Machine Learning 2024 Project Midterm Report

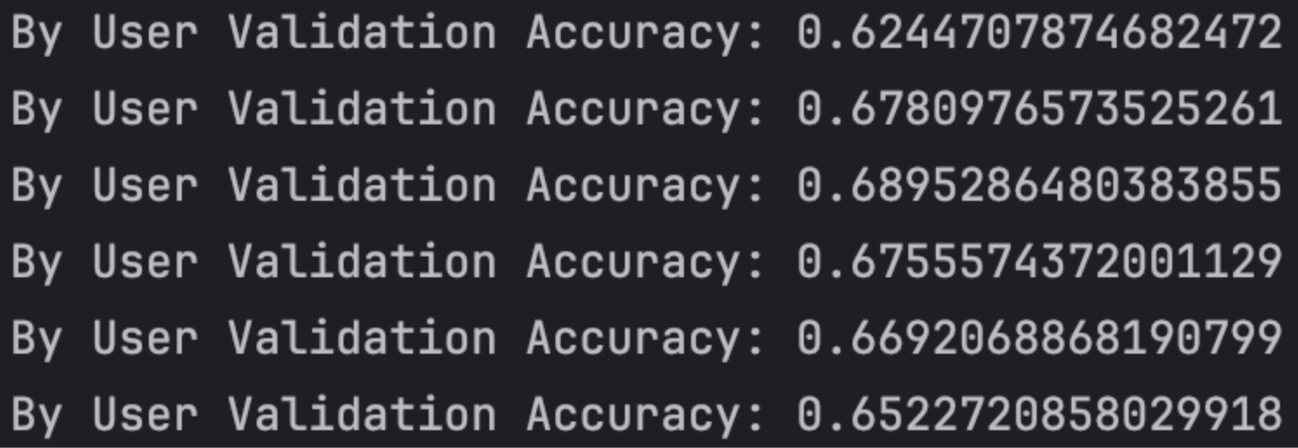
Do Duc Truong, Tran Minh Duc, Pham Hai Nam

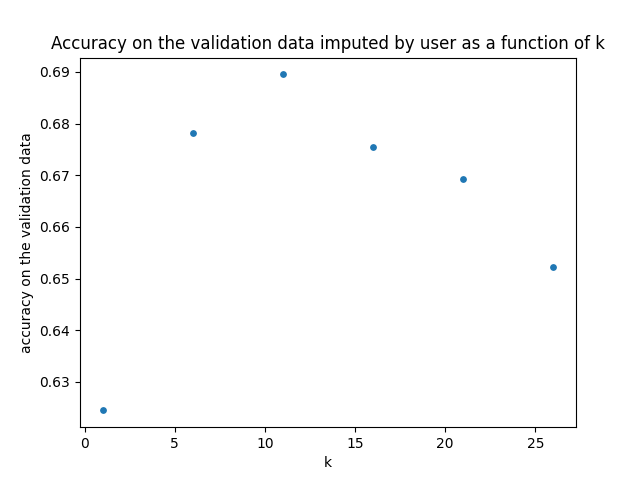
**Part A: Machine Learning Models**

**4.1  k-Nearest Neighbor (KNN)**

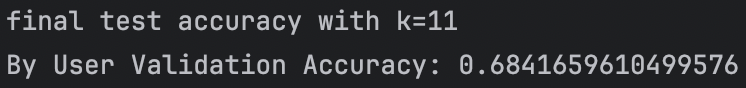
* **Step 2:** To observe how the accuracy of predictions changes with different numbers of nearest neighbors we vary the value of k to (k = 1, 6, 11, 16, 21, 26)

The accuracy on validation data for k ∈ {1, 6, 11, 16, 21, 26} is as the following:



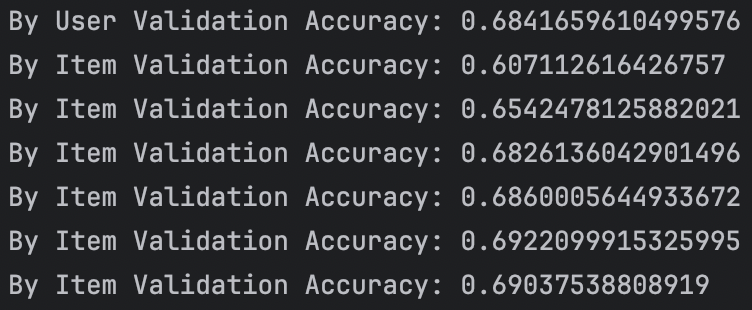


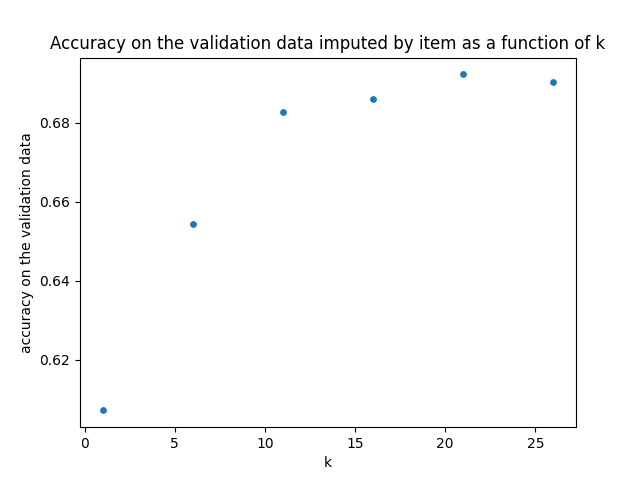
We use k\*=11 for *knn\_imputed\_by\_user* and the final test accuracy is:

