CSC311 - Final Assignment

Zelong Liu, Fizzah Mansoor, Harrison Deng

December 3, 2021

Contents

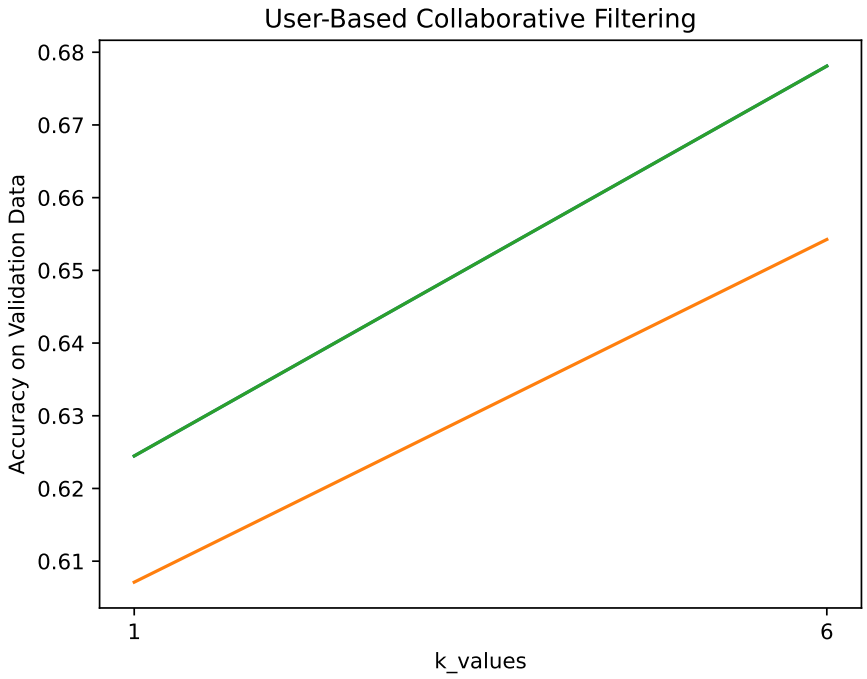
Part I
Predicting Student Correctness

1 K-Nearest Neighbor

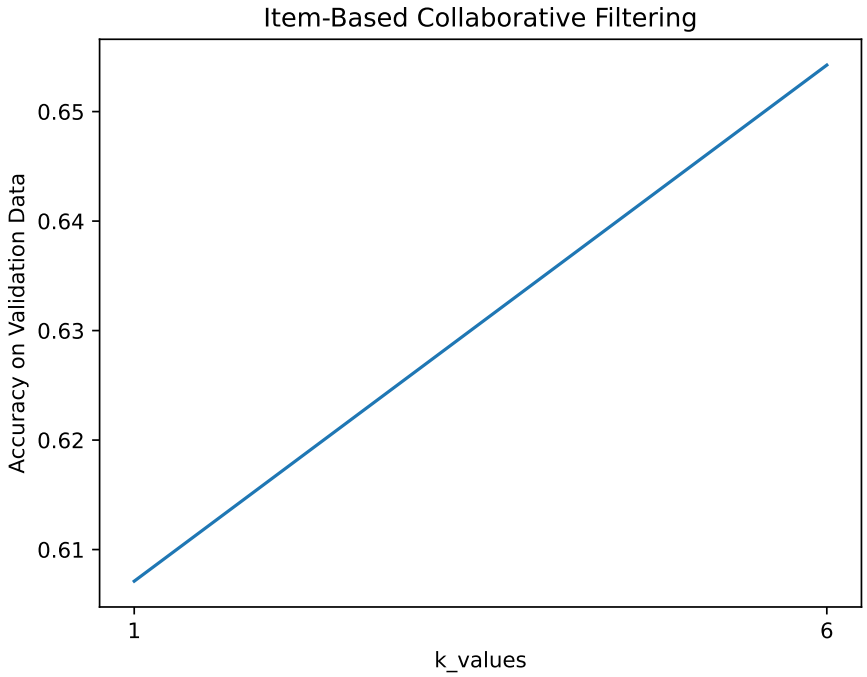
a Complete Main kNN, Plot and Report Accuracy

The implementation of all code is in the `part_a/knn.py` file. Following are plots of accuracy on validation data as a function of $k \in \{1, 6, 11, 16, 21, 26\}$:

```
import codebase.part_a.knn as knn
k_vals, val_user_acc, val_item_acc = knn.main("./codebase/data")

knn.accuracy_plot(k_vals, val_user_acc, "User-Based Collaborative
↪  Filtering")
```




```
knn.accuracy_plot(k_vals, val_item_acc, "Item-Based Collaborative
↪  Filtering")
```



b Selecting k*

c Implementing Impute by Item

Underlying assumption: if answers by certain users to Question A match those of Question B, then As answer correctness corresponding to a specific user matches that of question Y.

