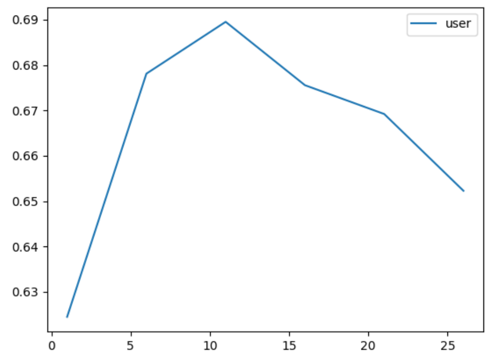
Prediction Models for Student Performance on Diagnostic Questions
CSC311 Final Project

Jia Lin Yuan, Steven Tran, Luyang Shang

December 15, 2020

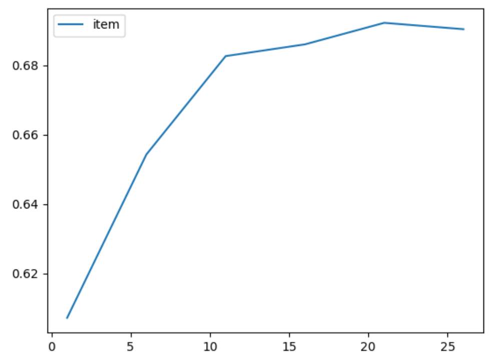
1 Part A

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 $k^* = 21$.

   An assumption made here is that questions that were answered similarly are equal in difficulty. This might not always be case, keeping in mind that questions may span completely different subjects and there are only so many ways to pose a question. A question in a introductory chemistry quiz may be answered similarly to a graduate-level statistics question if both of them are multiple-choice.

(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.

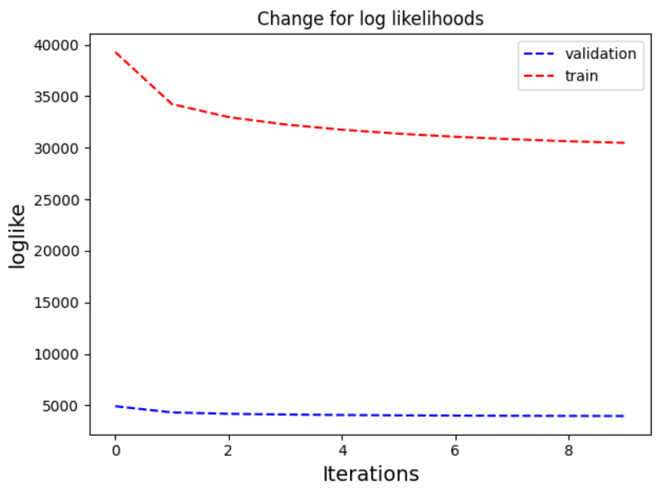
(a) Here, we derive an expression for the log-likelihood corresponding to this model and its derivative with respect to the user-ability and question-difficulty parameters.

$$\begin{aligned}
\mathcal{L}(\mathbf{C}|\boldsymbol{\theta},\boldsymbol{\beta}) &= \sum_{i=1}^{N}\sum_{j=1}^{J} P(c_{ij}=1|\theta_i,\beta_j)^{c_{ij}}(1-P(c_{ij}=0|\theta_i,\beta_j))^{1-c_{ij}} \\
\ell(\mathbf{C}|\boldsymbol{\theta},\boldsymbol{\beta}) &= \sum_{i=1}^{N}\sum_{j=1}^{J} c_{ij}\log(P(c_{ij}=1|\theta_i,\beta_j)) + (1-c_{ij})\log(1-P(c_{ij}=0|\theta_i,\beta_j)) \\
&= \sum_{i=1}^{N}\sum_{j=1}^{J}\left(c_{ij}\left[(\theta_i-\beta_j)-\log(1+\exp(\theta_i-\beta_j))\right] + (1-c_{ij})\log\left(\frac{1}{1+\exp(\theta_i-\beta_j)}\right)\right) \\
&= \sum_{i=1}^{N}\sum_{j=1}^{J}\left(c_{ij}(\theta_i-\beta_j)-\log(1+\exp(\theta_i-\beta_j))\right) \\
&= \sum_{i:\,(i,j)\in C}\;\sum_{j:\,(i,j)\in C}\left(c_{ij}(\theta_i-\beta_j)-\log(1+\exp(\theta_i-\beta_j))\right) \\
\frac{\partial\ell}{\partial\theta_i} &= \sum_{j:\,(i,j)\in C} c_{ij} - \sum_{j:\,(i,j)\in C}\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta j)} \\
\frac{\partial\ell}{\partial\beta_j} &= \sum_{i:\,(i,j)\in C} -c_{ij} + \sum_{i:\,(i,j)\in C}\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta j)}
\end{aligned}$$

