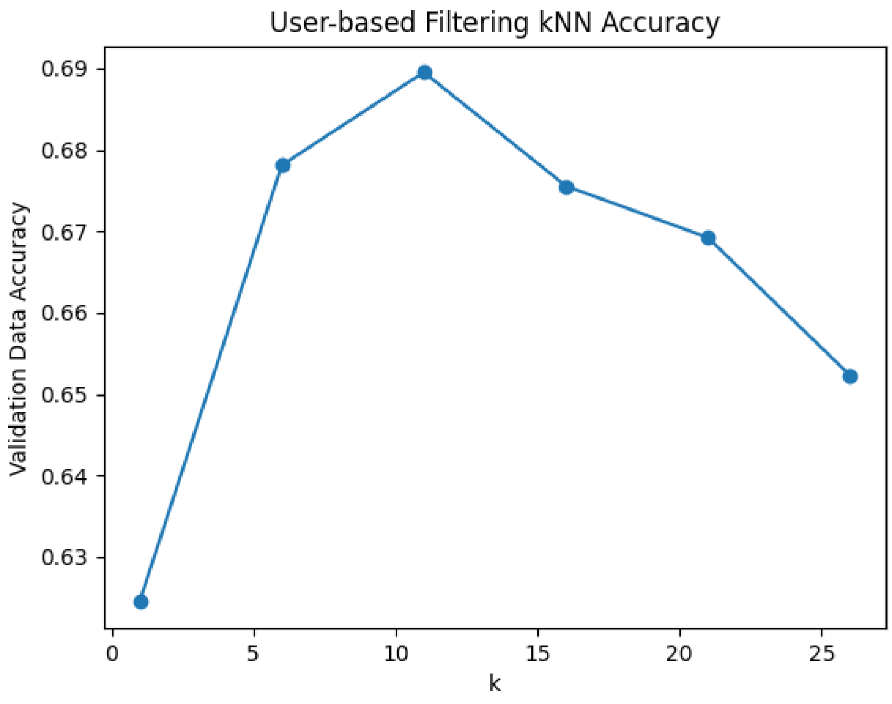
CSC311 Final Project

Alex Shih, Stella Cai, Cullen Pu

December 15, 2020

Part A

1. (a) The accuracy using user-based collaborative filtering on the validation data as a function of k is as follows:

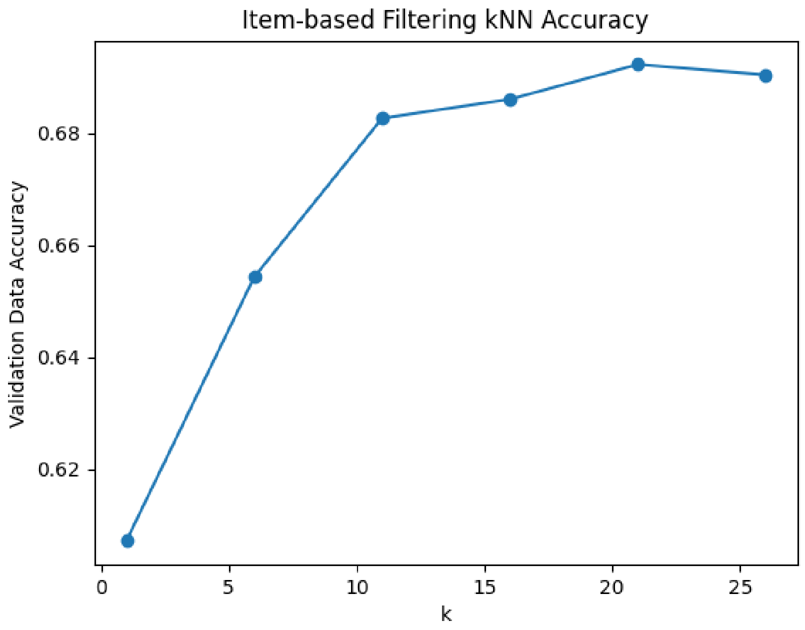


k	Accuracy
1	0.62447
6	0.67810
11	0.68953
16	0.67556
21	0.66921
26	0.65227

(b) The k^* that has the highest performance on validation data is $k = 11$. The test accuracy of $k = 11$ is 0.68417.

(c) The core underlying assumption for item-based collaborative filtering is that if in the training data question A is answered correctly if and only if question B is answered correctly, we can predict a student's correctness on question A matches their correctness on question B.

The accuracy using item-based collaborative filtering on the validation data as a function of k is as follows:



k	Accuracy
1	0.60711
6	0.65425
11	0.68261
16	0.68600
21	0.69221
26	0.69038

The k^* that has the highest performance on validation data is $k = 21$. The test accuracy of $k = 21$ is 0.66836.

(d) The test accuracy using item-based collaborative filtering was 0.66836. This was slightly worse than the test accuracy using user-based collaborative filtering, which was 0.68417.