****

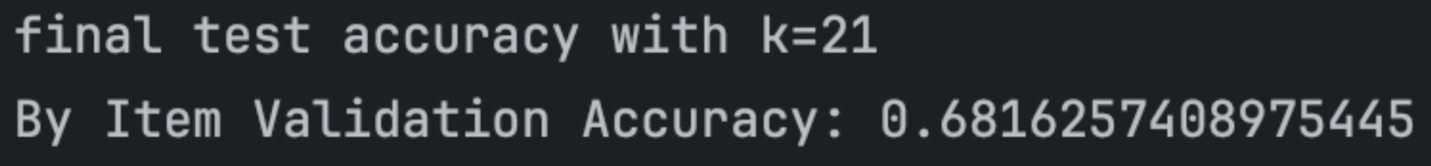
* **Step 3:** The fundamental premise of item-based collaborative filtering is that if a diagnostic question A has the same number of correct and incorrect responses from other users as question B, then the correctness of question A from a particular user is equal to that of question B.

The accuracy on validation data for k ∈ {1, 6, 11, 16, 21, 26} is as the following:

****

****

We use k\*=21 for *knn\_imputed\_by\_item* and the final test accuracy is as following:

****

* **Step 4:** The test performance between user- and item- collaborative filtering is quite similar with user- based performs slightly better. Some reasons can be consider to lead to this outcome:

+ Students might have consistent response patterns across questions

+ There could be clear groups of students with similar knowledge levels or learning styles

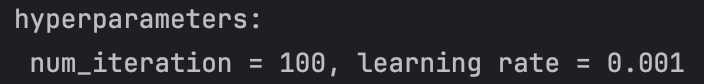
+ The number of students might be smaller than the number of questions, making user similarity more reliable

**4.2. Item Response Theory (IRT)**

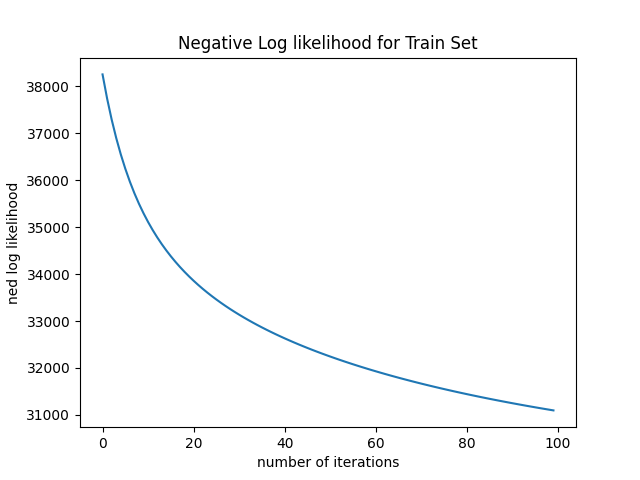
* **Step 2:** Let n = total number of students and m = total number of questions. The log-likelihood is derived as following under the Naive Bayes assumption:

Now we calculate the partial derivative of the log-likelihood with respect to each θi and βj

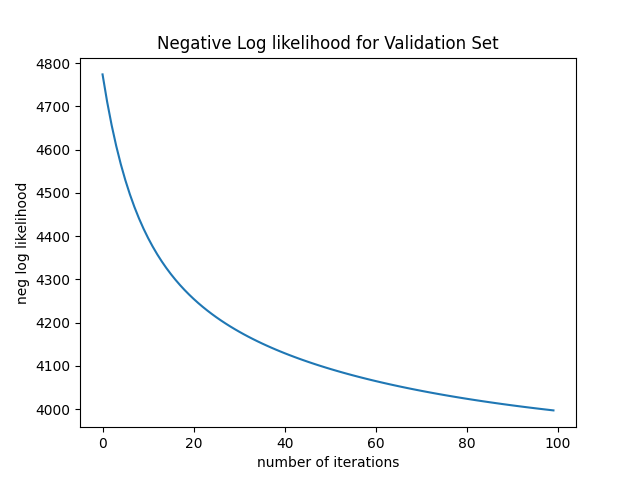
* **Step 3:** To achieve a stable training curve we adjust learning rate and the number of iterations as the following:



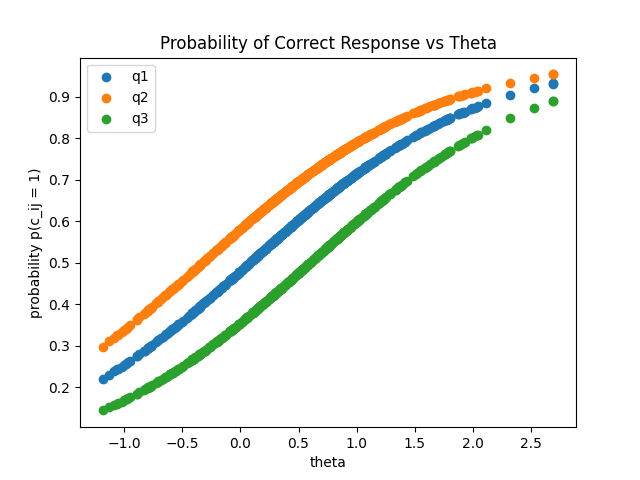
Negative log-likelihood in each interaction for the training set:



Negative log-likelihood in each interaction for the validation set:



* **Step 4:** Plot for the probability of getting correct response vs theta for 3 questions:



* **Step 5:** From this graph, we observe that when theta is small, the probability of correct responses is low, and as theta increases, the probability of correct responses also increases. This makes sense because theta represents a student's ability. Students with lower ability are less likely to answer correctly, while those with higher ability are more likely to do so. The three curves in the graph roughly follow an S-shape, as we used the sigmoid function σ(θ − β) to calculate the probability. As θ increases, the input to the sigmoid, θ − β, also increases. Since the sigmoid is a monotonically increasing function with an S-shape, the probability follows a similar pattern. The vertical displacement of the three curves could represent different levels of question difficulty. For easier questions, students of all abilities are more likely to get them right, whereas for harder questions, students of all abilities are more likely to get them wrong.

