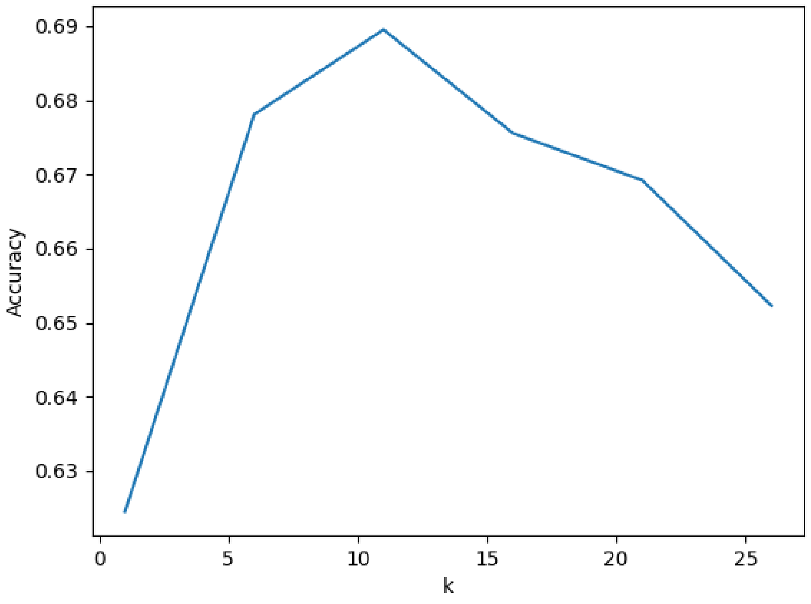
CSC311 Final Project

Jun Li

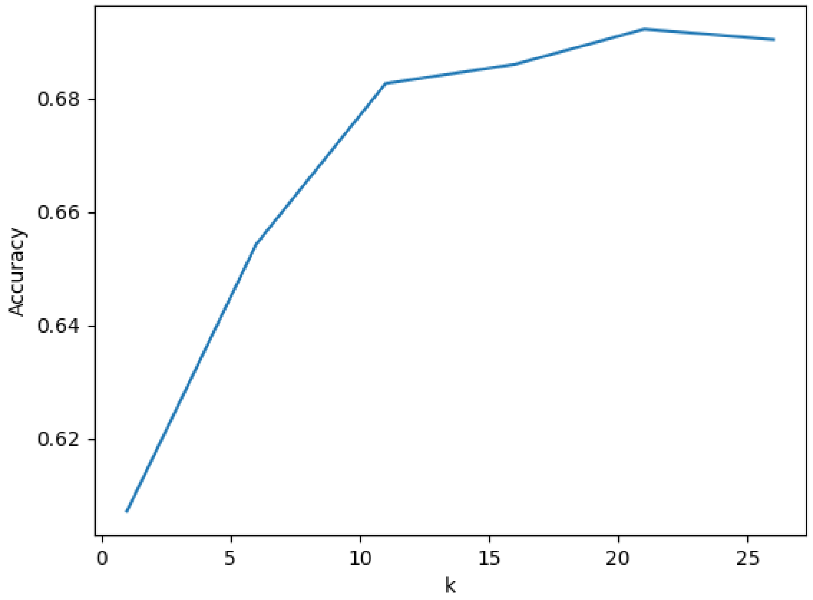
December 1, 2021

1 Part A

1. k-Nearest Neighbor:
(a) The plot of the accuracy on the validation data as a function of k is shown below:



(b) From the above plot, we can see that $k^* = 11$ has the highest performance on validation data. The final test accuracy for user-based collaborative filtering with $k^* = 11$ is 0.6841659610499576.
(c) The plot of the accuracy on the validation data as a function of k is shown below:



From the above plot, we can see that $k^* = 21$ has the highest performance on validation data. The final test accuracy for item-based collaborative filtering with $k^* = 21$ is 0.6816257408975445.
(d) Comparing the test performance results for k^* between user- and item-based collaborative filtering, user-based collaborative filtering performs better by a small margin (0.6841659610499576 vs. 0.6816257408975445).
(e) One of the limitation is the curse of dimensionality. As explained in the lecture, when we have a sample data of high dimensionality like the question data that we have for this project, the points are never close together, hence it breaks down the KNN assumptions since the algorithm relies on the distance between two points to identify the label. Another limitation is the sparsity of the data, or the amount of missing values in the data. For our train data, we can see that there are quite a few of NAs, this will interfere with KNN's ability to determine the labels of their neighbours.

