3 was used in training the base model again. Next, we trained the base model each using the three resample training data from previous step with the same hyperparameters we obtained from q3. Lastly, we averaged the predictions and yields a binary prediction with threshold ≥ 0.5, the averaged prediction is now evaluated for the validation data and the test data.

```
Final test Accuracy is  0.5979866403236428
Final valid Accuracy is  0.5940351867532223
```

The performance (test accuracy and valid accuracy) after applying bagging is worse than the performances in q3 without bagging. One possibility could be that the standard model from q3 is overfitting the current data points, which yields a relatively high performance. After applying bootstrapping method, the stability of the algorithm has improved, and it is less biased, in exchange, the performance is lower. This in turn would lower the overfitting of the data giving which, in this case, lowers accuracy of the model.

5 PartB

5.a Formal Description

We tried to modify the Neural network model from question 3(ii) in order to improve the test accuracy of the answers.
Firstly, we added an extra hidden layer to the model in autoencoder model, given a user $v \in R^{N_{questions}}$ from a set of users S, new objective is:

$$\min_{\theta} \sum_{v \in S} ||v - f(v;\theta)||_2^2 + \frac{\lambda}{2}(||W^{(1)}||_F^2 + ||W^{(2)}||_F^2 + ||W^{(3)}||_F^2)$$

and

$$f(v;\theta) = h(W^{(3)} s(W^{(2)} g(W^{(1)} + b^{(1)}) + b^{(2)}) + b^{(3)})$$

The added hidden layer is used in order to reduce under fitting of the data since less hidden layers may cause the algorithm to not fit the data as well as it could. Thus adding an extra hidden layer could cause the algorithm to extract more information from the data set, which could reduce under-fitting and even improve test accuracy. Below is the module with new layer. We add a new function in Autoencoder, and the way we get weight norm.

```
self.g = nn.Linear(num_question, k)
self.s = nn.Linear(k, k)
self.h = nn.Linear(k, num_question)
```

```
g_w_norm = torch.norm(self.g.weight, 2) ** 2
h_w_norm = torch.norm(self.h.weight, 2) ** 2
s_w_norm = torch.norm(self.s.weight, 2) ** 2
return g_w_norm + s_w_norm + h_w_norm
```

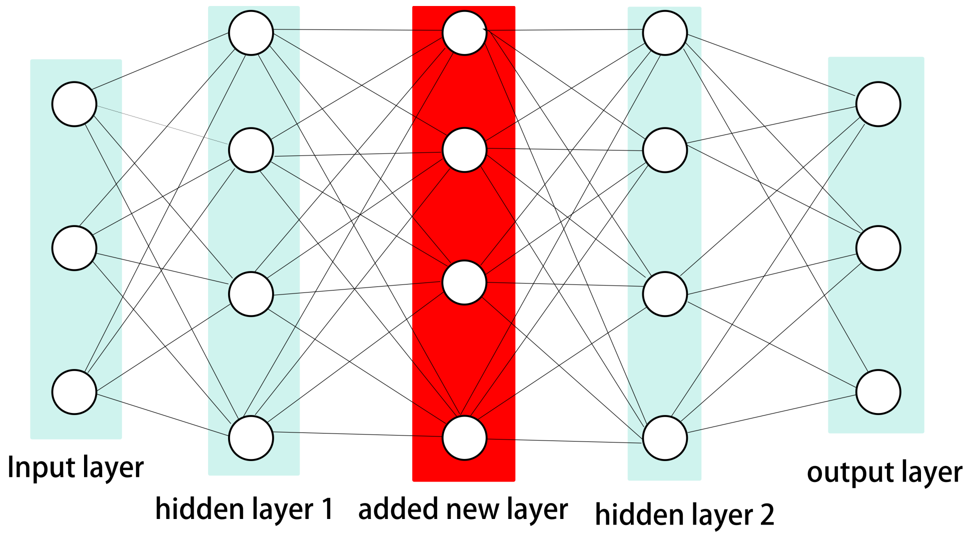
Then we tried to use meta data provided in data folder. We want to split the data into multiple groups based on associated features. From student_meta.csv, we use gender as a significant feature in neural networks. We split valid data into three groups by gender (male, female, unspecified), and train them separately. We did this in order to improve the accuracy on the training data and reduce under fitting.

Neural networks algorithm's performance also depends on the quality of features. If we got better features, then we would have better accuracy.

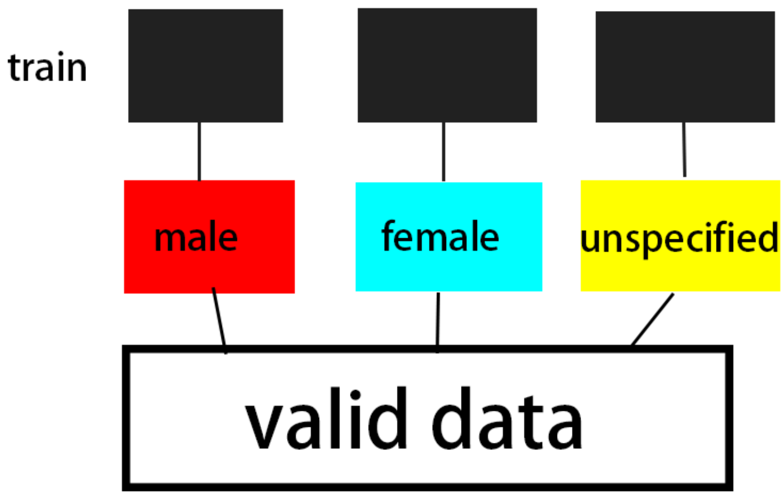
After these, we tuned the hyperparameters: $k \leftarrow 50, lr \leftarrow 0.03, num\_epoch \leftarrow 40,$