(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:

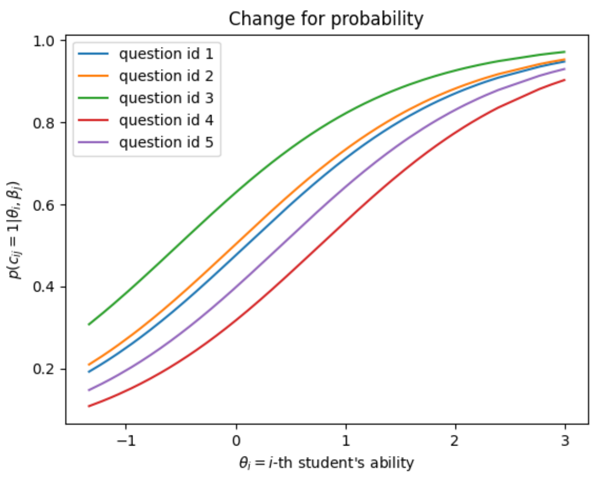
- Final validation accuracy: 0.7064634490544736
- Final test accuracy: 0.7039232289020604

(d) The plot is shown below. It shows the probability of correct response $P(c_{ij}|\theta_i, \beta_j)$ as a function of θ_i, which represents the i-th student's ability. For each fixed question, the relationship between $P(c_{ij}|\theta_i, \beta_j)$ and θ follows the sigmoid function

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

There is a positive relationship between $P(c_{ij}|\theta_i, \beta_j)$ and θ. It makes sense that a student having greater ability would be more likely to answer a question correctly if we hold the question difficulties constant.

Also, notice that different questions have different correctness probabilities if we hold the students' abilities constant. For instance, the curve for the question with ID 3 is above the curve for the question with ID 4, indicating that the former is more likely to be correctly answered by students with equal ability compared to the latter question, which is comparatively harder.



3. In this approach, we use neural networks.

   (ii) Neural Networks

   (a) Here are some differences between the neural network autoencoder and ALS:

   A. The parameterizations of the problems are different. In the neural network, we're training the weights of the activation functions g, h to minimize the squared error loss which is computed by passing the input through the two-layer autoencoder. In ALS, we're training the vectors $\mathbf{u}, \mathbf{z}$, reconstructing the matrix, and then computing the squared error loss by the reconstruction error.

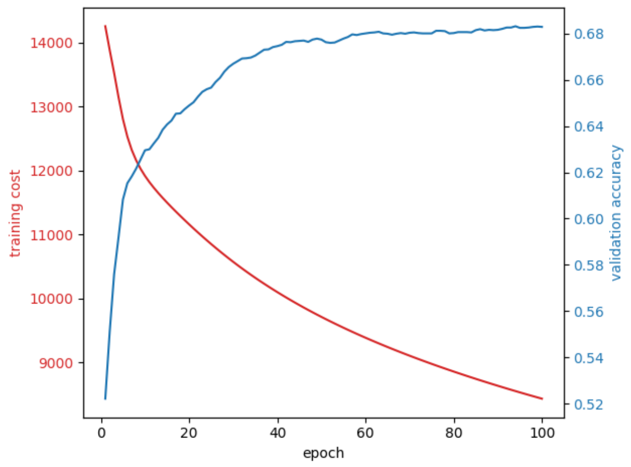
   B. We can use backpropagation in neural networks, but we can't do the same for the ALS optimization problem.

   C. We use different activation functions. In ALS, we classify predictions as correct if they're above 0.5 and incorrect otherwise, but we allow the predictions to go above 1 (consider the case where a question was very hard for many people, then the term would become very large). In the neural network, we use a sigmoid activation function for each of the layers, ensuring that the output numbers are valid probabilities between 0 and 1.