(e) One limitation for these methods occurs if we want to make predictions about a user or question with limited existing information. For example, with user-based collaborative filtering, if a new user joins the online learning platform, it is difficult to accurately match the user with similar users and make predictions on the new user's performance on diagnostic questions. Another limitation for kNN algorithms is poor scalability as the number of users and questions increases. For example, in *a single query* of user-based collaborative filtering, if we are trying to find the most similar user out of N users, the time to calculate D-dimensional euclidean distances is $\mathcal{O}(ND)$. We must then sort those distances to find the closest match, which is $\mathcal{O}(nlogn)$.

2. (a) The derivative for the log-likelihood log $p(\mathbf{C}|\boldsymbol{\theta}, \boldsymbol{\beta})$ is as follows

$$\begin{aligned}
p(\mathbf{C}|\boldsymbol{\theta}, \boldsymbol{\beta}) &= \prod_{i=1}^{N}\prod_{j=1}^{D} p(c_{ij}=1|\theta_i,\beta_j)^{c_{ij}} p(c_{ij}=0|\theta_i,\beta_j)^{1-c_{ij}} \\
&= \prod_{i=1}^{N}\prod_{j=1}^{D} \left(\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}\right)^{c_{ij}} \left(1-\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}\right)^{1-c_{ij}} \\
&= \prod_{i=1}^{N}\prod_{j=1}^{D} \left(\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}\right)^{c_{ij}} \left(\frac{1}{1+\exp(\theta_i-\beta_j)}\right)^{1-c_{ij}}
\end{aligned}$$

$$\begin{aligned}
\log\, p(\mathbf{C}|\boldsymbol{\theta}, \boldsymbol{\beta}) &= \log\, \prod_{i=1}^{N}\prod_{j=1}^{D} \left(\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}\right)^{c_{ij}} \left(\frac{1}{1+\exp(\theta_i-\beta_j)}\right)^{1-c_{ij}} \\
&= \sum_{i=1}^{N}\sum_{j=1}^{D} \left(c_{ij}\, \log\, \frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)} + (1-c_{ij})\, \log\, \frac{1}{1+\exp(\theta_i-\beta_j)}\right) \\
&= \sum_{i=1}^{N}\sum_{j=1}^{D} [c_{ij}\, \log\, \exp(\theta_i-\beta_j) - c_{ij}\, \log\, (1+\exp(\theta_i-\beta_j)) - (1-c_{ij})\, \log\, (1+\exp(\theta_i-\beta_j))] \\
&= \sum_{i=1}^{N}\sum_{j=1}^{D} [c_{ij}(\theta_i-\beta_j) - c_{ij}\, \log\, (1+\exp(\theta_i-\beta_j)) - (1-c_{ij})\, \log\, (1+\exp(\theta_i-\beta_j))] \\
&= \sum_{i=1}^{N}\sum_{j=1}^{D} [c_{ij}(\theta_i-\beta_j) - c_{ij}\, \log\, (1+\exp(\theta_i-\beta_j)) - \log\, (1+\exp(\theta_i-\beta_j)) + c_{ij}\, \log\, (1+\exp(\theta_i-\beta_j))] \\
&= \sum_{i=1}^{N}\sum_{j=1}^{D} [c_{ij}(\theta_i-\beta_j) - \log\, (1+\exp(\theta_i-\beta_j))]
\end{aligned}$$

$$\frac{\partial}{\partial\theta_i}\ell(\mathbf{C}|\boldsymbol{\theta}, \boldsymbol{\beta}) = \sum_{j=1}^{D}\left(c_{ij} - \frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}\right)$$

$$\begin{aligned}
\frac{\partial}{\partial\beta_j}\ell(\mathbf{C}|\boldsymbol{\theta}, \boldsymbol{\beta}) &= \sum_{i=1}^{N}\left(-c_{ij} + \frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}\right) \\
&= \sum_{i=1}^{N}\left(\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)} - c_{ij}\right)
\end{aligned}$$

(b) We tuned the hyperparameters of number of gradient descent iterations and the learning rate.
Iterations and accuracy on validation data

Num Iterations	Accuracy
5	0.70124
10	0.70604
20	0.70646
30	0.70689
50	0.70590
100	0.70618

