# CSC311 Final Project

Ian Huang, Benjamin Liu

## Part A

1. **$k$-Nearest Neighbor**

   **a**. See Figure 1 and Table 1.

   **b**. See Table 1.

   **c**. See Table 1. The underlying assumption for item-based similarity is that for two questions **a** and **b** with answers $a_1, \ldots, a_n$ and $b_1, \ldots, a_n$ for each student $i \in \{1, \ldots, n\}$, if $a_i = b_i$ for most $i \neq j$, then it is likely that $a_j = b_j$ as well.

   **d**. $k$-NN by user distance marginally outperformed $k$-NN by item distance based on the resulting test accuracies for the respective experimentally derived values for $k^*$.

   **e**. There are a number of potential limitations with this method. For instance, the assumption described above for item-based filtering and the similar assumption for user-based filtering may not hold. Additionally, another potential problem with this method is that of data scarcity. The sparse matrix from which item/user distance is computed is around 94% missing values[1], which may limit the efficacy of the underlying NaN-Euclidean distance metric used in the $k$-NN implementation.

| $k$ | `knn_impute_by_user` | `knn_impute_by_item` |
|---|---|---|
| 1 | 0.624 | 0.607 |
| 6 | 0.678 | 0.654 |
| 11 | **0.690** | 0.683 |
| 16 | 0.676 | 0.686 |
| 21 | 0.669 | **0.692** |

Table 1: Data represented in Figure 1. Validation accuracy with respect to $k$ for distance by user and by item. The highest validation accuracies were obtained by $k^* = 11$ and $k^* = 21$ for user and input similarity respectively for which the test accuracies were 0.684 and 0.682 respectively.

2. **Item Response Theory**

   **a**. Given that

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

[1] Calculated by `np.count_nonzero(np.isnan(sparse_matrix)) / sparse_matrix.size`

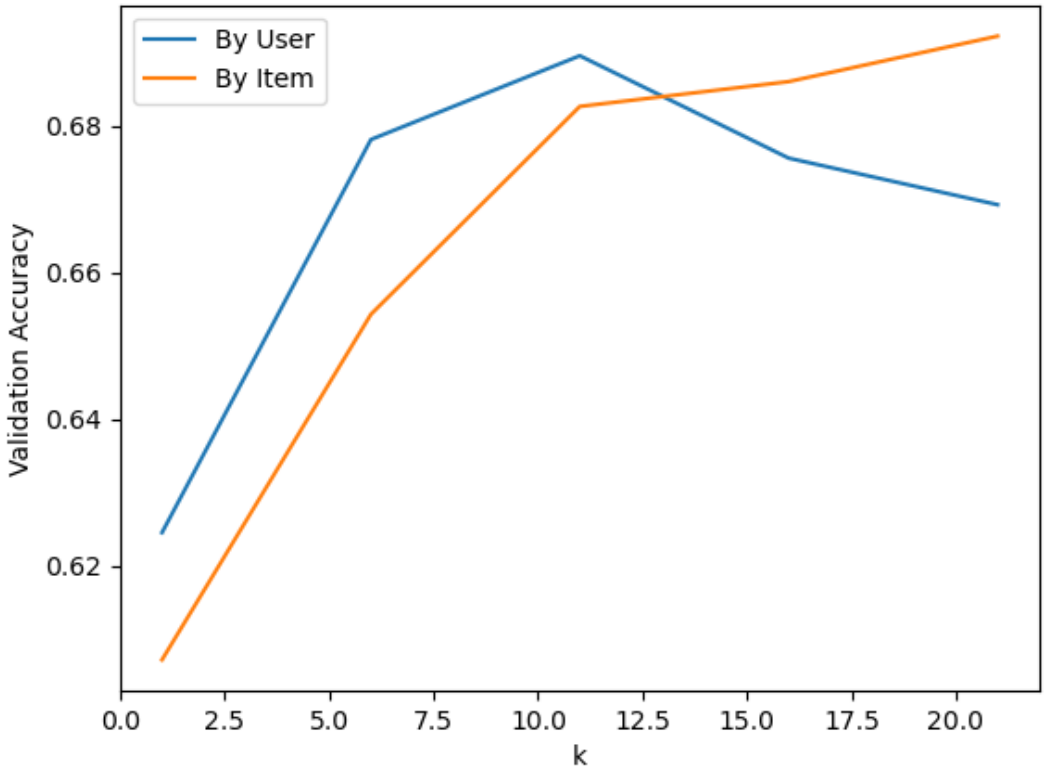



Figure 1: Validation accuracy with respect to $k$.

the (positive) log-likelihood $\log p(c_{ij} \mid \theta_i, \beta_j)$ is given by:

$$\begin{aligned} \ell &\stackrel{\text{def}}{=} \log p(c_{ij} \mid \theta_i, \beta_j) \\ &= \log\left(z^{c_{ij}}(1-z)^{1-c_{ij}}\right) \\ &= c_{ij}\log(z) + (1-c_{ij})\log(1-z). \end{aligned} \tag{1}$$