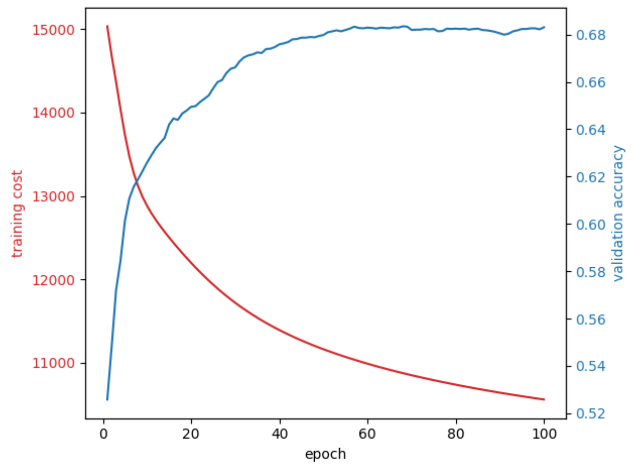
   (b) See `neural_network.py`

   (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]:

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

   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 hyperparameter

   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 (user, 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. In this section, we illustrate several ideas about our model.

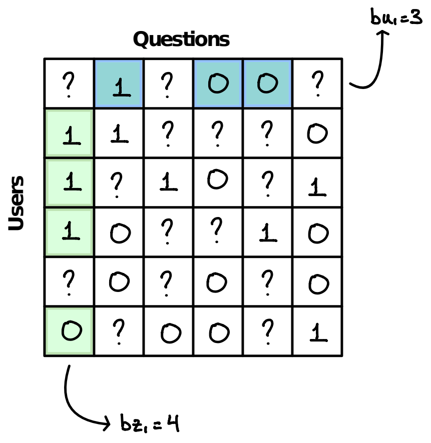


Figure 1: Sparse train matrix showing how the user and question biases are initialized

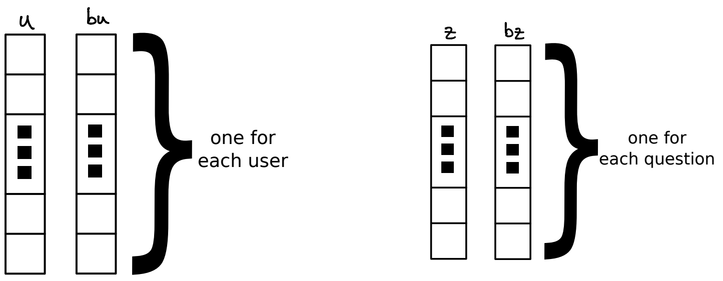


Figure 2: Parameters $\mathbf{u}, \mathbf{z}$ we aim to optimize for, and biases $\mathbf{b}_u, \mathbf{b}_z$

Firstly, the model's goal is to take the training matrix in Figure 1 and complete it so that the unobserved entries, marked '?', are predicted. The $N \times J$ matrix can split into two matrices $\mathbf{u}$ (size $N \times k$) and $\mathbf{z}$ (size $J \times k$), where k is the latent dimension hyperparameters, as shown in Figure 2 for $k = 1$. The reconstruction works by multiplying $\mathbf{u}, \mathbf{z}$:

$$\underbrace{\mathbf{u}}_{N\times k}\underbrace{\mathbf{z}^{\top}}_{k\times J} = \underbrace{\text{reconstructed matrix}}_{N\times J}$$

For bootstrapping the training data, we sampled the indices of the training data with replacement so that duplicates would be included, meaning some points could be represented more than once. This is shown in Figure 3.