d Comparing user and item based Collaborative Filtering

User-Based collaborative filtering performs better on test data. 68.416% accuracy on user-based filtering and 68.162% accuracy on item-based filtering.

e Potential Limitations of kNN in this Context

We can safely assume that there is a high correlation between both question difficulty and student ability on whether or not the question was answered correctly. But, feature importance is not possible for the KNN algorithm (there is no way to define the features which are responsible for the classification), so it will not be able to make accurate inferences based on these two parameters. In the algorithm used in this question, either one of the two parameters (user

ability or question difficulty) is focused on, so it has lower validation and test accuracy scores than other algorithms in Part A of this project.

KNN runs slowly. Finding the optimal k-value from the given list of possible k values (1, 6, 11, 21, 26) takes several minutes for each function.

2 Item Response Theory

a Mathematical Derivations for IRT

We are given that $p(c_{ij} = 1|\boldsymbol{\theta}, \boldsymbol{\beta})$. We will assume c_{ij} is a value in $\boldsymbol{C}$ where i and j as coordinates are in set O as defined:

$$O = \{(i, j) : \text{Entry } (i, j) \text{ of matrix } \boldsymbol{C} \text{ and is observed}\}$$

.

Since this c_{ij} is a binary value, we can describe $P(\boldsymbol{C}|\boldsymbol{\theta}, \boldsymbol{\beta})$ with a bernoulli distribution:

$$p(C|\boldsymbol{\theta}, \boldsymbol{\beta}) = \prod_{ij} [\frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)}]^{c_{ij}} [\frac{1}{1 + exp(\theta_i - \beta_j)}]^{(1-c_{ij})}$$

Therefore, our Likelihood function is:

$$L(\boldsymbol{\theta}, \boldsymbol{\beta}) = \prod_{ij} [\frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)}]^{c_{ij}} [\frac{1}{1 + exp(\theta_i - \beta_j)}]^{(1-c_{ij})}$$

Then, apply log to obtain the log-likelihood where N and M are the number of users and questions respectively:

$$\begin{aligned}
L(\boldsymbol{\theta}, \boldsymbol{\beta}) &= \prod_{ij} [\frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)}]^{c_{ij}} [\frac{1}{1 + exp(\theta_i - \beta_j)}]^{(1-c_{ij})} \\
log(L(\boldsymbol{\theta}, \boldsymbol{\beta})) &= \log(\prod_{ij} [\frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)}^{c_{ij}}] [\frac{1}{1 + exp(\theta_i - \beta_j)}^{1-c_{ij}}] \\
&= \sum_{i=1}^{N} \sum_{j=1}^{M} \log([\frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)}^{c_{ij}}] [\frac{1}{1 + exp(\theta_i - \beta_j)}^{1-c_{ij}}]) \\
&= \sum_{i=1}^{N} \sum_{j=1}^{M} c_{ij}((log(exp(\theta_i - \beta_j)) - log(1 + exp(\theta_i - \beta_j))) \\
&+ (1 - c_{ij})(\log(1) - \log(1 + exp(\theta_i - \beta_j))) \\
&= \sum_{i=1}^{N} \sum_{j=1}^{M} [c_{ij}(\theta_i - \beta_j) - \log(\frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)})]
\end{aligned}$$

Then, we solve for the partial derivative with respect to θ_i and β_j respectively:

$$\frac{\delta}{\delta \theta_i} = \sum_{j=1}^{M} [c_{ij} - \frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)}]$$