Using (1), we have

$$\begin{aligned} \log p(\mathbf{C} \mid \boldsymbol{\theta}, \boldsymbol{\beta}) &= \log \prod_i \prod_j p(c_{ij} \mid \theta_i, \beta_j) \\ &= \sum_i \sum_j \log p(c_{ij} \mid \theta_i, \beta_j) \\ &= \sum_i \sum_j \ell = \sum_i \sum_j c_{ij}\log(z) + (1-c_{ij})\log(1-z). \end{aligned}$$

The partial derivatives of $\ell$ with respect to $\theta_i$ and $\beta_j$ are given by[2]

$$\begin{aligned}\frac{\partial \ell}{\partial \theta_i} &= \frac{\partial \ell}{\partial z}\frac{\partial z}{\partial \theta_i} \\ &= \left(\frac{c_{ij}}{z} - \frac{1-c_{ij}}{1-z}\right) z(1-z) \\ &= c_{ij}(1-z) - z(1-c_{ij}) \\ &= c_{ij} - c_{ij}z - z + c_{ij}z \\ &= c_{ij} - z,\end{aligned} \qquad \begin{aligned}\frac{\partial \ell}{\partial \beta_j} &= \frac{\partial \ell}{\partial z}\frac{\partial z}{\partial \beta_j} \\ &= \left(\frac{c_{ij}}{z} - \frac{1-c_{ij}}{1-z}\right)(-z(1-z)) \\ &= \dots \\ &= -c_{ij} + z.\end{aligned}$$

**b**. The parameters $\boldsymbol{\theta}$ and $\boldsymbol{\beta}$ were randomly initialized element-wise using a uniform distribution over $[-0.1, 0.1]$ and optimized using the update rule[3]

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \frac{\partial \ell}{\partial \boldsymbol{\theta}} \quad \boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \alpha \frac{\partial \ell}{\partial \boldsymbol{\beta}}$$

with learning rate $\alpha = 0.001$ over 50 iterations. See Figures 2 and 3 for detailed results.

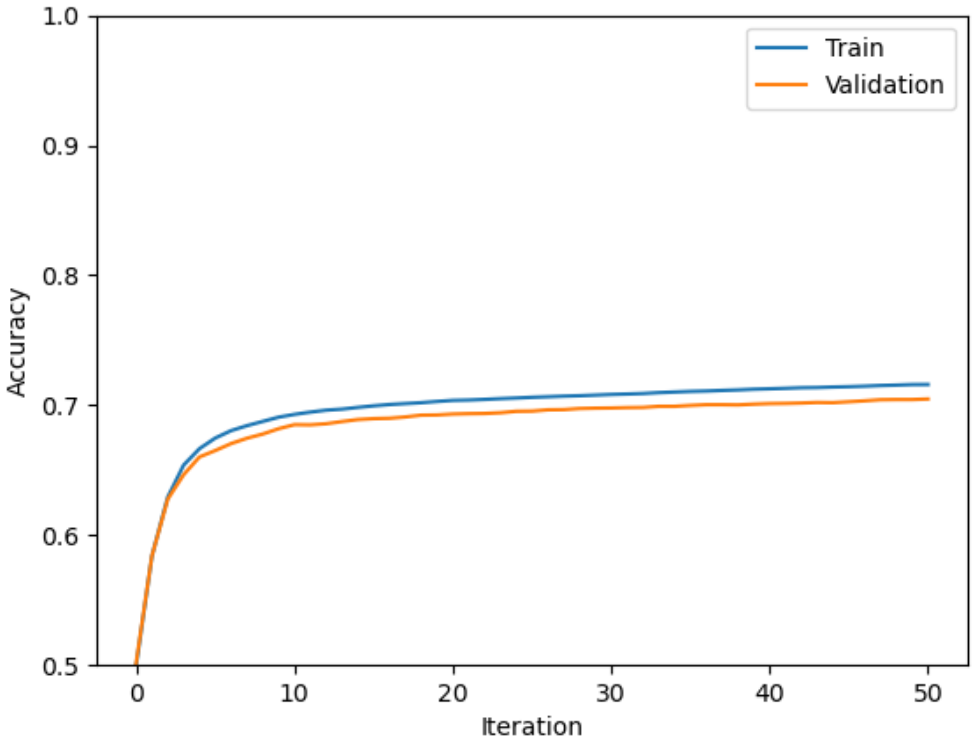
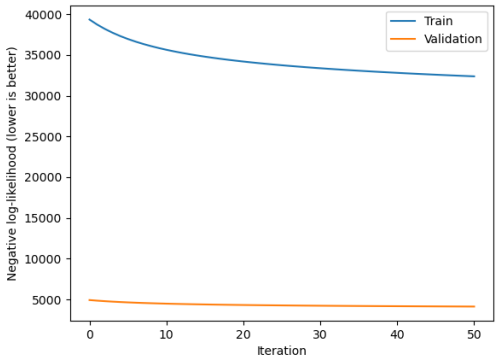