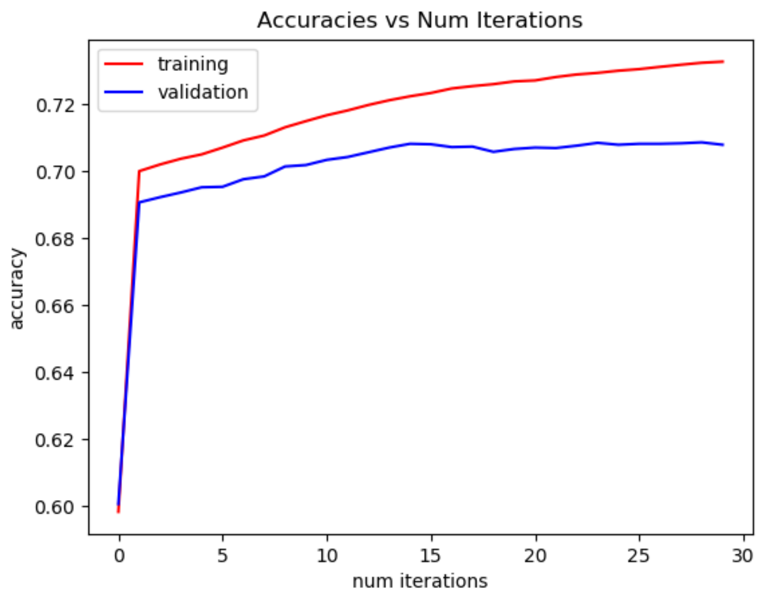
We tested this with a learning rate of 0.01. It seemed that the optimal number of iterations is 30.

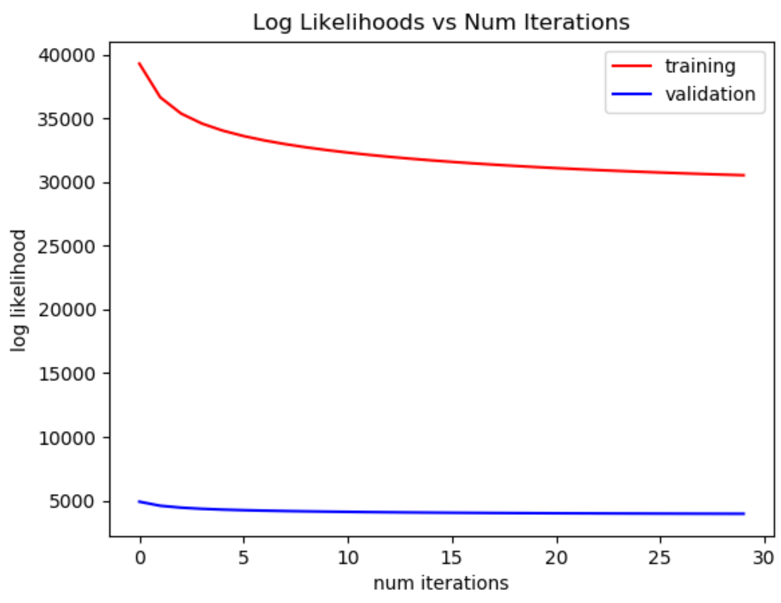
Learning Rate and accuracy on validation data

Learning Rate	Accuracy
0.001	0.69800
0.005	0.70802
0.01	0.70689
0.05	0.68473
0.1	0.63844

We can see that the optimal learning rate after 30 iterations of gradient descent is 0.005 which resulted in an accuracy of 0.70802. While watching the results of each iterations, it was interesting to see the smaller learning rates such as 0.001 and 0.005 improve in accuracy very slowly while larger learning rates such as 0.05 resulted in fluctuations after multiple iterations due to having too high of a learning rate.

With a learning rate of 0.005 and 30 iterations, this is the result.

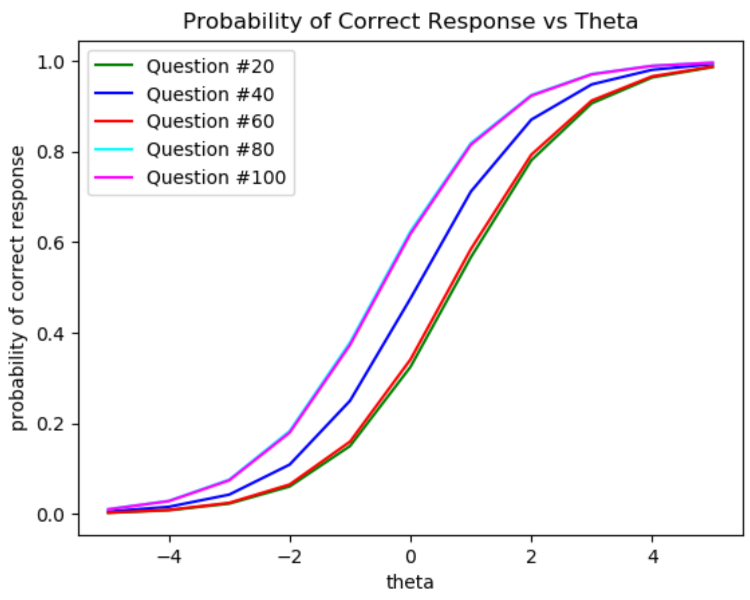




(c) The final validation and test accuracy with a learning rate of 0.005 and 30 iterations are as follows:

Data	Accuracy
Validation	0.70802
Test	0.70279

(d)



We can see that these curves are all shaped like a sigmoid, which makes sense as we are trying to predict questions with probability between 0 and 1. Additionally, the one-parameter IRT model is closely related to logistic regression. Since we have fixed our β values for each of these questions, these curves represent given a student's ability represented by θ, what is probability of answering the question correct. This makes sense, as we would expect a student with a higher ability to have a higher probability of answering the question.