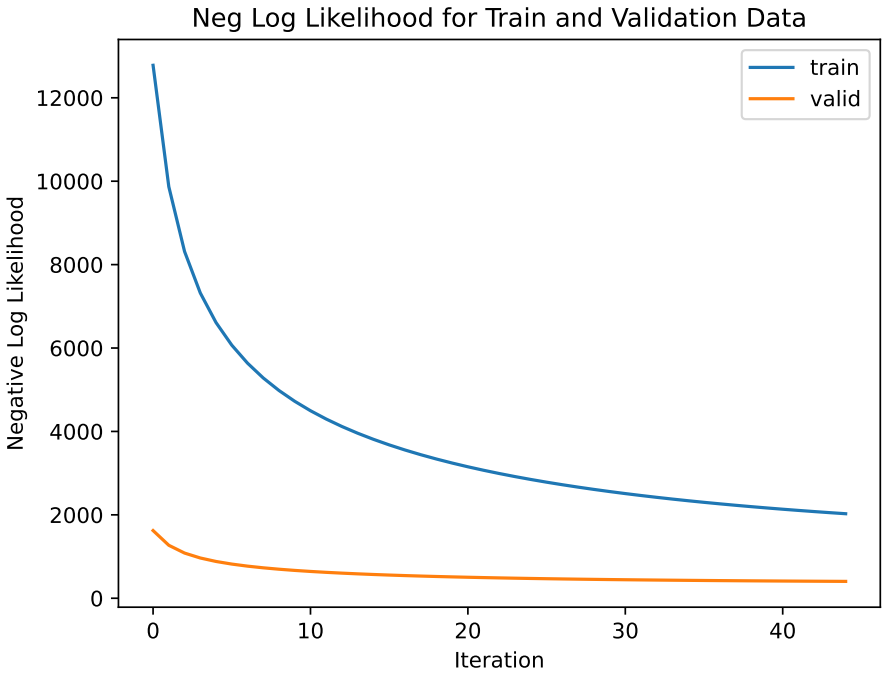
$$\frac{\delta}{\delta \beta_j} = \sum_{i=1}^{N} [-c_{ij} + \frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)}]$$

b Implementation of IRT

The implementation of IRT is in `part_a/item_response.py`. We chose the hyperparameters α and iterations number by performing multiple combinations of them and seeing which one had the highest validation score (automated, see mentioned code file for this automation). We then manually adjusted the set of tested values and repeated. Doing this a few times resulted in:

```
import codebase.part_a.item_response as irt
print()
irt_results = irt.main("./codebase/data")
```

```
Training with lr of 0.005 and 45 number of iterations.
Final accuracy: 0.7060400790290714
lr*: 0.005 iterations*: 45 acc*: 0.7060400790290714
test accuracy: 0.7019475021168501
```



c Reporting Validation and Test Accuracies

Validation and test accuracies have been calculated in the previous call to the main function. Again, implementation is in `part_a/item_response.py`.

Our validation accuracy:

```
print(irt_results["val_acc"])
```

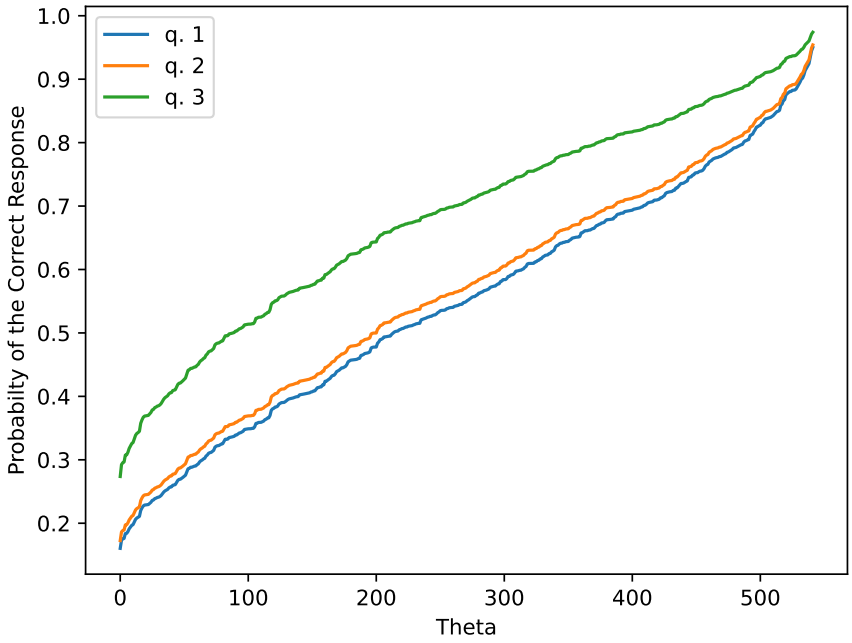
```
0.7060400790290714
```

Test accuracy:

```
print(irt_results["test_acc"])
```

```
0.7019475021168501
```

d Plots of Questions With Respect to θ and β



3 Neural Networks

a Differences Between ALS and Neural Networks

als optimizes 2 variable U and Z, neural net optimizes 1 variable W (with gradient descent),

ALS is an optimization algorithm that is incorporated as a part of a machine learning algorithm, while the neural network is a machine learning algorithm that that uses optimization algorithms to achieve learning.

in als, w is used to manipulate x while in als, x is completely gone.

b Implementing AutoEncoder

This part can be found in `part_a/neural_network.py`.