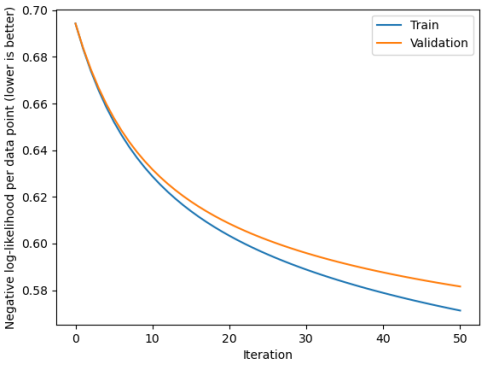

Figure 2: Training and validation accuracy over time.

**c**. The final training and validation accuracies after 50 iterations of training (see above) were 71.6% and 70.3% respectively[4].

[2] Since $z$ is the sigmoid function, we have $\partial z/\partial \theta_i = z(1-z)$ and $\partial z/\partial \beta_j = -z(1-z)$.
[3] In each iteration, $\boldsymbol{\theta}$ is updated before $\boldsymbol{\beta}$.
[4] According to the Faculty of Arts and Science, this is equivalent to a B-




(a) Absolute negative log-likelihood. Note that the validation set, being significantly smaller, has a much smaller negative log-likelihood.




(b) Negative log-likelihood, normalized by number of data-points.

Figure 3: Negative training and validation log-likelihood over time (lower is better).

**d**. See Figure 4. Note the S-shape of the curve resulting from the use of the sigmoid function with respect to a linear variation over its parameter $\theta - \beta_j$. These plots may be interpreted as the predicted probability of success on a question with a measure of difficulty corresponding to $\beta_j$ with respect to student competency measured by $\theta$.

3. **Neural networks**

**a**. The differences between Alternating Least Squares and Neural Networks are as follows: Firstly, the purpose behind Alternating Least Squares is to reduce the dimensions of the input data into two smaller matrices meant to represent, in this example, questions and students. Neural Networks, on the other hand, try to represent the data as a set of latent features, not differentiated between question and student. Secondly, Alternating Least Squares alternates between optimizing each of the two latent feature matrices in order to factor the matrix, while Neural Networks primarily use the multivariate chain rule to calculate the gradient in order to backpropagate, thus updating all of the weights of the network at the same time in order to make predictions. Alternating least squares is thus a linear, more iterative process, while Neural Networks are nonlinear and more parallelizable. Thirdly, Alternating Least Squares is an unsupervised learning algorithm, while Neural Networks can be used for both supervised and unsupervised learning. In this sense, the scope of Neural Networks is much broader than that of Alternating Least Squares; neural networks can be used for a much wider set of tasks than just dimensionality reduction.

**b**. The forward pass was implemented according to the docstring.

**c**. For every combination of learning rate $\alpha \in \{0.01, 0.02, 0.03, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5\}$ over $\mathcal{E} \in \{10, 25, 50, 100\}$ iterations and $k \in \{10, 50, 100, 200, 500\}$, $k* = 50$ was able to most consistently yield the highest validation accuracy. The highest validation accuracy with $k* = 50$ was with a learning rate of 0.01 and 50 iterations, yielding a validation accuracy of 0.6814846175557437.

**d**. The final test accuracy was 0.6850127011007621. The training loss and validation accuracy over time are shown in Figure 5.

**e**. For $\lambda = 0.001$, the test accuracy was 0.6836014676827548 and the validation accuracy was 0.6865650578605701. The adjusted loss function achieved a higher validation accuracy than the