2. Item Response Theory.
(a) We are given that $p(c_{ij} = 1|\theta_i, \beta_j) = \frac{exp(\theta_i - \beta_j)}{1+exp(\theta_i - \beta_j)}$. Then $p(c_{ij} = 0|\theta_i, \beta_j) = 1 - \frac{exp(\theta_i - \beta_j)}{1+exp(\theta_i - \beta_j)} = \frac{1}{1+exp(\theta_i - \beta_j)}$.

The likelihood for all students and questions is:

$$\begin{aligned} p(c|\theta,\beta) &= \Pi_{i=1}^{N}\Pi_{j=1}^{J} p(c_{ij}=1|\theta_i,\beta_j)^{c_{ij}} p(c_{ij}=0|\theta_i,\beta_j)^{1-c_{ij}} \\ &= \Pi_{i=1}^{N}\Pi_{j=1}^{J}[\frac{exp(\theta_i-\beta_j)}{1+exp(\theta_i-\beta_j)}]^{c_{ij}}[\frac{1}{1+exp(\theta_i-\beta_j)}]^{1-c_{ij}} \end{aligned}$$

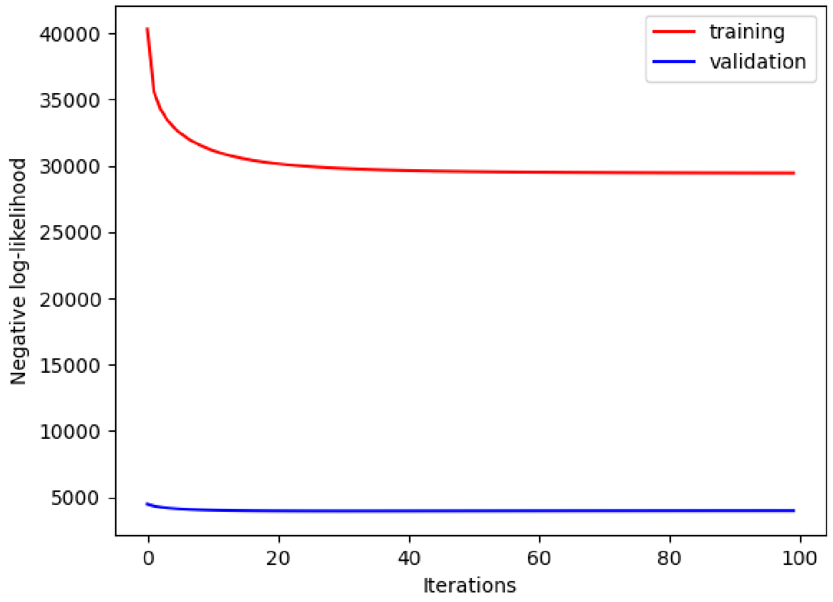
Note that this resembles the likelihood function for logistic regression.

$$\begin{aligned} logp(c|\theta,\beta) &= log\Pi_{i=1}^{N}\Pi_{j=1}^{J}[\frac{exp(\theta_i-\beta_j)}{1+exp(\theta_i-\beta_j)}]^{c_{ij}}[\frac{1}{1+exp(\theta_i-\beta_j)}]^{1-c_{ij}} \\ &= \sum_{i=1}^{N}\sum_{j=1}^{J} log[\frac{exp(\theta_i-\beta_j)}{1+exp(\theta_i-\beta_j)}]^{c_{ij}} + log[\frac{1}{1+exp(\theta_i-\beta_j)}]^{1-c_{ij}} \\ &= \sum_{i=1}^{N}\sum_{j=1}^{J}(c_{ij})log[\frac{exp(\theta_i-\beta_j)}{1+exp(\theta_i-\beta_j)}] + (1-c_{ij})log[\frac{1}{1+exp(\theta_i-\beta_j)}] \\ &= \sum_{i=1}^{N}\sum_{j=1}^{J}(c_{ij})[log(exp(\theta_i-\beta_j)) - log(1+exp(\theta_i-\beta_j))] + (1-c_{ij})[log1 - log(1+exp(\theta_i-\beta_j))] \\ &= \sum_{i=1}^{N}\sum_{j=1}^{J}(c_{ij})(\theta_i-\beta_j) - (c_{ij})log(1+exp(\theta_i-\beta_j)) - log(1+exp(\theta_i-\beta_j)) \\ &+ (c_{ij})log(1+exp(\theta_i-\beta_j)) \\ L(c|\theta,\beta) &= \sum_{i=1}^{N}\sum_{j=1}^{J} c_{ij}\theta_i - c_{ij}\theta_j - log(1+exp(\theta_i-\beta_j)) \end{aligned}$$