3. (a) k-values and accuracy on validation data

k	Accuracy
1	0.64282
5	0.65905
10	0.65862
20	0.65397
50	0.64846
100	0.64705

It seems the best k value for the validation data is $k = 5$. When this is run on the test set, we get a performance accuracy of 0.66356.

(b) SVD does not perform well on sparse matrices. As a result, we are filling in the missing NaN values using the mean for that column. Thus we are assuming that the prediction for a student's performance on a question is the average of all other students' performance on that question. However, the questions belong to different categories and the average performance on a question may not be indicitive of how that student performs on that question. For example, if a student is very poor at math and the entry whose value is NaN is a math question, their actual accuracy might be much lower than what the average is. Thus the "latent features" we are looking for may be influenced by how we are filling in NaN values.

(d) *A seed was set so that our results could be reproduced*
We tune the learning rate with $k = 5$ and iterations fixed at 500,000.

Learning Rate	Accuracy
0.001	0.51665
0.005	0.67626
0.01	0.70251
0.03	0.69023
0.05	0.67697
0.1	0.67147

From testing different learning rates through 500,000 iterations, we concluded that a learning rate of 0.01 performed the best.

We tune the number of iterations with $k = 5$ and a learning rate fixed to 0.01.

Num Iterations	Accuracy
10000	0.41152
50000	0.51552
100000	0.59822
200000	0.66413
500000	0.70421
1000000	0.69856

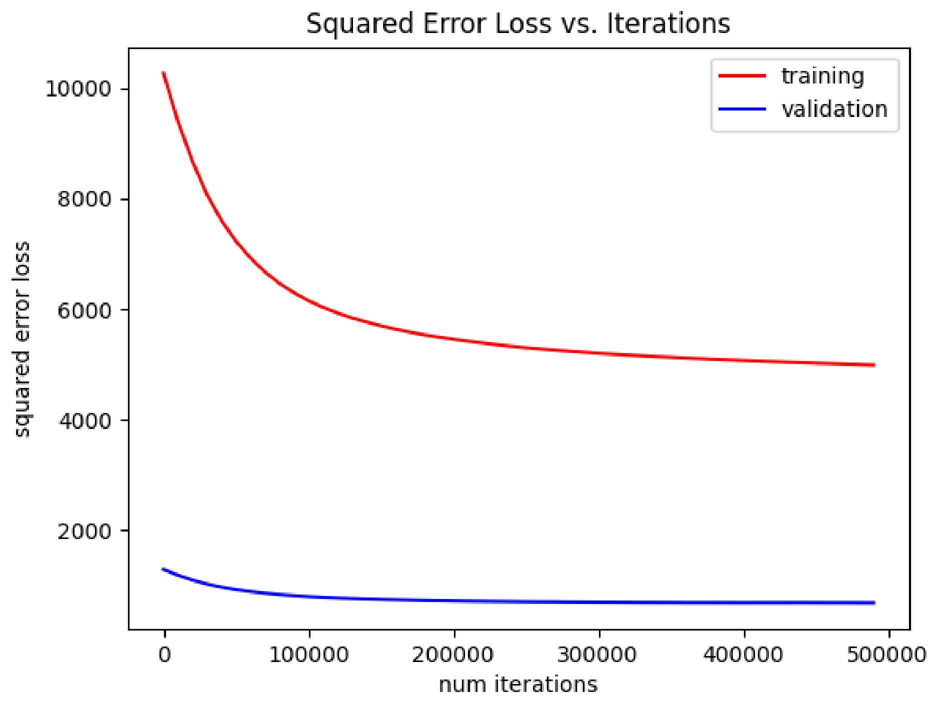
From our testing, we saw that the accuracy seemed to converge around 500,000 to 1,000,000 iterations. Due to convenience for testing, we decided that 500,000 would be optimal as it's accuracy was very similar to that of 1,000,000 and it ran much faster.

We tune k with learning rate fixed to 0.01 and iterations fixed to 500,000.

k	Accuracy
1	0.69659
5	0.70421
10	0.69983
20	0.69969
50	0.70717
100	0.70661

From testing different k-values with a learning rate of 0.01 and through 500,000 iterations, we concluded that a k-value of 50 performed the best.
Our chosen hyperparameters are 500,000 iterations, a learning rate of 0.01, and $k = 50$.

(e)



We can see that the squared error loss of both the training and validation sets decrease as number of iterations increases. We can also see that the training squared error loss is constantly much greater than the validation squared error loss. These are both expected results. We expect loss to decrease as our model performs better as iterations increase. Also because the training set has many more data points, the training squared error loss should be more than the validation squared error loss.
The following are results from using hyperparameters chosen above.