**4.3: Ensemble Learning**

In this section, we implemented a bagging ensemble method using the Item Response Theory (IRT) algorithm to enhance the accuracy and stability of our base models. The process consists of the following steps:

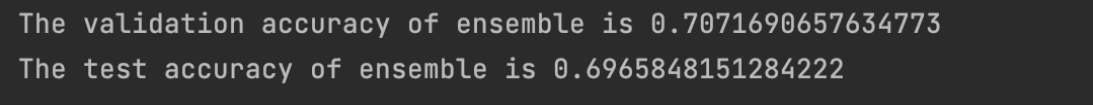
* **Step 1:** Data Resampling  
  We created three new training sets by resampling from the original dataset. Each of these sets was generated by randomly selecting samples with replacement, ensuring that they contained the same number of samples as the original dataset.
* **Step 2:** Model Predictions  
  For each resampled training set, we computed the predictions yiy\_iyi​ using the respective training set i.
* **Step 3:** Averaging Predictions  
  We then calculated the average prediction as follows:

Given that this task involves binary classification (determining whether student i answered question j correctly), we applied a threshold of 0.5 to derive the final prediction:

* **Step 4:** Performance Evaluation  
  Finally, we compared the ensemble predictions with the validation data to calculate the accuracy of the ensemble model.

**Final Results**

The hyperparameters used included 280 iterations and a learning rate of 0.0005.



**Performance Assessment**The validation accuracy for the ensemble was slightly lower than that of the individual base models, while the test accuracy remained unchanged. It is worth noting that the gap between validation and test accuracy for the ensemble was smaller compared to the original model, indicating that the ensemble approach provided greater prediction stability.

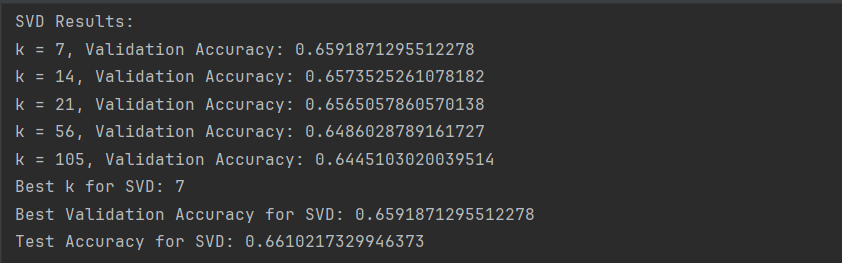
While bagging typically helps average out various sources of noise in the data, thus reducing variance, it does not alter the bias or Bayes error. Theoretically, using an ensemble model should lead to improved accuracy; however, in our case, the use of only three base models may not have been sufficient to achieve a significant boost in performance. Consequently, our test accuracy did not see any improvement.

**4.4. Matrix Factorization**

* **Step 1:** Singular value decomposition (SVD)

We implemented SVD to factorize the response matrix into two smaller matrices representing latent factors for students and questions. For the latent dimension k, we tried five distinct values: .

The validation accuracies for each k value are as follows:



The best k for SVD was found to be 7, and both the test and validation accuracy were approximately 0.66.

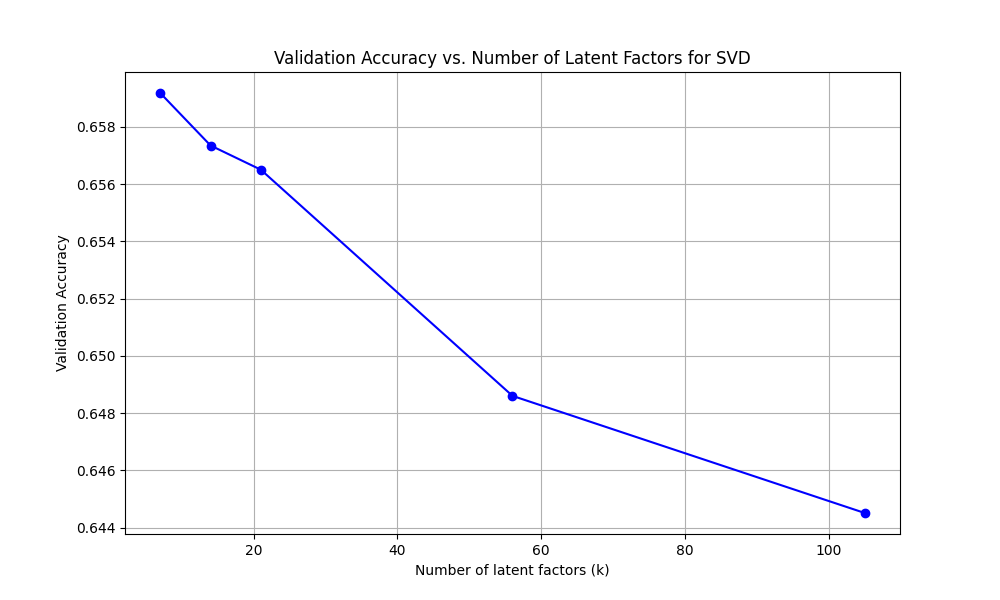
We also plot the results in the following figure:  


Figure 1. The validation accuracy versus the number of latent factors for the SVD method.