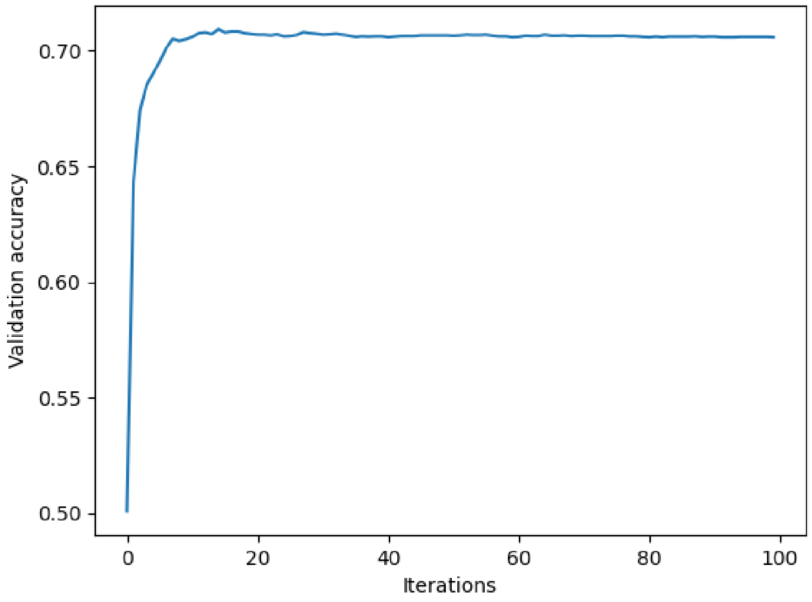
For the gradient descents, we want to obtain the derivative of the log-likelihood with respect to θ_i and β_j.

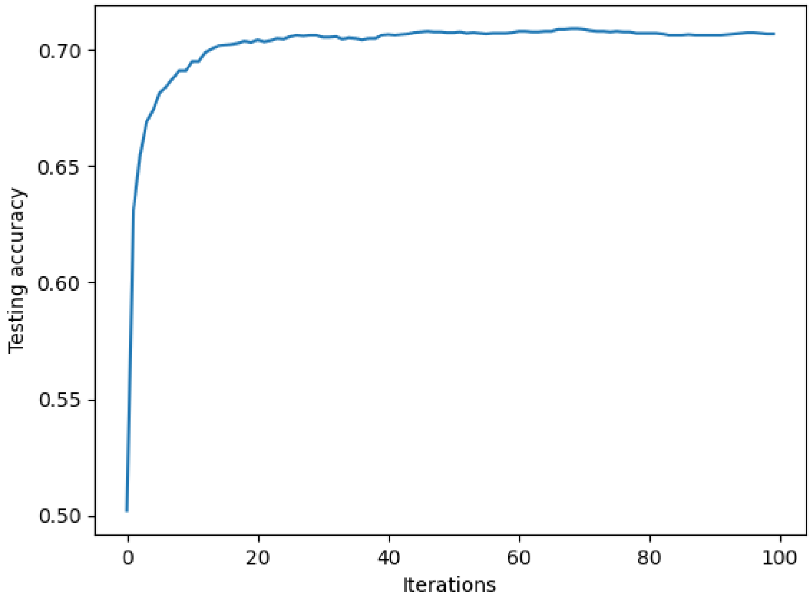
$$\begin{aligned} \frac{\partial L}{\partial \theta_i} &= \frac{\partial}{\partial \theta_i}\sum_{i=1}^{N}\sum_{j=1}^{J} c_{ij}\theta_i - log(1+exp(\theta_i-\beta_j)) \\ &= \sum_{j=1}^{J} c_{ij} - \frac{exp(\theta_i-\beta_j)}{1+exp(\theta_i-\beta_j)} \\ \frac{\partial L}{\partial \beta_j} &= \sum_{i=1}^{N} -c_{ij} + \frac{exp(\theta_i-\beta_j)}{1+exp(\theta_i-\beta_j)} \end{aligned}$$

(b) For the hyperparameters, a learning rate of 0.01 was chosen. The number of iterations was set to be 100 to see if accuracy score would improve with a larger number of iterations (discussed in (c)). The training curve plot is shown below:



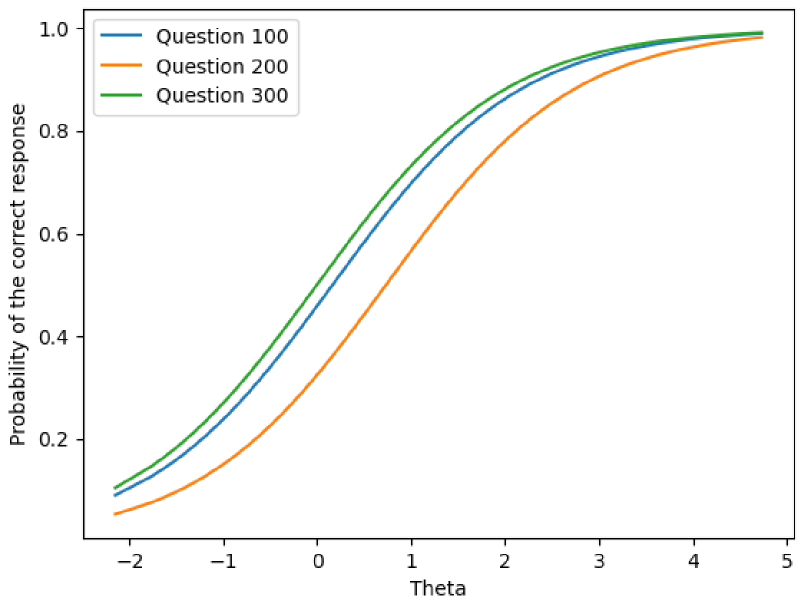
(c) Following the question in (b), validation accuracy is the highest after around 10-20 iterations, and starts oscillating from there. In reality, only 20-30 iterations are needed to achieve the optimal results. There seems to be no pattern for the testing accuracy.





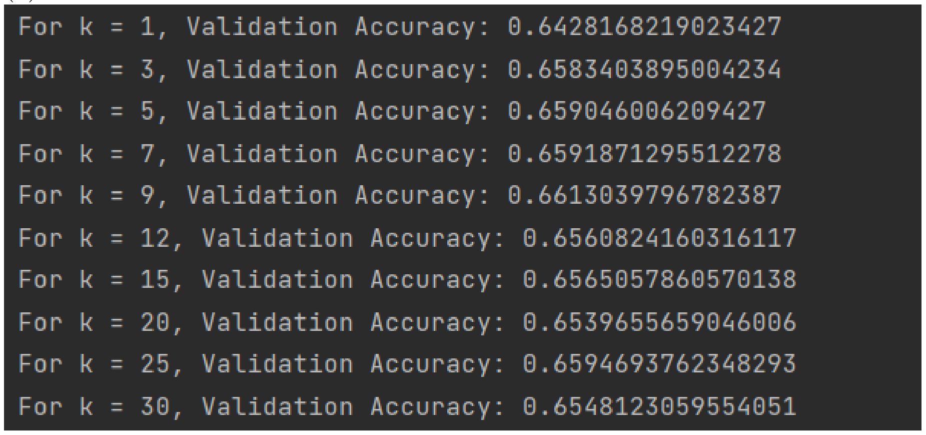
The final validation accuracy is: 0.7092859158904883. The final test accuracy is: 0.7090036692068868. Note that the results vary each time since random θ

and β are selected.
(d) The probability curve is shown below:



Recall that θ represents a student's ability to answer a question. Therefore, as the student's ability increases, the likelihood of obtaining the correct response increases as well. As for the shape of the curve, it is similar to the curve of a sigmoid function.

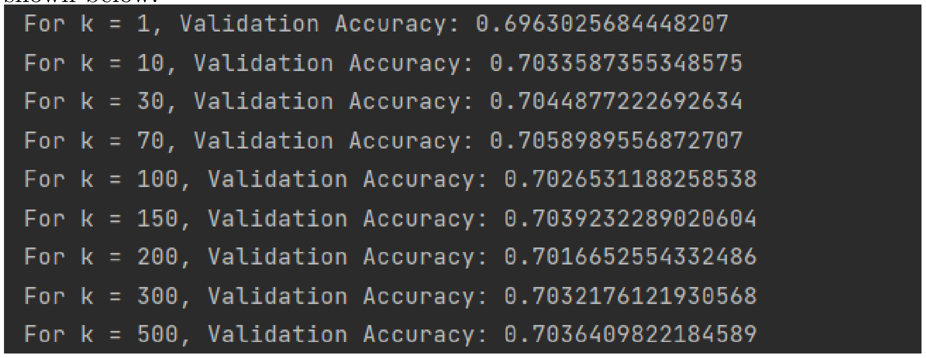
3. Matrix Factorization:
(a) The validation accuracies for different k are shown below:



```
For k = 1, Validation Accuracy: 0.6428168219023427
For k = 3, Validation Accuracy: 0.6583403895004234
For k = 5, Validation Accuracy: 0.659046006209427
For k = 7, Validation Accuracy: 0.6591871295512278
For k = 9, Validation Accuracy: 0.6613039796782387
For k = 12, Validation Accuracy: 0.6560824160316117
For k = 15, Validation Accuracy: 0.6565057860570138
For k = 20, Validation Accuracy: 0.6539655659046006
For k = 25, Validation Accuracy: 0.6594693762348293
For k = 30, Validation Accuracy: 0.6548123059554051
```