Data	Accuracy
Train	0.74280
Validation	0.70265
Test	0.70082

(f) If our model was a binary classification problem, we would only want our range to be in $\{0, 1\}$. Thus an appropriate loss function to use would be the logistic cross entropy loss function which would interpret our outputs as estimated probability, which can be used for binary classification.

4. *A seed was set so that our results could be reproduced*

Our bagging ensemble consisted of the ALS base model from question 3 and three random resamples. After sampling with replacement 3 times, we trained our base model on each resample with the chosen hyperparameters from part (e). These hyperparameters were 500,000 iterations, a learning rate of 0.01, and $k = 50$. Finally, we averaged the the prediction matrices before thresholding to get a binary prediction. Our final validation and test accuracy are as follows:

Data	Accuracy
Train	0.74372
Validation	0.70040
Test	0.70449

As noted in part 3, the validation accuracy of a single instance of the base ALS model was 0.70265. We see that our bagged ensemble achieves a validation accuracy of 0.70040 - slightly lower than the standalone model. We note the test accuracy for the ensemble was slightly higher, coming in at 0.70449 while the standalone ALS model had a test accuracy of 0.70082.

As a result, we cannot conclude that bagging improved our predictions. This was surprising to us because our individual ALS model was overfitting to the training data, so we believed that bagging would reduce overfitting by averaging predictions. One reason that the results did not turn out as expected is because we are only resampling 3 times. Had we taken more bootstrap samples, we may start seeing the performance of the ensemble improve.

Part B: 2-PL IRT, Funk SVD, and Ensembling

Formal Description

As noted in question 4, our ensemble showed no improvement over the base ALS model. To improve our ensemble, we decided to both increase the number of resamples and train using 2 improved base models. The models we wanted to improve on are the IRT model and the ALS matrix factorization model. We first go over the improvements made to both models and then discuss the performance of the ensemble.

Improving Item Response Theory

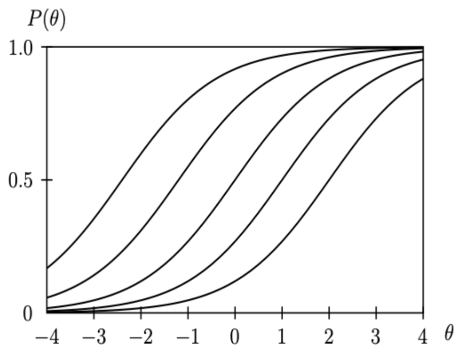
In question 2, we implemented the one-parameter IRT model that assigned a student i the ability value θ_i and each question j a difficulty value β_j to formulate a probability distribution. The probability that question j is correctly answered by student i is the following:

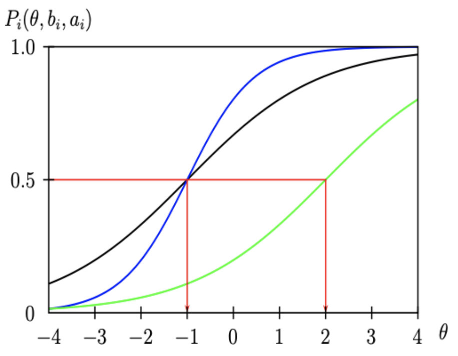
$$p(c_{ij} = 1|\theta_i, \beta_j) = \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)}$$

In our extension of the IRT model, we hoped to implement the two-parameter IRT model, which, in addition to the ability parameter θ_i and the difficulty parameter β_j, also features a *discrimination parameter* a_j. In the two-parameter IRT model, called the 2-PL, the probability that question j is correctly answered by student i is the following:

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

The idea behind the a parameter is that in our original 1-PL model, the slope of the response curves of every question was nearly identical. In the context of the data, where diagnostic questions are answered by students from 7 up to 18 years old, we felt as if this restricted our model. Questions that are geared towards younger students may be difficult for them, but quickly become trivial for students even slightly older. Likewise, questions geared towards older students may be borderline impossible for students even slightly younger. Questions meant for the extremes of the age range - 7 year olds and 18 year olds - are likely better modeled with a high distinction; that is, the slope would be very steep. The addition of the a parameter serves to modify the steepness (the slope) of the item response curves so that they are not nearly parallel, as shown in the figures below:





We hypothesized that the two-parameter IRT model would improve our predictions because the a_j parameters allows us to find additional patterns within each question. For instance, in this example, the question with the blue probability curve is more difficult for students with higher abilities, while the question with the black probability curve is more difficult for students of lower abilities.