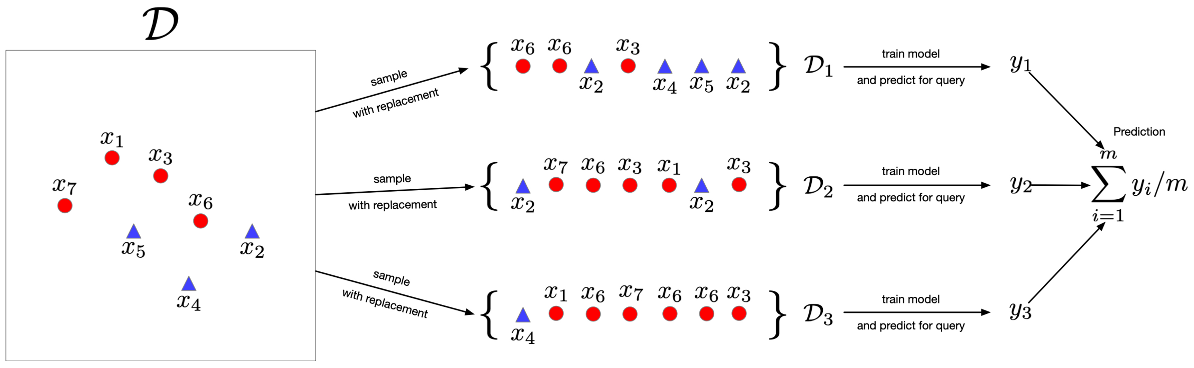

$\mathcal{D}$
x_1 x_3 x_7 x_6 x_5 x_2 x_4
sample with replacement
x_6 x_6 x_2 x_3 x_4 x_5 x_2 $\mathcal{D}_1$
x_2 x_7 x_6 x_3 x_1 x_2 x_3 $\mathcal{D}_2$
x_4 x_1 x_6 x_7 x_6 x_6 x_3 $\mathcal{D}_3$
train model and predict for query
y_1
y_2
y_3
Prediction
$\sum_{i=1}^{m} y_i/m$

Figure 3: Illustration of bagging approach for the various matrix factorization models, taking the average of the predictions for the base models as the overall prediction [GBM20a].

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.

   Below, the left plots in Figures 4, 6, 8, 10, 12, 14 show the validation loss as the original models without bias and regularization are trained for 1 million iterations. The right plots in Figures 5, 7, 9, 11, 13, 15 show the validation loss as the new models with bias and regularization are trained for 4 million iterations. Note that the validation losses are on different scales due to the addition of various terms in the new model, but the general trend is the same. The models with bias and regularization have a very erratic trend.

The corresponding validation and test accuracies of the original models are as follows:

k	validation accuracy	test accuracy
40	0.7085802991814846	0.7087214225232854
50	0.7056167090036692	0.7061812023708721
60	0.707310189105278	0.707310189105278
70	0.7050522156364663	0.7084391758396839
80	0.707310189105278	0.7061812023708721
90	0.7061812023708721	0.703076488851256

The validation and test accuracies of our final, bootstrapped with bias and weight regularization were 0.7068868190798758 and 0.7058989556872707, respectively.

Thus, the performance did not improve by much but the model is more generalizable because we bagged the models. This more or less falls in line with our expectations; we don't expect bagging to increase the accuracy.