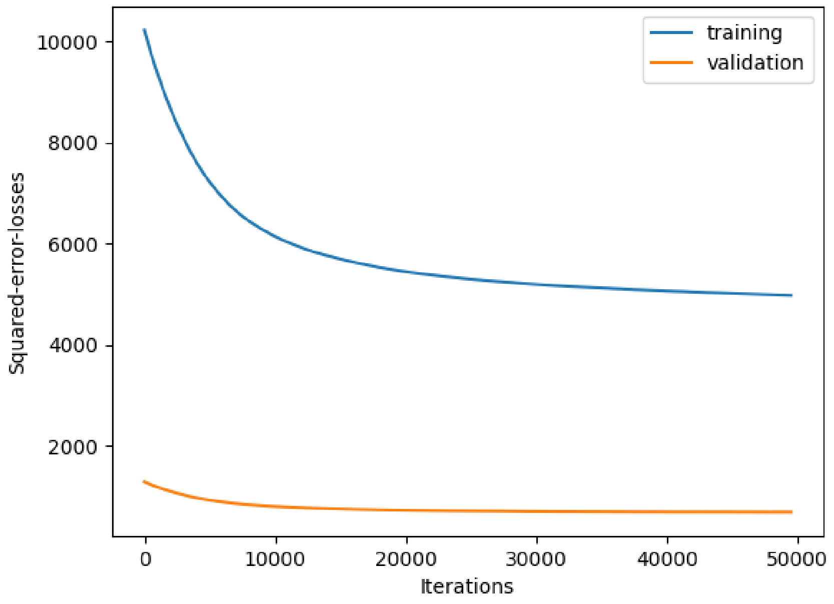
The final validation accuracy with $k^* = 9$ is 0.6613039796782387. The final test

accuracy with $k^* = 9$ is 0.6587637595258256.
(b) For SVD calculation, we used the average on the current item to fill in the missing values. This might not be the optimal method. It has an impact on the item variability since we are filling all missing values with one single mean value.
(d) For the hyperparameters, a learning rate of 0.01 was chosen. The number of iterations was set to be 500000. The validation accuracies for different k are shown below:



```
For k = 1, Validation Accuracy: 0.6963025684448207
For k = 10, Validation Accuracy: 0.7033587355348575
For k = 30, Validation Accuracy: 0.7044877222692634
For k = 70, Validation Accuracy: 0.7058989556872707
For k = 100, Validation Accuracy: 0.7026531188258538
For k = 150, Validation Accuracy: 0.7039232289020604
For k = 200, Validation Accuracy: 0.7016652554332486
For k = 300, Validation Accuracy: 0.7032176121930568
For k = 500, Validation Accuracy: 0.7036409822184589
```

The best k^* that achieves the highest validation accuracy is $k^* = 70$.
(e) The squared-error-losses plot is shown below:



The validation and training errors decrease as the number of itractions increase. This is expected because we are minimizing latent vectors until convergence, so the results are getting more accurate with each iteration.

The final validation accuracy with $k^* = 70$ is 0.7058989556872707. The final test accuracy with $k^* = 70$ is 0.7013830087496472.

4. Ensemble:
For the bagging ensemble, because this ensemble method is typically done with one base algorithm, I chose to use KNN algorithm with three randomized k values between 10 to 20. Usually more base models would be implemented in the ensemble, so they do not have to be stable models or use extensive data sets. But since we are only using three models here, I thought it would be better to choose a more stable model, hence the choice of KNN.
The bootstrapping was done by generating random indexes with replacement for the items in the sparse matrix, the sample size remains the same as the original training matrix. The same algorithm was used to implement user-based collaborative filtering. The only change was that, when calculating the rating matrix, we take the average of the three matrices and evaluate the average matrix to obtain the accuracy scores.
The final validation accuracy is 0.6604572396274344. The final testing accuracy is 0.6514253457521874. This performance is worse than the baseline accuracy of around 0.68. This is to be expected. As mentioned above, ensemble methods generally generate a lot more data groupings than the 3 groups that we have. Hence the requirements for the data quality and the model quality are lower than if we were to implement a single model. Also for this question, the three datasets would have duplicate entries in them, which would likely result in some prediction inaccuracy. Lastly, KNN is already a stable algorithm, so there will not be a big improvement in terms of accuracy using bagging ensemble.