This graph shows how the validation accuracy changes with the number of latent factors for the SVD method. The accuracy is highest at the lowest number of factors (around ) and steadily decreases as the number of factors increases. This suggests that for SVD, simpler models (with fewer latent factors) perform better on the validation set, possibly due to better generalization.

* Step 2: Alternating least square (ALS)

In this step, the ALS is implemented using stochastic gradient descent (SGD) by first initializing and iteratively updating the user (U) and question (Z) matrices randomly through the function *update\_u\_z().* Then we use this function in the *als()* function to run the entire process. We also use the same set of latent dimensions k in this step . We also use different values for the learning rate lr to look for promising convergence time and the vali dation accuracy decreased without any fluctuation (unlike some larger learning rates). The *lr* value we choose is 0.1.

The validation accuracies for each k value are as follows:

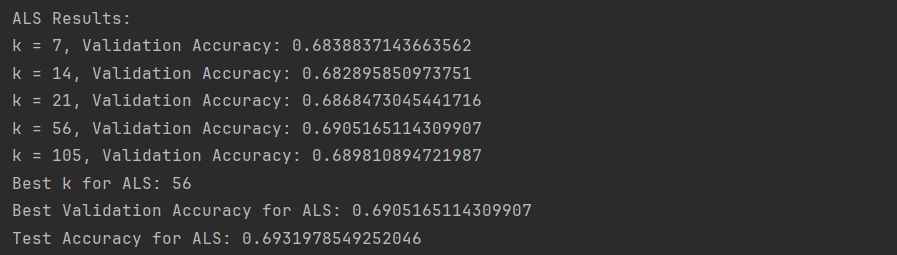


Figure 2. The ALS validation accuracies for each k values

The best k for ALS was found to be 56, with the validation accuracy to be nearly 0.69 and the test accuracy to be approximately 0.693.

The following figure shows the relationship between the validation accuracy and the number of latent factors (k) for the ALS method.

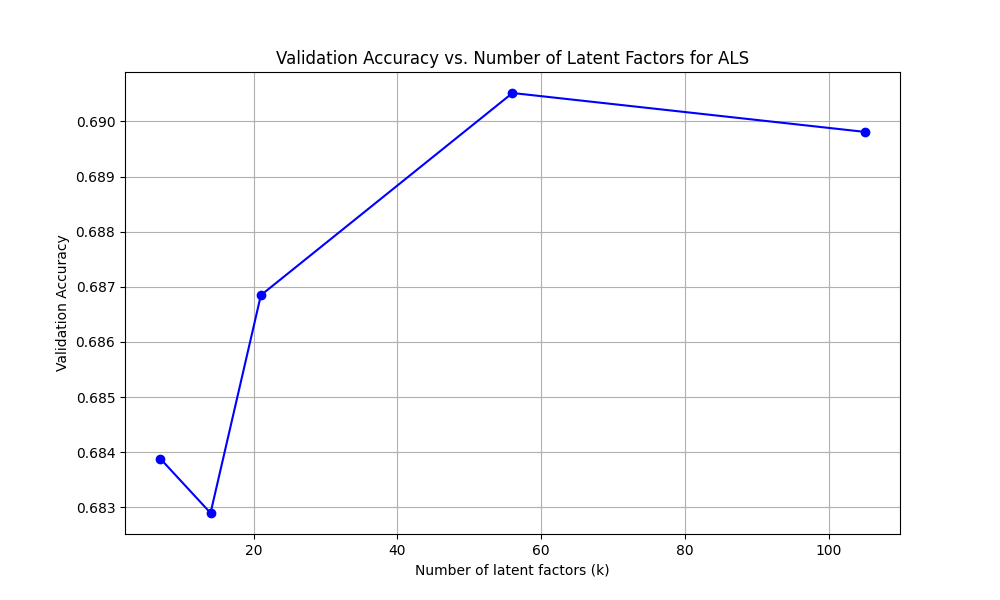


Figure 3. The validation accuracies versus the number of latent factors for the ALS.

The peak accuracy appears to occur around , with a validation accuracy of approximately 0.6905. After this peak, there's a slight decrease in accuracy as k increases to 100, but the decline is minimal.

We also have the following figure to present the training losses and validation losses in each iteration of the best k:

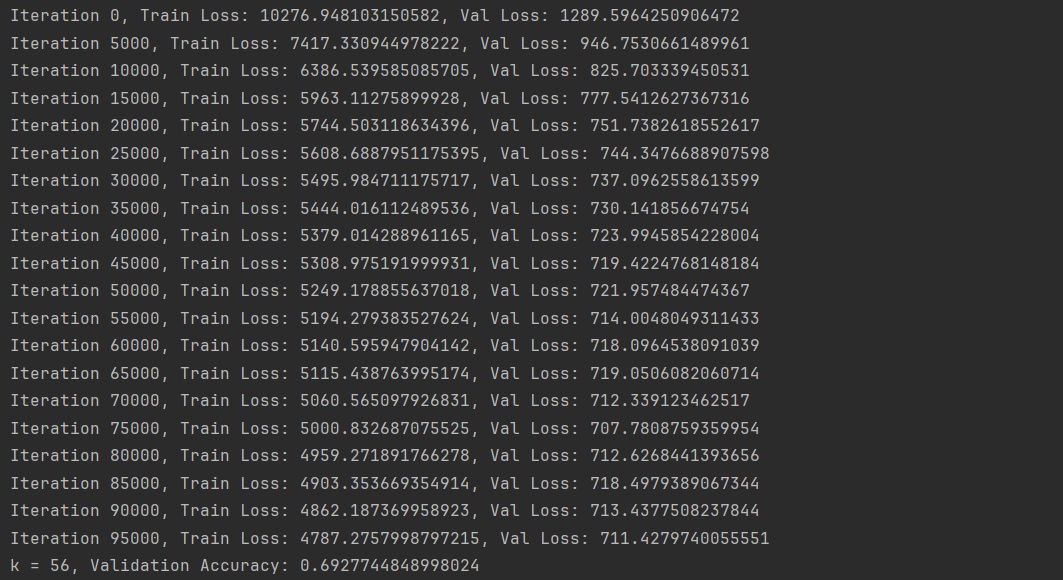


Figure 4. The training losses and validation losses in each iteration of .

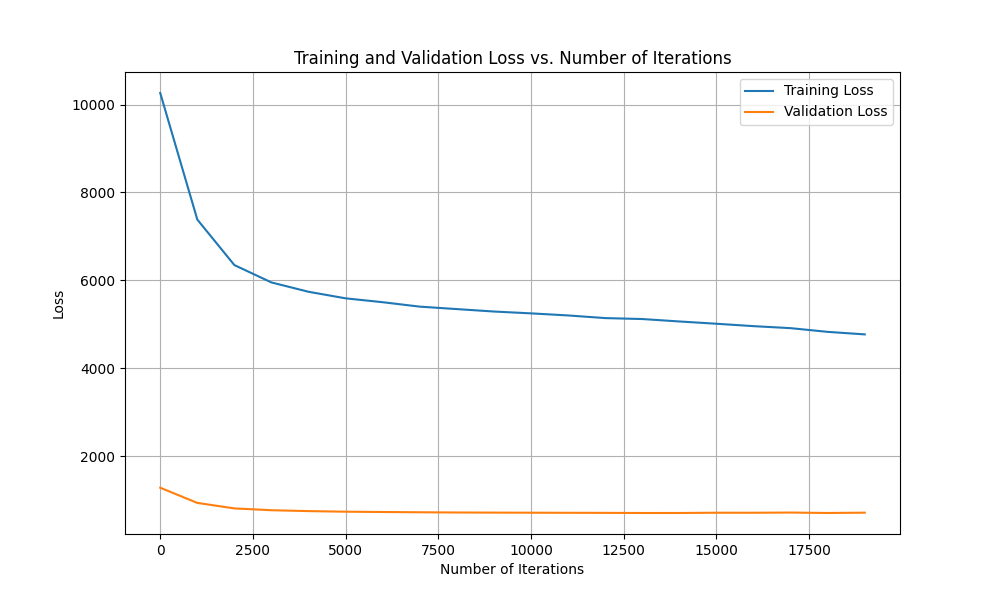


Figure 5. The training and validation loss versus the number of iterations for the ALS method.

This figure illustrates how the training and validation losses change with the number of iterations in the machine learning model. The training loss (blue line) starts very high, and consistently decreases as the number of iterations increases, indicating that the model is learning from the training data. The rate of decrease is rapid initially and then becomes more gradual.

The validation loss (orange line) starts much lower, around 1,500, and also decreases, but at a much slower rate compared to the training loss. It seems to remain unchanged after about 5,000 iterations, with only minimal changes afterwards. This suggests that the model's performance on unseen data stabilizes relatively early in the training process.

The continuing decrease in training loss, coupled with the relatively stable validation loss, indicates that the model is improving its fit to the training data without significant overfitting to the validation set.

* **Step 3:**

1. Comparison of SVD and ALS

We compared the results of SVD and ALS by plotting the validation accuracies for both methods against the number of latent factors (k). This comparison is visualized in the following figure:

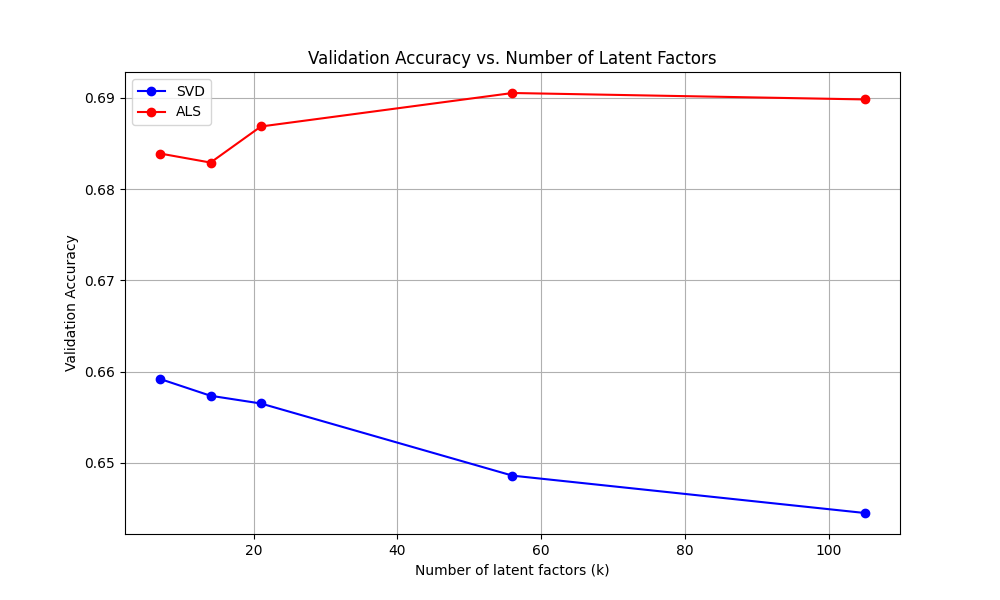


Figure 6. The validation accuracy versus the number of latent factors for both SVD (Singular Value Decomposition) and ALS methods.

This figure compares the validation accuracy of two methods, SVD (Singular Value Decomposition) and ALS (Alternating Least Squares), across different numbers of latent factors.

The ALS method consistently outperforms SVD across all numbers of latent factors. The ALS line shows a general upward trend as the number of latent factors increases, with a slight dip around 20 factors. It reaches its peak around 60 factors and then plateaus.

The SVD line shows a downward trend as the number of latent factors increases, starting at its highest point with 10 factors and declining steadily.

For both methods, the validation accuracy stays within a relatively narrow range (approximately 0.645 to 0.69), suggesting that changing the number of latent factors has a limited impact on overall performance.

2. Limitation of Matrix Factorization

* Handling of Missing Data: In the SVD implementation, we filled in missing values using the average of the current item. This approach may introduce bias, especially if the missing data is not randomly distributed.
* Cold Start Problem: Matrix factorization methods may struggle with new users or items that have no or very few ratings, as there isn't enough information to accurately place them in the latent space.
* Interpretability: The latent factors discovered by these methods are not always easily interpretable, making it challenging to explain why certain predictions are made.
* Scalability: As the number of users and items grows, the computational cost of these methods can increase significantly, especially for larger values of k.
* Assumption of Linearity: These methods assume that the interaction between user and item factors is linear, which may not always hold true in real-world scenarios.
* Temporal Dynamics: Standard matrix factorization doesn't account for changes in user preferences or item characteristics over time, which could be relevant in an educational context where student knowledge evolves.

**Part B: Algorithm Modification**

**Based on the results from Part A, In Part B**, we chose the Item Response Theory (IRT) algorithm to extend and improve prediction accuracy.

**Identify the modification you are making and why it is expected to improve performance**

**• Extension 1: Add parameter αⱼ to describe the discriminative power of the question**

In the original model, there is a simplified assumption that the probability of answering correctly depends only on two parameters:

* **θᵢ**: the ability of student i
* **βⱼ**: the difficulty of question j.

However, assuming that all questions are equally discriminative may not hold true in practice. Therefore, we decided to introduce the parameter **αⱼ** to describe the discriminative power of each question.

Figure 5 in the Figures/Charts section illustrates how different values of **α** affect the probability of answering correctly. When **x** is negative, the blue line is lower than the red line; when **x** is positive, the blue line is higher than the red line. This indicates that students with lower ability are less likely to answer highly discriminative questions correctly, and vice versa.

We add **α** to the model, compute the derivative of the log-likelihood function with respect to **αⱼ**, and apply gradient descent on **α** similarly to **θ** and **β** to optimize the log-likelihood function. Additionally, we constrain **α** to the range (0,2) since this is a more realistic range as suggested by Columbia Public Health. For further details, refer to the IRT theory.

**• Extension 2: Add a hyperparameter c describing the probability of guessing the correct answer**

We assume that all questions are multiple-choice with 4 options (as in the example diagnostic question provided in the materials). Given that students may guess when they do not know the answer, we decided to add a hyperparameter **c** to the model. After tuning, the optimal value of **c** is 0.25.

After this extension, our probability function becomes:

This extended model introduces two additional factors: the discriminative power of questions and the chance of students guessing, adding complexity and better capturing the characteristics of the dataset. Therefore, we expect our extended model to improve optimization and achieve higher test accuracy.