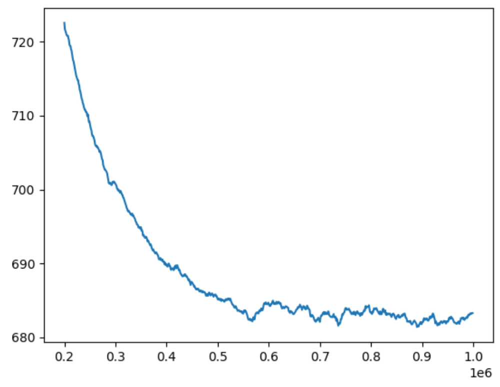


Figure 4: No bias & regularization, $k = 40$

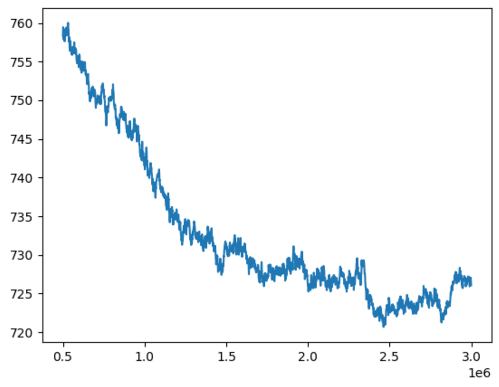


Figure 5: With bias & regularization, $k = 40$

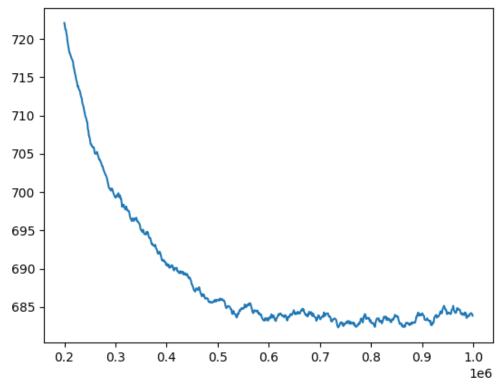


Figure 6: No bias & regularization, $k = 50$

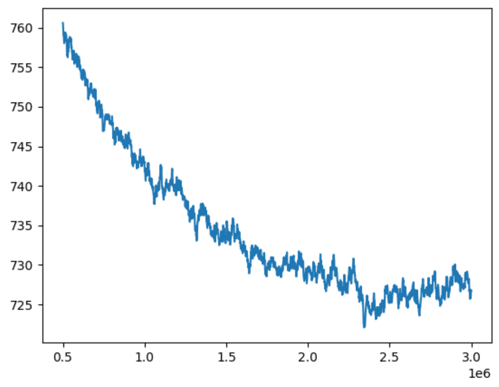


Figure 7: With bias & regularization, $k = 50$

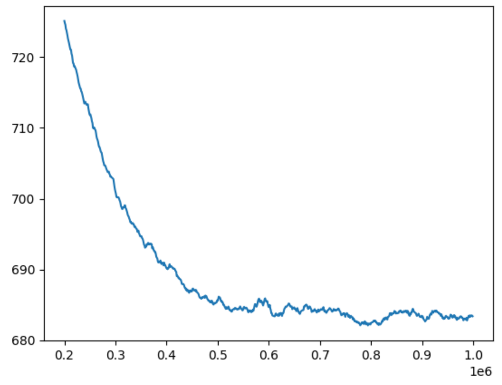


Figure 8: No bias & regularization, $k = 60$

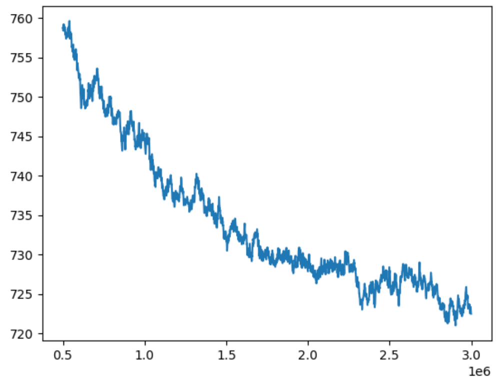


Figure 9: With bias & regularization, $k = 60$

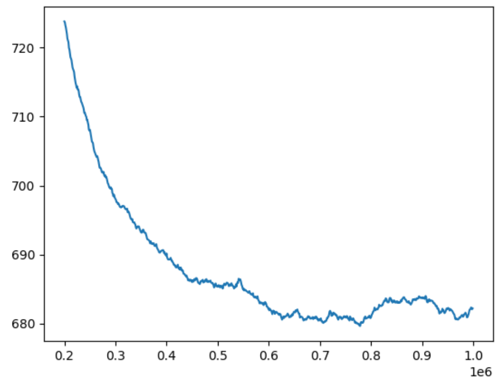


Figure 10: No bias & regularization, $k = 70$

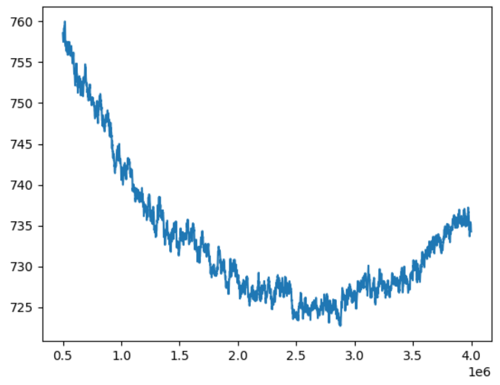


Figure 11: With bias & regularization, $k = 70$

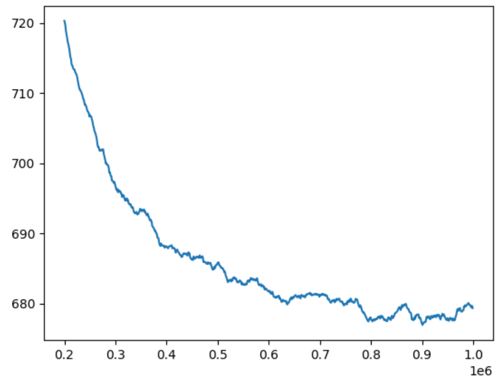


Figure 12: No bias & regularization, $k = 80$

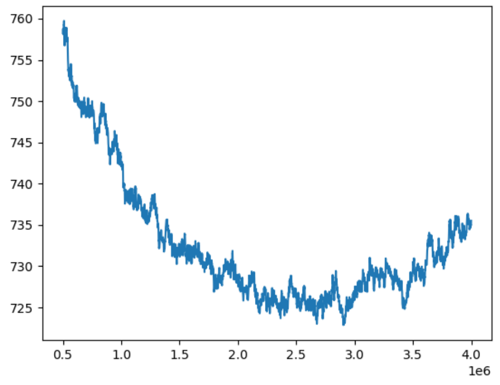


Figure 13: With bias & regularization, $k = 80$

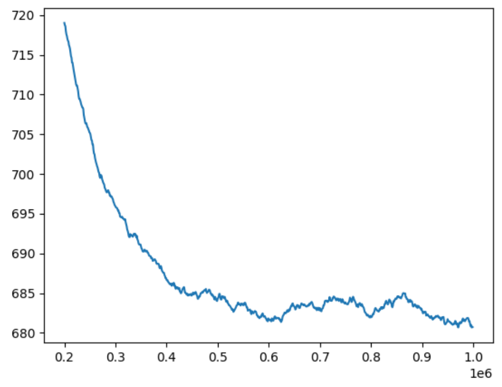


Figure 14: No bias & regularization, $k = 90$

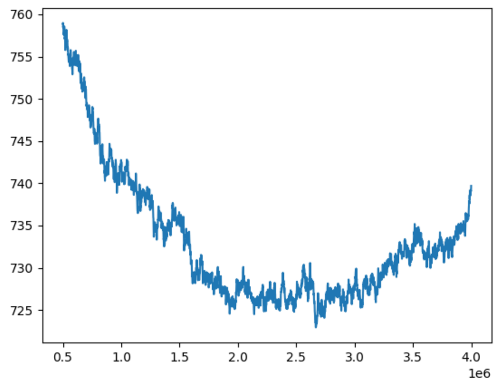


Figure 15: With bias & regularization, $k = 90$

4. Our method may not perform optimally if the weights are roughly equal, since there wouldn't be a gain in this instance. It wouldn't perform badly – it would just take awhile to train. The addition of weight regularization means that it makes many more iterations to train the models, and if the overall performance remains the same, then there is no point in incorporating the biases and weight regularization. Another shortcoming is the lack of appropriate metadata that could help with weighing the features.

   A major limitation of matrix factorization is over-specialization. The new prediction for a particular student is based on the student's past performance; when a new question appears, a student who's historically achieved high grades is predicted to be more likely to answer it correctly. This may not always be the case, because students with poor grades in the past may improve their study habits to perform well in the future. In addition, the data we used doesn't sufficient data to represent the student's ability in terms of other academic competencies.

