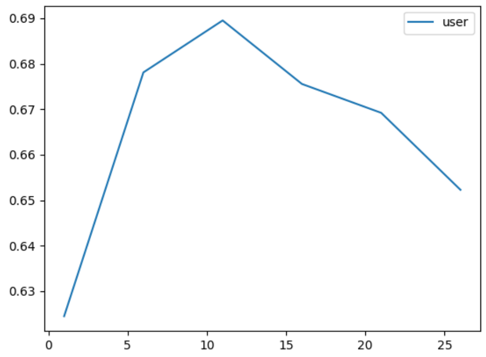
CSC311 Final Project

Jia Lin Yuan, Steven Tran, Luyang Shang

December 15, 2020

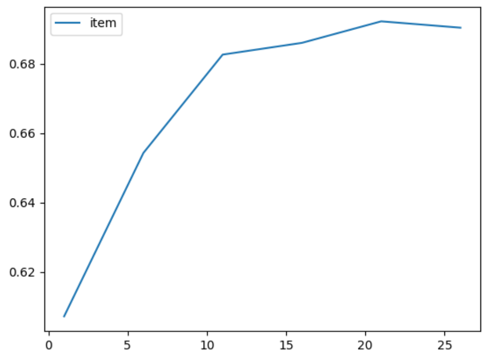
1 Part A

1. For this approach, we use the KNN algorithm to impute the missing values.
   (a) See `knn.py` for the supporting code. For distance by user, we have:



k	Validation Accuracy
1	0.6244707874682472
6	0.6780976573525261
11	0.6895286480383855
16	0.6755574372001129
21	0.6692068868190799
26	0.6522720858029918

   (b) The best value of k for user-based collaborative filtering is $k^* = 11$.
   (c) For distance by item, we have:



k	Validation Accuracy
1	0.607112616426757
6	0.6542478125882021
11	0.6826136042901496
16	0.6860005644933672
21	0.6922099915325995
26	0.69037538808919

   And the best value of k for item-based collaborative filtering is $k^* = 21$.
   (d) For the chosen values of k: $k^*_{\text{user}} = 11$, $k^*_{\text{item}} = 21$, we find that the test accuracy is 0.68416596105 and 0.68165274090, respectively. So the user-based collaborative filtering performs better.
   (e) Here are two limitations of KNN for this task:

- KNN is slow. Even with only 542 items and 1774 users it takes a while to predict.
- Using Euclidean distance, we consider distances in all dimension to be equal. For example if A and B's math skills are very different but english, physics, and other subjects are similar, the KNN will still predict A's math question similar to B's math questions (since skill in math is treated equally with other subjects).

2. In this approach, we use Item Response Theory to predict students' correctness to diagnostic questions.

   (a)

$$\mathcal{L}(\mathbf{C}|\boldsymbol{\theta},\boldsymbol{\beta}) = \sum_{i=1}^{N}\sum_{j=1}^{J} P(c_{ij}=1|\theta_i,\beta_j)^{c_{ij}}(1-P(c_{ij}=0|\theta_i,\beta_j))^{1-c_{ij}}$$

$$\ell(\mathbf{C}|\boldsymbol{\theta},\boldsymbol{\beta}) = \sum_{i=1}^{N}\sum_{j=1}^{J} c_{ij}\log(P(c_{ij}=1|\theta_i,\beta_j)) + (1-c_{ij})\log(1-P(c_{ij}=0|\theta_i,\beta_j))$$

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

$$= \sum_{i=1}^{N}\sum_{j=1}^{J}\left(c_{ij}(\theta_i-\beta_j)-\log(1+\exp(\theta_i-\beta_j))\right)$$

$$= \sum_{i:\,(i,j)\in C}\ \sum_{j:\,(i,j)\in C}\left(c_{ij}(\theta_i-\beta_j)-\log(1+\exp(\theta_i-\beta_j))\right)$$

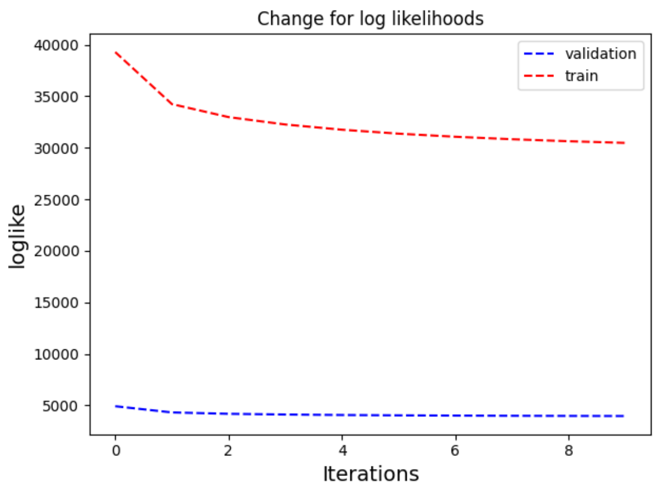
$$\frac{\partial \ell}{\partial \theta_i} = \sum_{j:\,(i,j)\in C} c_{ij} - \sum_{j:\,(i,j)\in C}\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta j)}$$

$$\frac{\partial \ell}{\partial \beta_j} = \sum_{i:\,(i,j)\in C} c_{ij} + \sum_{i:\,(i,j)\in C}\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta j)}$$