2 Part B

(1) For the modification of the algorithm, we will perform it on the matrix factorization algorithm implemented in Part A. For this, weighted matrix factorization (WMF) is used. Recall that our original objective was:

$$min_{\mathbf{U},\mathbf{Z}}\frac{1}{2}\sum_{(n,m)\in O}(C_{nm}-\mathbf{u}_n^T\mathbf{z}_m)^2$$

To avoid over-fitting, regularization terms are added to the function and penalizes higher degree polynomials. This will reduce the generalization error. The function then becomes:

$$min_{\mathbf{U},\mathbf{Z}}\frac{1}{2}\sum_{(n,m)\in O}(C_{nm}-\mathbf{u}_n^T\mathbf{z}_m)^2+\lambda_{\mathbf{u}}||\mathbf{u}||^2+\lambda_{\mathbf{z}}||\mathbf{z}||^2$$

where $\lambda_{\mathbf{u}}$ and $\lambda_{\mathbf{z}}$ are used to adjust the importance of the latent feature matrices $\mathbf{u}$ and $\mathbf{z}$. For the weighting component, we add a weight matrix $\mathbf{W}_{nm}=1+log(1+C_{nm}\times 10^{\epsilon})$ where ϵ is a constant used to control the rate of increment. After reformulating the function, we have:

$$min_{\mathbf{U},\mathbf{Z}}\frac{1}{2}\sum_{(n,m)\in O}\mathbf{W}_{nm}(C_{nm}-\mathbf{u}_n^T\mathbf{z}_m)^2+\lambda_{\mathbf{u}}||\mathbf{u}||^2+\lambda_{\mathbf{z}}||\mathbf{z}||^2$$

Taking the partial derivatives with respect to $\mathbf{u}_n$ and $\mathbf{z}_m$, we obtain:

$$\frac{\partial L}{\partial \mathbf{u}_n}=\sum_{(n,m)\in O}\mathbf{W}_{nm}(C_{nm}-\mathbf{u}_n^T\mathbf{z}_m)\mathbf{z}_m^T+\lambda_{\mathbf{u}}\mathbf{u}_n^T$$

$$\frac{\partial L}{\partial \mathbf{z}_m}=\sum_{(n,m)\in O}\mathbf{W}_{nm}(C_{nm}-\mathbf{u}_n^T\mathbf{z}_m)\mathbf{u}_n^T+\lambda_{\mathbf{z}}\mathbf{z}_m^T$$

And update the gradient descents:

$$\mathbf{u}_n:=\mathbf{u}_n-\alpha\sum_{(n,m)\in O}\mathbf{W}_{nm}(C_{nm}-\mathbf{u}_n^T\mathbf{z}_m)\mathbf{z}_m^T+\lambda_{\mathbf{u}}\mathbf{u}_n^T$$

$$\mathbf{z}_m:=\mathbf{z}_m-\alpha\sum_{(n,m)\in O}\mathbf{W}_{nm}(C_{nm}-\mathbf{u}_n^T\mathbf{z}_m)\mathbf{u}_n^T+\lambda_{\mathbf{z}}\mathbf{z}_m^T$$

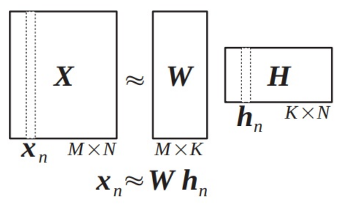
where α is the learning rate. For this model, we are using ALS to compute the latent feature matrices. A simple algorithm box is shown below:

Algorithm 1 Weighted SGD Matrix Factorization

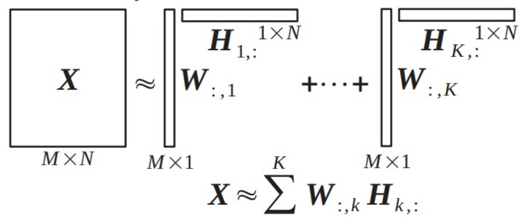
```
Input: training matrix $V$, an integer $k$, learning rate $\alpha$, regularization meter
    $\lambda$, weighting meter $\epsilon$
Output: rating matrix $M$
1: initialize $\mathbf{u}$ and $\mathbf{z}$ to UniformReal(0, $\frac{1}{\sqrt{k}}$)
2: repeat
3:     for random $V_{nm} \in V$ do
4:         $W_{nm} = 1 + log(1 + C_{nm} \times 10^{\epsilon})$
5:         $\mathbf{u}_{n*} := \mathbf{u}_{n*} - \alpha \mathbf{W}_{nm}(C_{nm} - \mathbf{u}_{n*}^T \mathbf{z}_{*m})\mathbf{z}_{*m}^T + \lambda_{\mathbf{u}} \mathbf{u}_{n*}^T$
6:         $\mathbf{z}_{*m} := \mathbf{z}_{*m} - \alpha \mathbf{W}_{nm}(C_{nm} - \mathbf{u}_{n*}^T \mathbf{z}_{*m})\mathbf{u}_{n*}^T + \lambda_{\mathbf{u}} \mathbf{z}_{*m}^T$
7:     end for
8: until convergence
```