   In other words, we don't have enough data, and that's likely the reason why most models we have attain only around 70% accuracy on the training, validation, and test sets.

   One way to improve the first limitation is that we could try to add some randomness to the model, changing the original squared error loss to be

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

   where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ is a hyperparameter we select using the validation set in order to promote/simulate 'improvements' in students' performance. With this definition, most students would have small improvements (ε_i would be near the mean of 0), but some may increase/decrease in ability drastically (ε_i would be at the tails of the normal distribution).

   The second limitation could be addressed by using the question metadata, which contains information about the question subject. It might be useful to introduce an additional constant s_{ij} which represents the student's ability for a certain subject. For example, let's assume student i answers question j, where question j is a geometry question. Then we could define this term to be a function of the number of geometry questions the student has answered correctly:

$$s_{ij} = f(\#\text{ of geometry questions the student has answered correctly})$$

   It would be interesting to see if there are other datasets available for this problem and to see if there are any class-wide breakthroughs in models for this problem, just like there was for the Netflix recommendation system problem.

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.

[GBM20a] Roger Grosse, Juhan Bae, and Christopher Maddison. CSC311 Lecture 6. Oct. 2020.

[GBM20b] Roger Grosse, Juhan Bae, and Christopher Maddison. CSC311 Lecture 8. Nov. 2020.