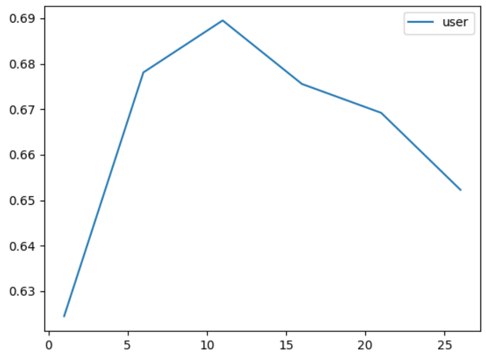
CSC311 Final Project

Jia Lin Yuan, Steven Tran, Luyang Shang

December 15, 2020

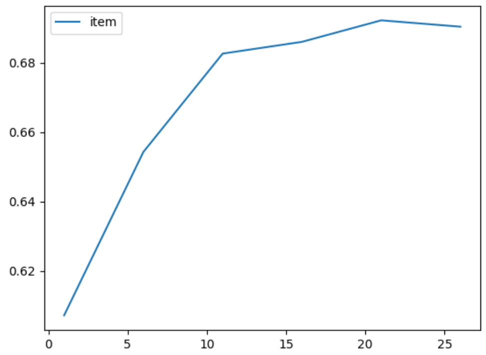
1 Part A

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. TODO: KNN for user has better test accuracy than item, Alan
   (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)

   (b)

   (c)

   (d)

3. In this approach, we use both matrix factorization and neural networks.

   (i) Matrix Factorization remove this if we are not using it

   (a) d

   (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 the best combinations were:

k (latent dim.)	learning rate	epochs	λ (penalty)	train cost	val. accuracy
10	0.01	100	0.1	10545.283	0.68784
10	0.01	100	0.5	19321.324219	0.69038
50	0.05	10	0.5	17700.197	0.690799
100	0.01	50	0.1	7667.879	0.68246
100	0.05	10	0	6923.447	0.68346
200	0.01	50	0.1	6638.1338	0.6782388
200	0.01	50	0.5	16577.74	0.6745696

   Setting $k = 500$ led to comparatively low validation accuracy (mean average of 0.651150). For the other values of k, there weren't discernible correlations between the choice of hyper-parameters and the performance of the model; the lower training cost and higher validation accuracy can probably be attributed to luck. We choose $k^* = 10$ since the combination on the second row has a validation accuracy of 0.69038.

   (d) d

2 Part B

1. d
2. d
3. d
4. d