The next step of our process was to derive the derivatives of the log-likelihood with respect to each parameter θ_i, β_j, a_j, which are used to perform gradient descent. A few steps are omitted because it is very similar to the derivation found in question 2. The partial derivatives for the log-likelihood log $p(\mathbf{C}|\boldsymbol{\theta}, \mathbf{a}, \boldsymbol{\beta})$ is as follows:

$$\begin{aligned} p(\mathbf{C}|\boldsymbol{\theta}, \mathbf{a}, \boldsymbol{\beta}) &= \prod_{i=1}^{N}\prod_{j=1}^{D} p(c_{ij}=1|\theta_i, a_j, \beta_j)^{c_{ij}} p(c_{ij}=0|\theta_i, a_j, \beta_j)^{1-c_{ij}} \\ &= \prod_{i=1}^{N}\prod_{j=1}^{D} \left(\frac{\exp(a_j(\theta_i-\beta_j))}{1+\exp(a_j(\theta_i-\beta_j))}\right)^{c_{ij}} \left(\frac{1}{1+\exp(a_j(\theta_i-\beta_j))}\right)^{1-c_{ij}} \end{aligned}$$

$$\begin{aligned} \log\, p(\mathbf{C}|\boldsymbol{\theta}, \mathbf{a}, \boldsymbol{\beta}) &= \log\, \prod_{i=1}^{N}\prod_{j=1}^{D} \left(\frac{\exp(a_j(\theta_i-\beta_j))}{1+\exp(a_j(\theta_i-\beta_j))}\right)^{c_{ij}} \left(\frac{1}{1+\exp(a_j(\theta_i-\beta_j))}\right)^{1-c_{ij}} \\ &= \sum_{i=1}^{N}\sum_{j=1}^{D} \left(c_{ij}\, \log\, \frac{\exp(a_j(\theta_i-\beta_j))}{1+\exp(a_j(\theta_i-\beta_j))} + (1-c_{ij})\, \log\, \frac{1}{1+\exp(a_j(\theta_i-\beta_j))}\right) \\ &= \sum_{i=1}^{N}\sum_{j=1}^{D} \left[c_{ij}\cdot a_j(\theta_i-\beta_j) - \log\, (1+\exp(a_j(\theta_i-\beta_j)))\right] \end{aligned}$$

$$\frac{\partial}{\partial\theta_i}\ell(\mathbf{C}|\boldsymbol{\theta}, \mathbf{a}, \boldsymbol{\beta}) = \sum_{j=1}^{D}\left(c_{ij}\cdot a_j - \frac{a_j \exp(a_j(\theta_i-\beta_j))}{1+\exp(a_j(\theta_i-\beta_j))}\right)$$

$$\frac{\partial}{\partial a_j}\ell(\mathbf{C}|\boldsymbol{\theta}, \mathbf{a}, \boldsymbol{\beta}) = \sum_{i=1}^{N}\left(c_{ij}(\theta_i-\beta_j) - \frac{(\theta_i-\beta_j)\exp(a_j(\theta_i-\beta_j))}{1+\exp(a_j(\theta_i-\beta_j))}\right)$$

$$\frac{\partial}{\partial\beta_j}\ell(\mathbf{C}|\boldsymbol{\theta}, \mathbf{a}, \boldsymbol{\beta}) = \sum_{i=1}^{N}\left(\frac{a_j \exp(a_j(\theta_i-\beta_j))}{1+\exp(a_j(\theta_i-\beta_j))} - c_{ij}\cdot a_j\right)$$

As before, gradient descent was used to minimize the loss. After tuning for hyperparameters, we found that to maximize the validation accuracy, the ideal number of iterations was 30 and the ideal learning rate was 0.009. Plots showing the log-likelihood and validation accuracy versus the number of gradient descent iterations can be found in the appendix.

Improving Matrix Factorization

One problem we ran into with our matrix factorization model was that we found our model overfitted to training data. That is, as mentioned in part (e) of question 3, after hyperparameter tuning we obtained a training accuracy of 0.74280 and validation and test accuracies of 0.70265 and 0.70082, respectively. We thought that the 4% difference in accuracy was noteworthy enough to consider ways to prevent overfitting.

If we have n users and m questions total, we are trying to decompose our predicted correctness matrix $\mathbf{C} \in \mathbb{R}^{n\times m}$ into two matrices $\mathbf{U} \in \mathbb{R}^{n\times k}$ and $\mathbf{Z} \in \mathbb{R}^{m\times k}$ such that $\mathbf{C} = \mathbf{U}\mathbf{Z}^T$. We hypothesized that a contributor to the overfitting was the size of the latent factors found by our matrix factorization; that is, the size of k in the decomposition above. If the value of k is too large, our model tends to pick up on patterns in the training data that do not exist in reality and cannot be generalized.