**Evaluate the accuracy of the algorithm**

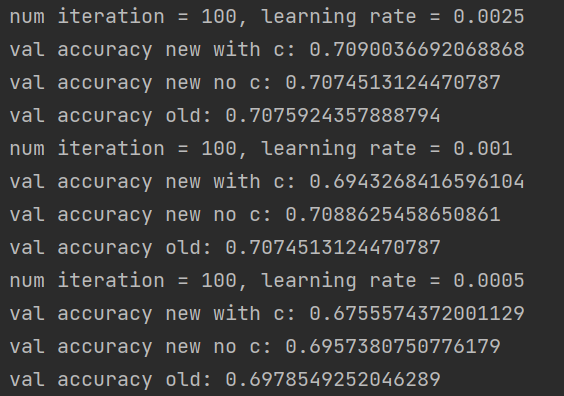
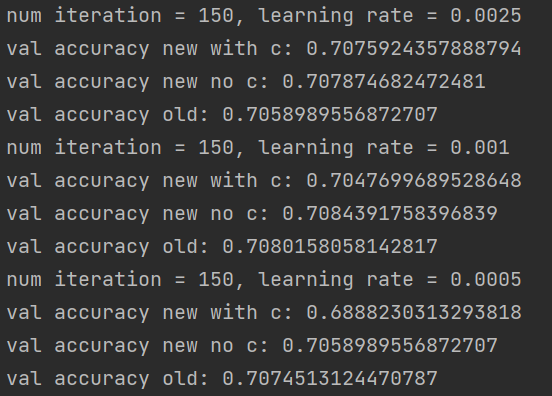
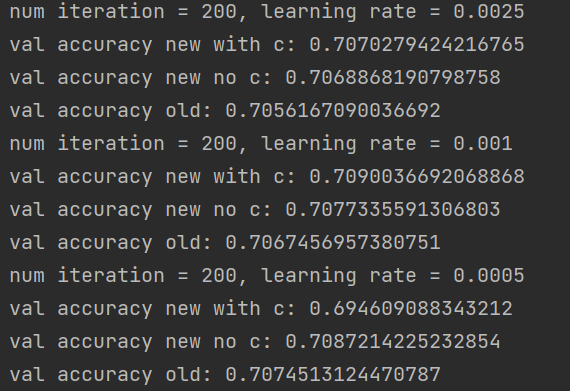
Our hypothesis is that adding parameters **α** and **c** will improve performance because it increases the model's complexity, allowing it to capture more characteristics of the dataset. To test this hypothesis, we will implement the modified model as described earlier and run it on the dataset to evaluate its performance. (The code for this section can be found in \part a\part2 irt.py).

We chose three models to compare the performance of the modified algorithm: the original IRT model, the modified IRT model with **c = 0** (representing no random guessing), and the modified IRT model with **c = 0.25** (representing a 25% chance of answering a question correctly through random guessing).

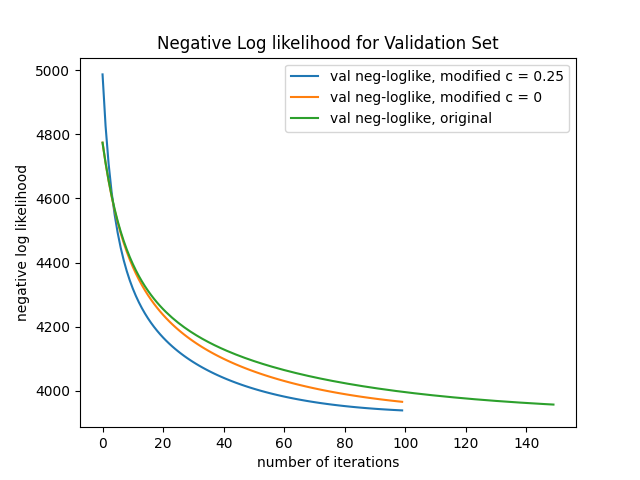
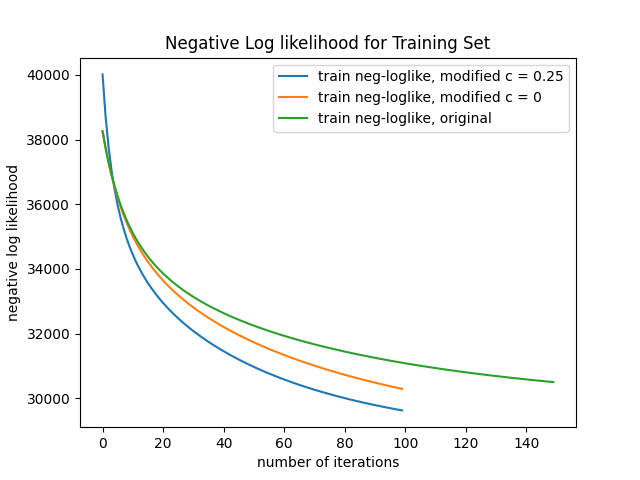
For a set of 9 different hyperparameter combinations, which consist of loop iterations and learning rates , we find the best validation accuracy each model can achieve. The validation accuracies are summarized in the table below:

| Iterations | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 100 | | | 150 | | | 200 | | |
| learning rate | original irt | modified irt,  c= 0 | modified irt,  c= 0.25 | original irt | modified irt,  c= 0 | modified irt,  c= 0.25 | original irt | modified irt,  c= 0 | modified irt,  c= 0.25 |
| 0.0025 | 0.70759 | 0.70745 | 0.70900 | 0.70590 | 0.70787 | 0.70759 | 0.70562 | 0.70687 | 0.70703 |
| 0.001 | 0.70745 | 0.70886 | 0.69433 | 0.70802 | 0.70844 | 0.70447 | 0.70675 | 0.70773 | 0.70900 |
| 0.0005 | 0.69785 | 0.69574 | 0.67556 | 0.70745 | 0.70590 | 0.68882 | 0.70745 | 0.70872 | 0.69461 |

As observed from the table, the maximum validation accuracy achieved by the modified model with **c = 0.25** (0.70900) is higher than that achieved by the modified model with **c = 0** (0.70886). Additionally, both modified models perform better than the original model, which achieved a maximum validation accuracy of 0.70802

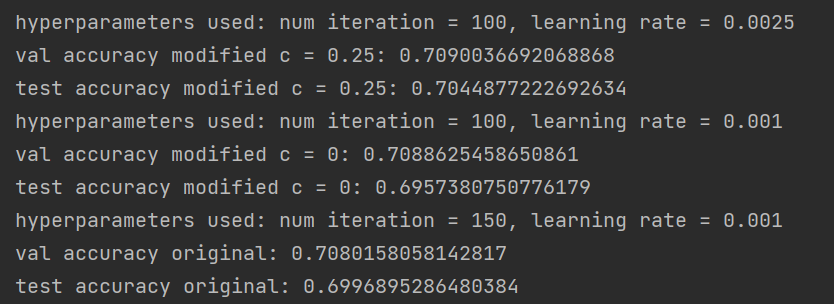
.  