   (b) We found the following hyperparameters to be the best:

   - learning rate=0.016
   - number of iterations=10

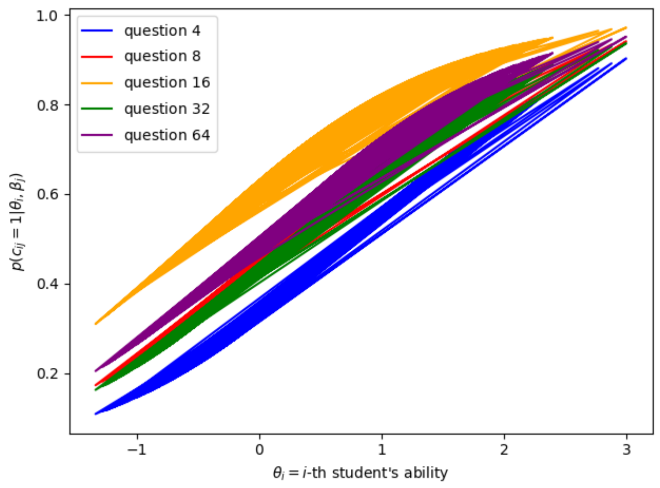
   Here is the plot of log-likelihoods on the training and validation data:



(c) For the same hyperparameters, we achieved:

- Final validation accuracy: 0.7064634490544736
- Final test accuracy: 0.7039232289020604

(d) The plot is shown below. explain what it represents



3. In this approach, we use neural networks.

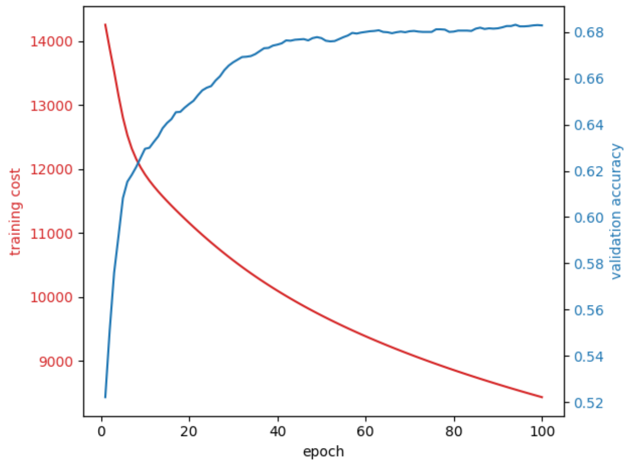
   (ii) Neural Networks

   (a) Not sure what this is asking.

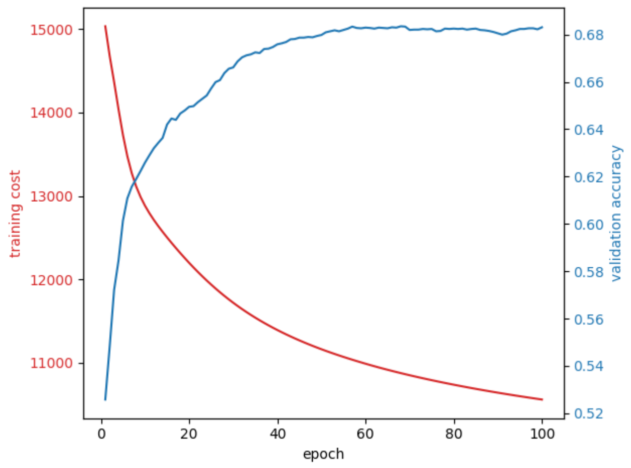
   (b) See `neural_network.py`

   (c) We trained models using a variety of hyperparameters and found that a good combination was k (latent dim.) $= 10$, learning rate $= 0.01$, epochs $= 100$. This achieved a training cost of 8376.565430 and validation accuracy of 0.6891052780129834.

   (d) The final test accuracy was 0.6838837143663562.



   (e) We chose $\lambda = 0.1$ and all of the other hyperparameters remain the same. This results in a final validation accuracy of 0.6830369743155518 and a test accuracy of 0.6850127011007621.