(2) The overall model digram is shown below:

$$min_{\mathbf{W},\mathbf{H}} \frac{1}{2} \sum_{(M,N)\in O} \mathbf{Weight}(C_{MN} - \mathbf{W}_M^T \mathbf{H}_N)^2 + \lambda_{\mathbf{W}} ||\mathbf{W}||_F^2 + \lambda_{\mathbf{H}} ||\mathbf{H}||_F^2$$



$\boldsymbol{x}_n$ is approximated by a linear combination of the columns of $\boldsymbol{W}$



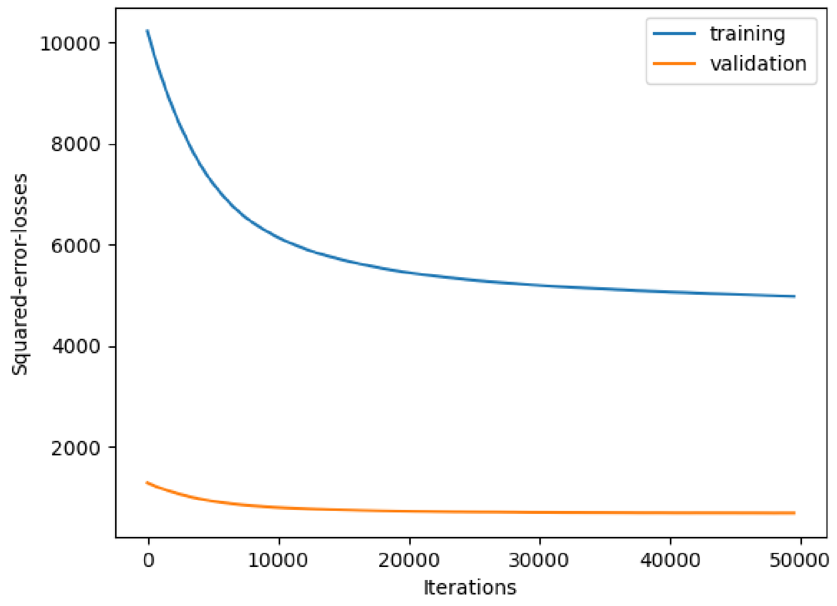
$\boldsymbol{X}$ is approximated by a linear combination of rank-1 matrices

(3) A table for the validation accuracy with different k values is shown below. The same random seed is set for both the baseline and modified model for the comparison.

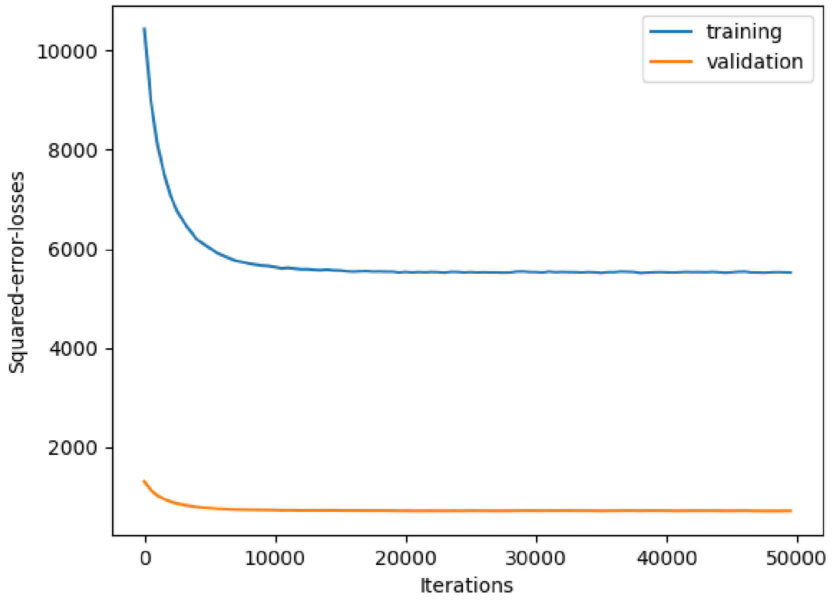
	Baseline model		Modified model
k=1	0.7006773920406435	k=1	0.6895520180637877
k=10	0.7027942421676545	k=2	0.6879762912785775
k=30	0.6979960485464296	k=4	0.6930567315834039
k=70	0.7044877222692634	k=6	0.6944679650014113
k=100	0.7053344623200677	k=7	0.6909398814563928
k=150	0.7005362686988428	k=8	0.6947502116850127
k=200	0.7011007620660458	k=9	0.6892464013547841
k=300	0.703076488851256	k=10	0.6902342647473892
k=500	0.705193338978267	k=15	0.6917866215071973