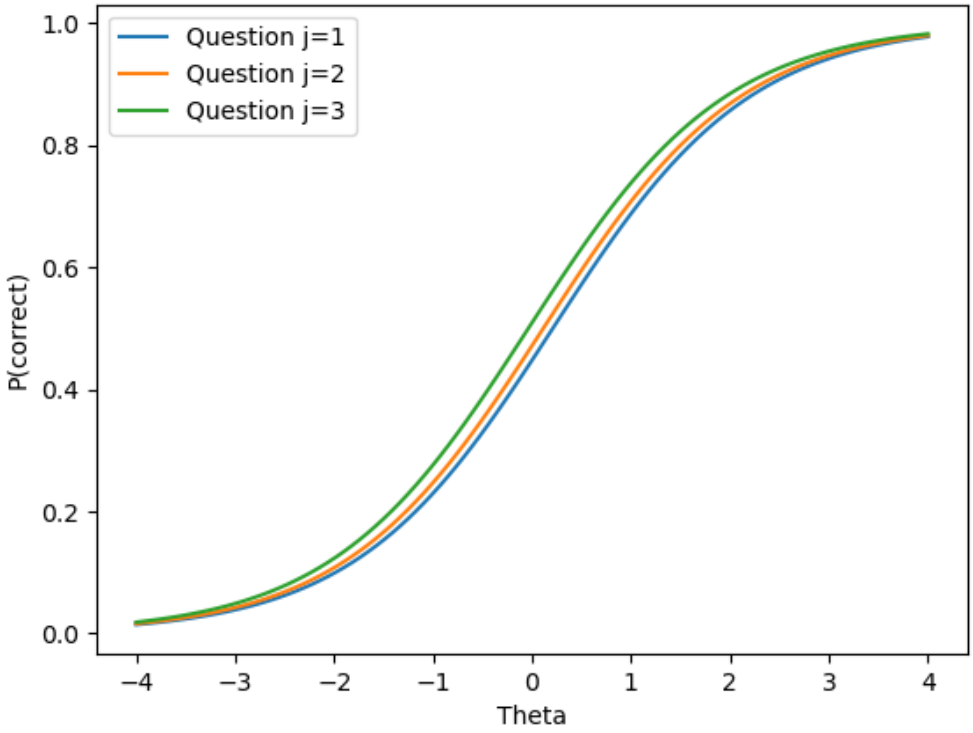
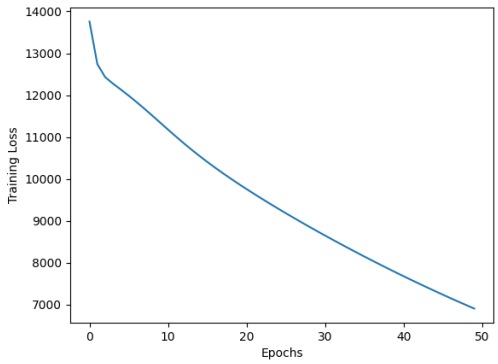
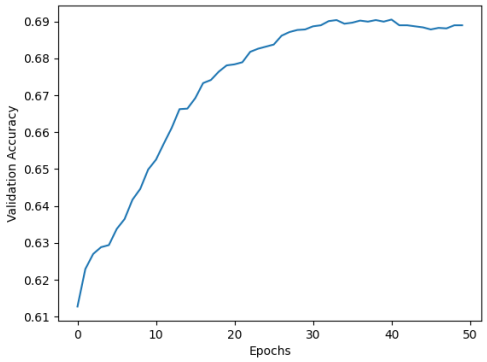



Figure 4: $p(\text{correct} \mid \theta, \beta_j) = \dfrac{\exp(\theta - \beta_j)}{1 + \exp(\theta - \beta_j)}$ with respect to $\theta$ for $j \in \{1, 2, 3\}$.



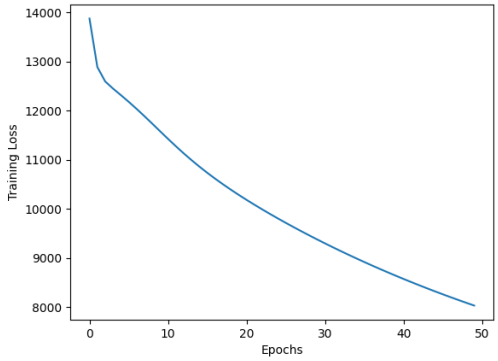

(a) Training loss over time without modified loss.



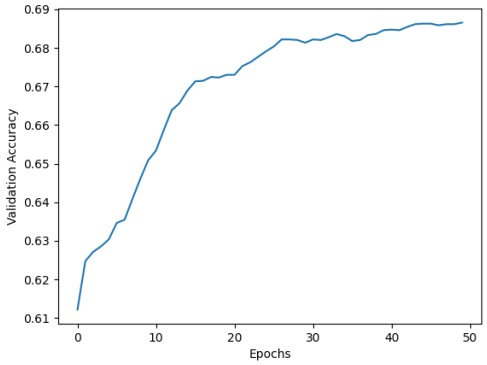

(b) Validation accuracy over time wihout modified loss.

Figure 5: Training loss and Validation accuracy over time without modified loss (lower loss and higher accuracy is better).

original loss function, however, it had slightly lower test accuracy than the original test function. The validation accuracy over time, as shown in 6, behaves more erratically than when using the unmodified loss function, while the training loss over time behaves similarly to that of the unmodified loss function, though due to the added term, in the loss function, is shifted upwards.




(a) Training loss over time with modified loss.




(b) Validation accuracy over time with modified loss.

Figure 6: Training loss and Validation accuracy over time with modified loss (lower loss and higher accuracy is better).