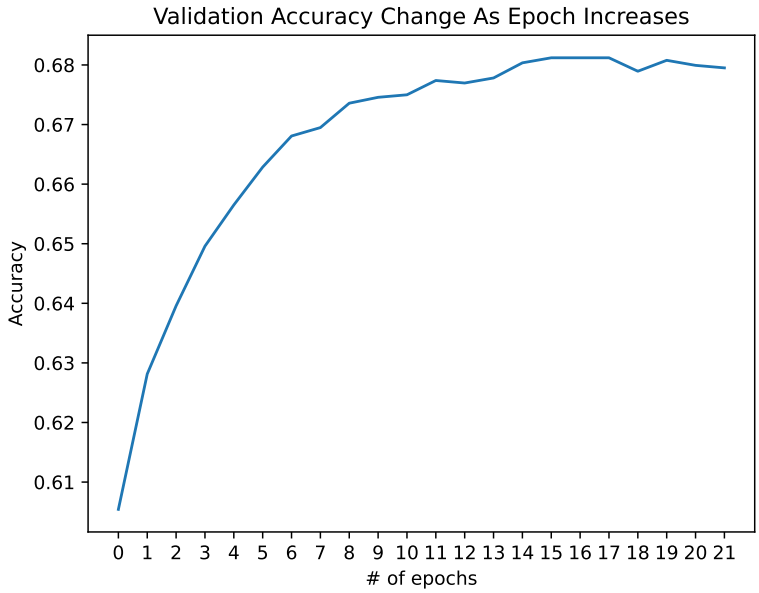
c Tuning and Training NN

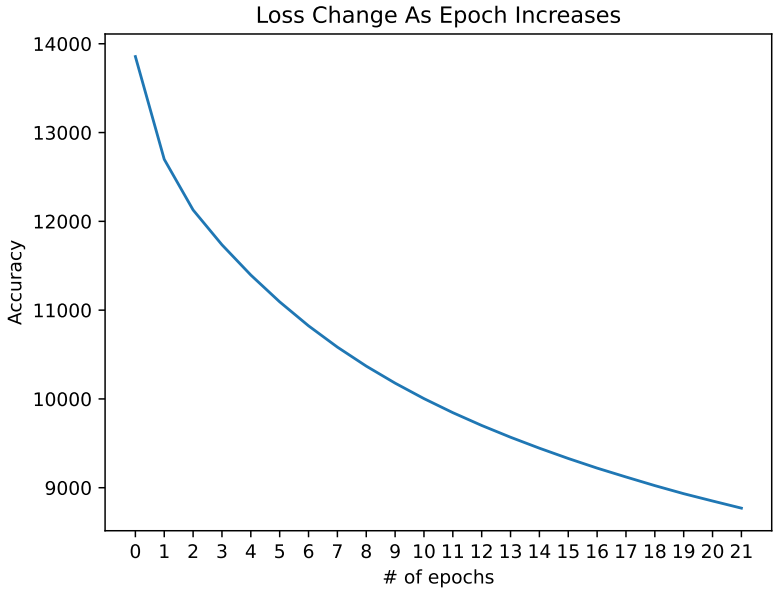
```
import codebase.part_a.neural_network as nn
nn.main(data_path="./codebase/data", verbosity = 1)
```

```
Starting k, learning rate, number of epochs, and lambda optimizations.
 k Learn. Rate # of Epochs Lambda
 10 0.05 22 0.00025

Final training results:

Epoch: 22/22, Loss: 8769.933279, Valid acc.: 0.6795088907705334
 Final Validation Accuracy* 0.6795088907705334
 Final Test Accuracy* 0.6852949477843635
Concluding learning rate, number of epoch and lambda optimizations.
Results:
Final acc.: 0.6795088907705334
 k*: 10
 Learning rate*: 0.05
 Number of epochs*: 22
 Lambda*: 0.00025
```

d Plotting and Reporting

The following are the plots generated:





e Implementing L_2 Regularization

L_2 regularization has been implemented in the same code file (`part_a/neural_network.py`).

```
lamb=0
```

```
Final Validation Accuracy*    0.6858594411515665
Final Test Accuracy*    0.6836014676827548
lamb=0.001
Final Validation Accuracy*    0.6869884278859724
Final Test Accuracy*    0.6785210273779283


(optional extra finding):
but with lamb=0.00025
Final accuracy    0.6848715777589613
Final Test Accuracy*    0.6861416878351679
```

There are improvements on the validation accuracy, but not on the test accuracy.

4 Ensemble

Part II
Modifying for Higher Accuracy

5 Formal Description

6 Figure or Diagram

7 Comparison or Demonstration