This demonstrates that adding parameters **α** and **c** contributes to performance improvement in some aspects.



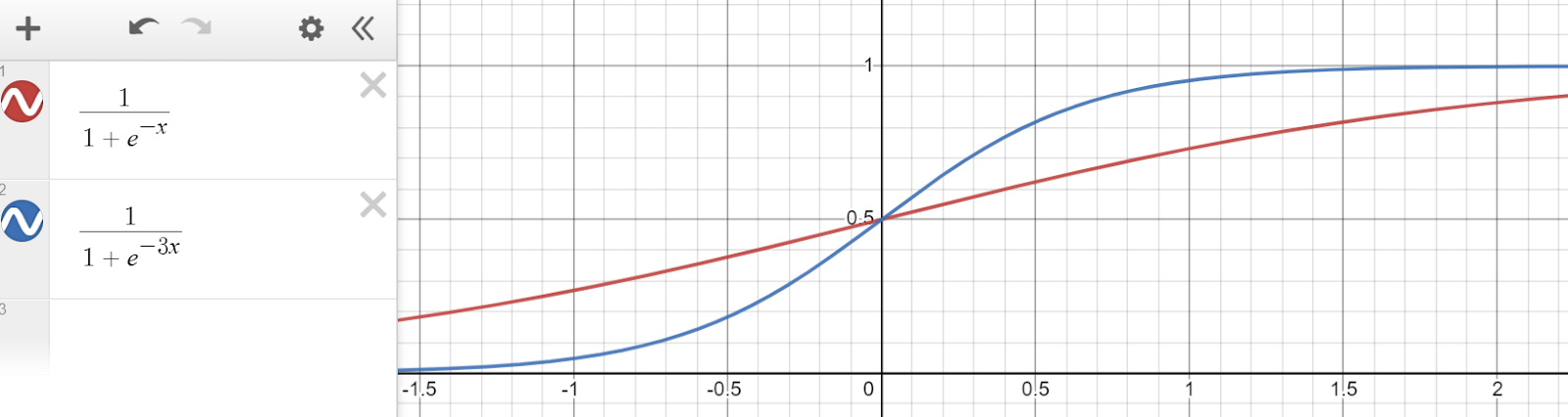
By examining the changes in negative log-likelihood across iterations for both the training and validation sets, we can observe the correspondence between the negative log-likelihood curves and validation accuracies for different models shown in the table. For instance, the negative log-likelihood of the modified model with **c = 0.25** is lower than the other two curves for most iterations, indicating that the prediction likelihood of this modified model is higher. This is consistent with the model having the highest validation accuracy.

The test accuracies for all three models also show improved performance for the modified model with **c = 0.25**.



However, the increase in validation accuracy is not substantial. For the modified model with **c = 0.25**, the slight improvement in accuracy suggests that our assumption of all questions being multiple-choice with four options may not have been ideal. Using a uniform **c** value could have limited further improvements, as different questions may have varying numbers of answer choices.

**Visualization**



*Figure 5: Probability of correct response for different α value*

**Limitations and potential improvements**

Although our extended IRT model has achieved good validation and test accuracies, there are several limitations that affect its performance in certain situations. Below, we summarize these limitations and propose potential extensions to address them.

* **Assumption of Uniform Multiple-Choice Structure**: In this project, we lacked specific information on how many answer options were present for each multiple-choice question. We assumed that all questions had four options with only one correct answer, which might not always hold true. This assumption led us to set a uniform hyperparameter **c** (with a value of 0.25 for four options), which could be inappropriate if different questions have varying numbers of answer choices. A poor choice of **c** may cause underfitting when the data includes questions with different option counts. A possible improvement is to replace the scalar **c** with a vector , where each **cⱼ** represents the probability of guessing the correct answer for question **j**, thus accounting for variable numbers of options across questions.
* **Limited Generalization to Non-Multiple-Choice Questions**: All four models (KNN, IRT, NN, and ensemble) were trained on datasets containing only multiple-choice questions. As a result, these models are expected to perform poorly on datasets with other question types, such as short-answer or fill-in-the-blank questions. Instead of assuming a binary outcome (correct/incorrect) for all questions, one possible extension is to adapt the model for polytomous outcomes, where responses can receive partial credit and be rated on a scale (e.g., 1-5). This would allow the model to handle a wider range of question types more effectively.
* **Single Ability Value Limitation**: In our current IRT model, each student **i** is assigned a single ability score **θᵢ**. Predictions on whether a student can answer question **j** correctly depend solely on their overall ability , the difficulty , and the discriminative power **αⱼ** of the question. This approach fails to account for a student's varying ability across different subjects or topics. For example, a student may excel in most subjects but struggle with specific areas, such as multivariable calculus. Despite their overall high ability score **θᵢ**, the model may overestimate their likelihood of answering a calculus question correctly. To address this issue, we could introduce an ability matrix **θ**, where represents the student's ability in subject **n**. This matrix could be initialized using the provided metadata and refined based on the student's performance on subject-specific questions. By normalizing the matrix and incorporating it into the probability calculation, we could improve the model’s ability to predict performance across diverse subjects. The updated probability for a student answering a question correctly would then be based on their specific abilities in the relevant subject areas, increasing the overall accuracy of the model.