For the baseline model:
The final validation accuracy with $k* = 100$ is 0.7053344623200677.
The final test accuracy with $k* = 100$ is 0.7019475021168501.
For the modified model:
The final validation accuracy with $k* = 8$ is 0.6947502116850127.
The final test accuracy with $k* = 8$ is 0.6844482077335591.

The squared-error-loss plots are also included to check the fitting:
Baseline model:



Modified model:



Overall the models performed similarly in terms of accuracy, with the baseline model slightly out-performing the modified model. This can be due to many factors. For example, the modified model added 2 new hyperparameters to be tuned manually, this increases the probability of the model set-up being not optimal, hence more room for error. It can also be that there is an insufficient number of examples for training. Or it is even possible that weighted matrix factorization may not be the best model for our prediction problem.
Taking outside of the accuracy, we observe that the modified increases the speed of convergence: on the squared-error-loss plot, the modified model converges a lot quicker than the baseline model. This is expected with the addition of regularization, since we have reduced generalization loss. However, we do still face some problems with the fit for both models: the validation and training curve are far away from each other, with the validation losses being a lot lower than the training losses. This is counter-intuitive to think in machine learning, because the model is learning from the training data to predict the validation data! This could be an indication that there is something wrong with our data: it can be that there is not enough training data like mentioned, since we do have sparsity in the data, making it harder to learn for the model. Another probability is that our validation data happens to be very easy to predict. We also encounter a loss plateau in the modified model, and this does not improve even with decreased learning rate. So the best solution to the fit problem is to improve the data quality.
(d)
Generally speaking, one of the limitations for matrix factorization is the cold

start problem. The cold start problem concerns the system's inability to recommend new unrated items, or items that has no interaction with the users. This is also a common problem in all collaborative filtering models. This is because collaborative filtering models rely on the item's interactions to make recommendations. A solution to the problem for item-based matrix factorization is term frequency–inverse document frequency, short for TF-IDF. This method generates the recommendation for a user based on the data's provided by the user profile. Then we can use collective factorization with TF-IDF to obtain the the estimated item rating from the user.
Another limitation is the sparsity level of the data. With lower sparsity, there is more information to train the model with, hence the error will likely be reduced. This is highly related to the cold-start problem explained above. A solution to this is to use the content-boosted matrix factorization model. Content-boosted collaborative uses item-item similarity or user-user similarity to convert a sparse interaction matrix to a dense matrix. Then collaborative filtering models can be used from there, in our case it is matrix factorization.
Another problem specific to our project is the data quality, this is explained above in the results section.

3 Reference

1. https://sw.cs.wwu.edu/ tuora/aarontuor/materials/presentations/poster5.pdf
2. Gillis, Glineur, F. (2012). A multilevel approach for nonnegative matrix factorization. Journal of Computational and Applied Mathematics, 236(7), 1708–1723. https://doi.org/10.1016/j.cam.2011.10.002
3. Nguyen, Zhu, M. (2013). Content-boosted matrix factorization techniques for recommender systems. Statistical Analysis and Data Mining, 6(4), 286–301. https://doi.org/10.1002/sam.11184
4. Ranjbar, Moradi, P., Azami, M., Jalili, M. (2015). An imputation-based matrix factorization method for improving accuracy of collaborative filtering systems. Engineering Applications of Artificial Intelligence, 46, 58–66. https://doi.org/10.1016/j.engappai.2015.08.010
5. Tran, Lee, K., Liao, Y., Lee, D. (2018). Regularizing Matrix Factorization with User and Item Embeddings for Recommendation. Proceedings of the 27th ACM International Conference on Information and Knowledge Management, 687–696.
https://doi.org/10.1145/3269206.3271730
6. Wang, Peng, H., Jin, Y., Sha, C., Wang, X. (2017). Local Weighted Matrix Factorization for Top-n Recommendation with Implicit Feedback. Data Science and Engineering, 1(4), 252–264. https://doi.org/10.1007/s41019-017-0032-6
7. Yifeng Li. (2016). Advances in multi-view matrix factorizations. 2016 International Joint Conference on Neural Networks (IJCNN), 3793–3800.
https://doi.org/10.1109/IJCNN.2016.7727689