For old ALS algorithm, our objective was to minimize

$$\min_{\mathbf{U},\mathbf{Z}} \frac{1}{2} \sum_{(n,m)\in\mathcal{O}} (\mathbf{C}_{nm} - \mathbf{U}_n^T \mathbf{Z}_m)^2$$
$$\text{where } \mathcal{O} = \{(n,m) : \text{entry } (n,m) \text{ is observed in the training matrix}\}$$

We looked for an alternative objective, particularly one that took into account regularization for the size of the latent factors. During this process, we found the Funk SVD algorithm, popularized by Simon Funk during the Netflix Prize competition. In this method of matrix factorization, we also perform updates on some additional variables, namely the user biases $\mathbf{bu} \in \mathbb{R}^n$ and question biases $\mathbf{bq} \in \mathbb{R}^m$. These biases are helpful in taking into account a question's difficulty or a user's tendency to answer questions correctly.

The objective of this algorithm is to minimize the following objective:

$$\min_{\mathbf{U},\mathbf{Z}} \frac{1}{2} \sum_{(n,m)\in\mathcal{O}} (\mathbf{C}_{nm} - \mathbf{U}_n^T \mathbf{Z}_m)^2 + \lambda(bu_i^2 + bq_j^2 + \|\mathbf{U}_i\|^2 + \|\mathbf{Z}_j\|^2)$$
$$\text{where } \mathcal{O} = \{(n,m) : \text{entry } (n,m) \text{ is observed in the training matrix}\}$$
$$\text{and } \lambda \text{ is the regularization parameter}$$

We can estimate the correctness of a user i on question j by thresholding

$$\hat{c_{i,j}} = \bar{c} + bu_i + bq_j + U_i^T Z_j$$

Then, in each step of gradient descent, we perform the following updates:

$$\begin{aligned}
err &= c_{i,j} - \hat{c}_{i,j} \\
bu_i &= bu_i + \alpha(err - \lambda * bu_i) \\
bq_j &= bq_j + \alpha(err - \lambda * bq_j) \\
\mathbf{U}_i &= \mathbf{U}_i + \alpha(err * \mathbf{Z}_j - \lambda \times \mathbf{U}_i) \\
\mathbf{Z}_j &= \mathbf{Z}_j + \alpha(err * \mathbf{U}_i - \lambda \times \mathbf{Z}_j)
\end{aligned}$$

where α is the learning rate and λ is the regularization term.

We ran the Funk SVD algorithm on different hyperparameters for the learning rate, k value, regularization penalty and number of iterations. The plots for tuning hyper parameters can be found in our appendix.

After tuning the hyperparameters, we found the best ones to be $\alpha = 0.01$, $\lambda = 0.10$, $k = 50$, and 500 iterations.

Ensemble Implementation & Results

Now that we have implemented algorithms for Funk-SVD and 2 Parameter Item Response Theory, we looked into bagging the two models. Funk-SVD resulted in a slight improvement in accuracy after finding optimal hyperparameters compared to our original matrix factorization algorithm implemented in part A. In our report, we saw that Funk-SVD still resulted in overfitting. For our 2 Parameter Item Response Model, we saw there was not an improvement to our original model.

We decided to try and improve both of these new models by bagging them. Ultimately, we wanted to see if there was improvement between this ensemble versus our original ensembling. As we can expect a bagged classifier to be stronger than the average underlying model, this seemed like a good way to reduce overfitting and improve from the Funk-SVD and the 2 Parameter Item Response models.

We trained each model 5 times using the hyperparameters chosen above on different resamples, resulting in 10 resamples total. We then made predictions using each of the 10 trained models and generated an average of all the results to get our predictions. Our results are as follows:

Data	Accuracy
Train	0.69616
Validation	0.70816
Test	0.70675

Surprisingly, we see that our results are almost did not improve much compared to the individual models. This is similar to our ensemble results for Part A, which is surprising to us as this did not result in a decrease in overfitting. This was likely due to a lack of data points, which is explained in depth in the limitations section of this report.

Limitations

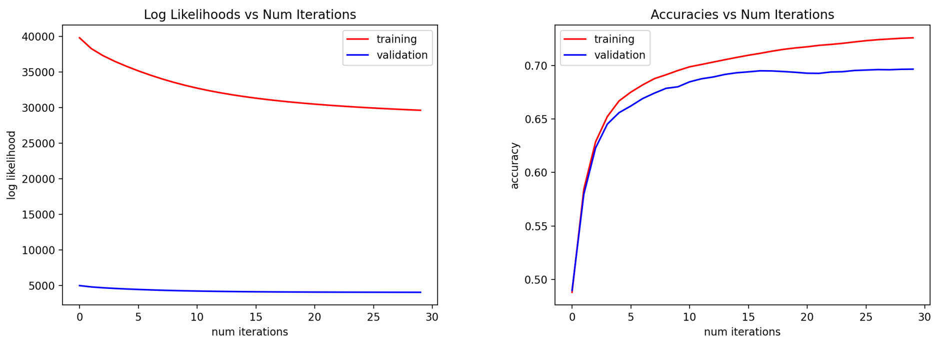
Despite the measures we took to regularize the sizes of the latent factors through Funk SVD and optimize the item response model by including an extra parameter, ultimately we did not see any significant improvements to our test accuracy. With our ten-model ensemble, we achieved a final test accuracy of 0.70674, which is only very slightly higher than the test accuracy achieved in the one-parameter item response model in Part A Question 2 (0.70279) and the test accuracy from unregularized matrix factorization in Part A Question 3 (0.70082). We can see that ensembling our models resulted in our training set having lower accuracy than our validation. This means that this ensembling performed much better in terms of decreasing overfitting when compared to the ensembling performed in Part A.