4. For our ensemble, we decided to use the following base models:

   (a) Item response theory model with hyperparameters: 10 iterations, 0.016 learning rate

   (b) ALS matrix factorization with hyperparameters: latent dimension 35, 0.01 learning rate, 500000 iterations

   (c) Random-splitting decision tree with $\log_2 n$ levels, i.e. it splits a node to be an internal node if there is at least 2 samples at that node.

   We also bootstrapped the training set by sampling three times with replacement, and then using one of those datasets for each of the models. The decisions of the three models are weighted equally, so the bagged model prediction was given by

$$\frac{\sum_{d \in \text{data}} (\text{pred}_{\text{IRT}}(d) + \text{pred}_{\text{ALS}}(d) + \text{pred}_{\text{tree}}(d))}{3}$$

   and then compared to the usual 0.5 threshold.

   The objective and reason why we employed this ensembling process was because we found it quite hard to push past the $\sim 0.68 - 0.70$ test accuracy threshold using only one base model without drastically overfitting on the training data. It was apparent through tuning of hyperparameters, especially for the neural network, that improvements may have been due to chance and that the models wouldn't generalize well to new data. In adding more predictors via bagging, we reduce the variance and reduce overfitting, but increase the bias.

   The results of our implementation are as follows:

   - Validation accuracy: 0.7039232289020604
   - Test accuracy: 0.694609088343212

2 Part B

1. We chose to improve upon our matrix factorization model because our testing in Part A yielded the most promising results with this model on the test data.

   We amended the initial squared error loss; the new objective function we aimed to minimize is [KBV09]:

$$\mathcal{J}_{b\lambda} := \frac{1}{2} \sum_{(n,m)\in O} \left(C_{nm} - \left(\mathbf{u}_n^\top \mathbf{z}_m + \mathbf{b}_{u_n} + \mathbf{b}_{z_m} + \mu \right)^2 \right) + \lambda \left(\mathbf{u}_n^\top \mathbf{u}_n + \mathbf{z}_m^\top \mathbf{z}_m + \mathbf{b}_{u_n} + \mathbf{b}_{z_m} \right)$$

   where:

   - $\mathbf{b}_{u_n}$ is a bias term for user n. It is initialized to the ratio of number of answers by user n to the total amount of observed data, i.e.

$$\mathbf{b}_{u_n} := \frac{|\{(n,m) \in O\}|}{\text{total observed data}}$$

   - $\mathbf{b}_{z_m}$ is a bias term for question m. It is initialized to the ratio of number of answers for question m to the total amount of observed data, i.e.

$$\mathbf{b}_{z_m} := \frac{|\{(n,m) \in O\}|}{\text{total observed data}}$$

   - μ is a scalar initialized to the ratio of correct data to the total amount of observed data, i.e.

$$\mu := \frac{\text{correct answers}}{\text{total observed data}}$$

   - λ is a penalty/regularization term

   We added weight regularization to the gradient descent method of optimizing the loss in our alternating least squares matrix factorization algorithm and bias/weight terms to every user and question. Intuitively, it is useful to weigh the users and questions that have more responses higher because they give more information, which is very useful when dealing with our issue of sparsity in the data. To start, given k, $\mathbf{u}$ and $\mathbf{z}$ were initialized by taking a uniform random sample of shape $(\#\ \text{user/question}, k)$ over $\left[0, \left\lfloor \frac{1}{\sqrt{k}} \right\rfloor\right]$. Then, we obtained the following update rules for $\text{argmin}_{\mathbf{u},\mathbf{z}}\, \mathcal{J}_{b\lambda}$ via stochastic gradient descent:

$$\begin{aligned}
\mathbf{u}_n' &\leftarrow \mathbf{u}_n - \alpha \left(- \left(c_{nm} - \mu - \mathbf{b}_{u_n} - \mathbf{b}_{z_m} - \mathbf{u}^\top \mathbf{z} \right) \mathbf{z} + \lambda \mathbf{u}_n \right) \\
\mathbf{z}_m' &\leftarrow \mathbf{z}_m - \alpha \left(- \left(c_{nm} - \mu - \mathbf{b}_{u_n} - \mathbf{b}_{z_m} - \mathbf{u}^\top \mathbf{z} \right) \mathbf{u} + \lambda \mathbf{z}_m \right) \\
\mathbf{b}_{u_n}' &\leftarrow \mathbf{b}_{u_n} - \alpha \left(-(c_{nm} - \mu - \mathbf{b}_{u_n} - \mathbf{b}_{z_m} - \mathbf{u}^\top \mathbf{z}) + \lambda \mathbf{b}_{u_n} \right) \\
\mathbf{b}_{z_m}' &\leftarrow \mathbf{b}_{z_m} - \alpha \left(-(c_{nm} - \mu - \mathbf{b}_{u_n} - \mathbf{b}_{z_m} - \mathbf{u}^\top \mathbf{z}) + \lambda \mathbf{b}_{z_m} \right)
\end{aligned}$$

   Furthermore, we took this training approach for six models, each trained differently with corresponding k values of $[40, 50, 60, 70, 80, 90]$. We did this in an attempt to reduce overfitting to the training data, reducing variance, and also increasing the bias. Their predictions were averaged and then compared to the usual threshold of 0.5, classifying the $(\text{user}, \text{question})$ pair as correct if it was greater. Another objective of this approach was to hopefully increase test performance by weighing more important points higher.