5.b Figure and Diagram

Below is the basic diagram about the new layer we added

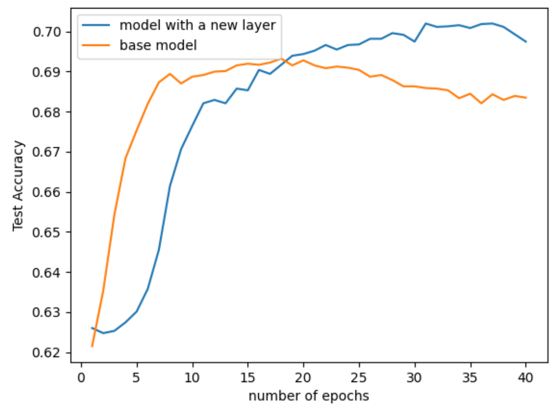


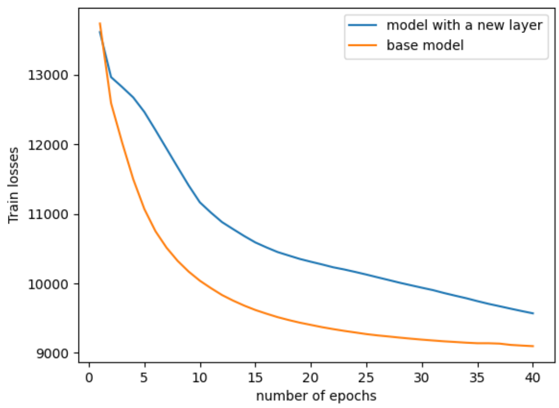
Below is a brief illustration for using meta data:



5.c Comparison and Demonstration

Firstly, we add a new layer in Neural network.
Below is the diagram of validation accuracy and training loss changes as a function of epoch
Note that for base model, we use the original hyperparameters except the number of epochs



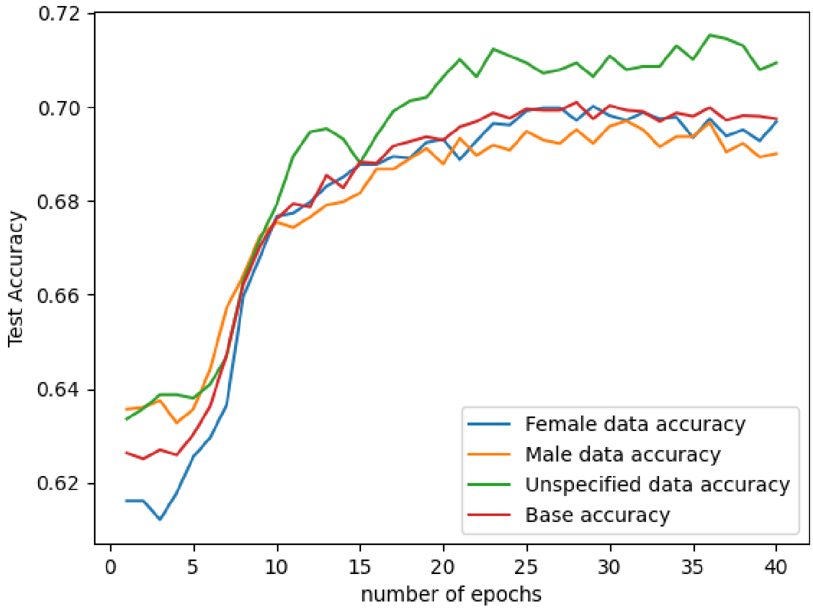


After adding a layer, the accuracy of the model has improved compared to the original. However, the train loss of the improved model is higher. Table below should the comparison of accuracy with baseline models and the modified models.

Model	Test Accuracy	Validation Accuracy
improved neural networks	0.69292	0.69588
KNN(user)	0.68417	0.68953
KNN(item)	0.68163	0.69221
Item Response	0.70703	0.70604
Neural networks	0.68275	0.68318

From above figure, it can be seen that the accuracy of the improved neural networks is higher than all algorithms except item response.

Then we move to step 2: using meta data to get associated features and split valid data by features, then train them separately to get the accuracy of each group.
The validation data will be separated into 3 parts: female, male, unspecified gender
Below is the accuracy of each groups:



Below are the final validation and test accuracy

Data	Test Accuracy	Validation Accuracy
Female	0.69517	0.69842
Male	0.69771	0.69517
unspecified	0.70574	0.70312
base	0.69827	0.69573

The data of unspecified gender yields a much higher accuracy than the original data.

5.d Limitation

In modifying our NN algorithm from part A q3, we have attempted different ways to improve the performance, this includes testing a different activation function (ReLU) and adding a new hidden layer. After testing new activation functions (i.e., sigmoid and ReLU), the sigmoid activation seems to yield the best performance with 68% accuracy while ReLU only got 50% around. This does not mean that other activation functions are bad, it simply means that the chosen model helps generalize our current data and to differentiate the outputs better than other functions.
In terms of adding a new hidden layer to our NN algorithm, it is a tedious process where the hyperparameters must be re-adjusted for each newly added hidden layer to our implementation. Given that there are many more activation functions that we can apply for our NN algorithm, we could try different activation functions other than the ones we have already tested as a possible way improve the performance. Other methods such as trying bigger variety of k values and adding more hidden layers could be some possible ways to improve the performance of the current NN models.

6 Contributions

Part A

- Question 1:Jonathan Chung
- Question 2:Shiven Taneja
- Question 3:Yuyang Chen
- Question 4:Jonathan Chung

Part B

- Formal Description:Shiven Taneja
- Figure or Diagram:Yuyang Chen
- Comparison or Demonstration:Yuyang Chen
- Limitations:Jonathan Chung