One reason we thought the Funk SVD resulted in similar accuracies as our original SVD algorithms because they are implemented similarly. Both Funk SVD and our original SVD model utilized matrix factorization at the core of the algorithm while minimizing a similar function. However, we expected that adding a regularization term would reduce overfitting. Although our final validation and test accuracy were both higher, the difference between the accuracy of the training set and the validation set was larger using Funk SVD. This was after using the optimal hyperparameter settings that we tuned, including a penalty term, which was surprising. One way that we can extend and improve the Funk SVD model is implementing the SVD++ algorithm, which includes an additional term in the summation we are minimizing that can be interpreted as including the effects of "implicit information". This tries to help the model understand that the user getting a question right or wrong can indicate the user's preference for that type/subject of question.

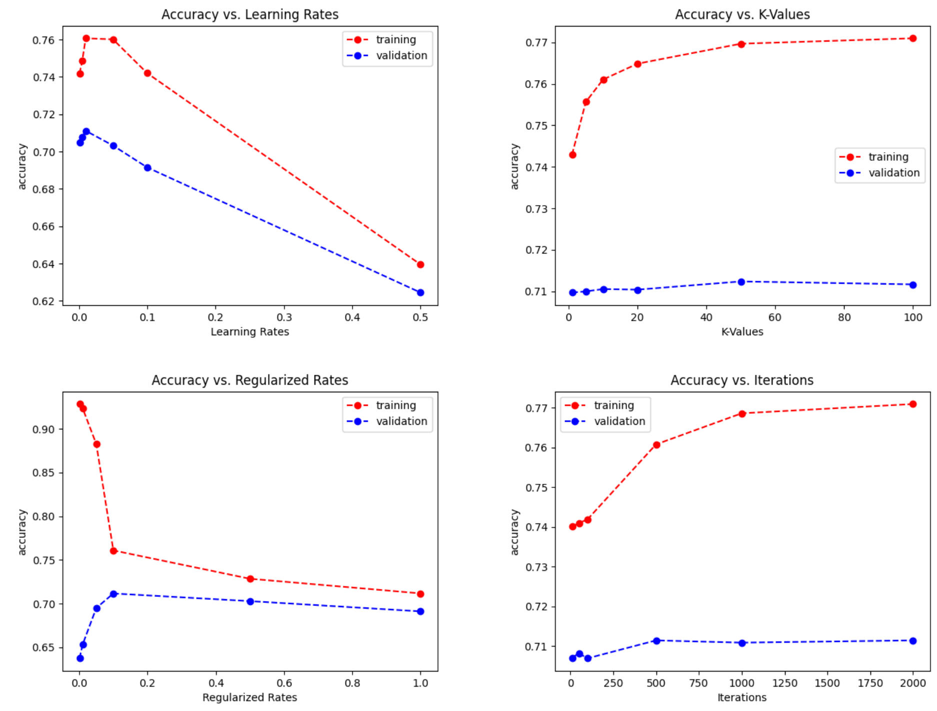
The problem of overfitting in our Funk SVD model was likely due to the algorithms being performed on a sparse matrix. We should note that in our matrix of 542 students and 1774 diagnostic questions, only 56,688 out of the possible 961,508 questions were answered. This means that around 94.10% of the matrix we used for our models are NaN values. Due to the sparsity of the matrix, this could result in overfitting solely due to the lack of data collected. This would make sense as we saw that ensembling our models drastically reduced the problem of overfitting. Thus, in general, training our models on a matrix with more data would improve our accuracies and reduce the overfitting that we currently see in our Funk SVD model.

One thing that we would have liked to include in our models was the metadata provided of both the students and the questions. If we had used neural networks or decision trees, we would have been able to include visible features included from the metadata for either students or questions such as student gender or question subject. Additionally, looking for and incorporating patterns from the metadata that cannot be detected solely from the correctness matrix could have aided us in making better and more generalizable predictions on the test dataset.

0.1 Appendix



The plots for the IRT 2-PL hyperparameter tuning



The plots for Funk SVD hyperparameter tuning

0.2 Contributions

- Stella: Implemented ALS algorithm, Implemented ensemble, Part B
- Cullen: Implemented knn, Derived IRT log likelihood, Part B
- Alex: Implemented ALS algorithm, Implemented IRT, Part B

0.3 References

1. 2-PL model and images: `https://www.metheval.uni-jena.de/irt/VisualIRT.pdf`
2. Funk SVD algorithm implementation: `https://github.com/gbolmier/funk-svd`