2.

3. In this section, we compare the performance of our model using the same initialization of $\mathbf{u}, \mathbf{z}$, learning rate, and number of iterations with the baseline ALS matrix completion model. We will consider the training loss and prediction accuracy on the test data to see if our changes were useful.

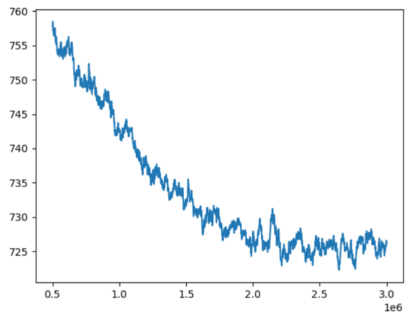


Figure 1: $k = 30$

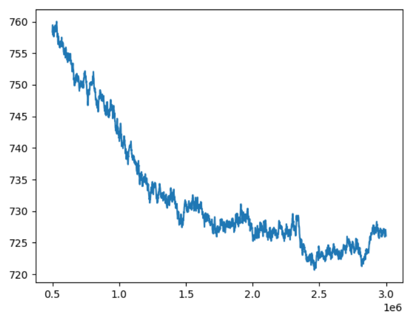


Figure 2: $k = 40$

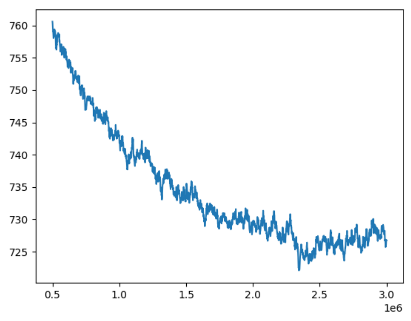


Figure 3: $k = 50$

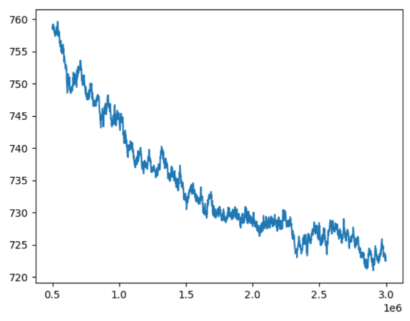


Figure 4: $k = 60$

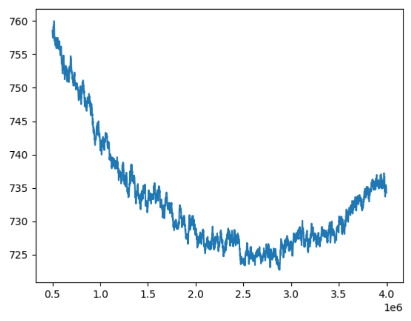


Figure 5: $k = 70$

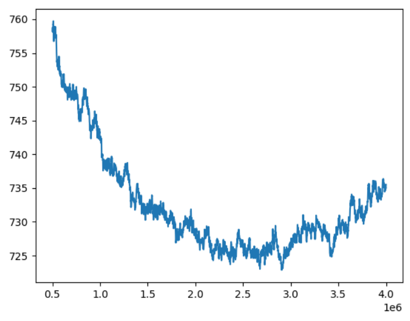


Figure 6: $k = 80$

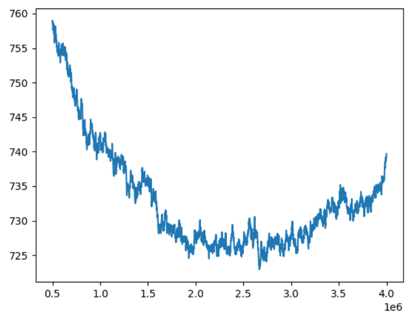


Figure 7: $k = 90$

4. Our method may perform poorly if the weights are roughly equal, since there wouldn't be a gain in this instance. Another shortcoming is the lack of appropriate metadata that could help with weighing the features. We also don't have a lot of data to work with, and that's likely the reason why most models we have attain only around 70% accuracy on the training, validation, and test sets.

3 References

[KBV09] Yehuda Koren, Robert Bell, and Chris Volinsky. "Matrix Factorization Techniques for Recommender Systems". In: Computer 42.8 (Aug. 2009), pp. 30–37. ISSN: 0018-9162. DOI: `10.1109/MC.2009.263`. URL: `https://doi.org/10.1109/MC.2009.263`.