Deep Matrix Factorization for Collaborative Filtering in Recommender Systems

EQUIPE-404 : Alexandre Ngau, Thomas Boudras and Mathilde Kretz[1]

[1]PSL Research University - Data Science Project

October 2023

The objective of this paper is to report the work done during the time devoted for the first assignment of the Data Science Lab course. It will recall the context we are dealing with, the two main techniques that have been implemented to resolve the problem and the results obtained.

1 Context of collaborative filtering

We briefly recall the context of the assignment and set the notations for the future explanations.

Collaborative Filtering is a method used to build recommender systems. In practice, the problem is modeled by a rating matrix R for which $R_{i,j}$ is the rating of user i for item j with $i \in [\![0, m]\!]$ and $j \in [\![0, n]\!]$. The objective of collaborative filtering is to predict notations of items from users, knowing how they rate other items, using the R matrix.

For this assignment, the rating matrix used to operate the resolution contains $m = 610$ users and $n = 4980$ items to rate. The vast majority of this matrix is empty, as users generally rate few items. The following figures 1 2 show the distribution of 'missing values' that are the representation of items that have not been rated by users.

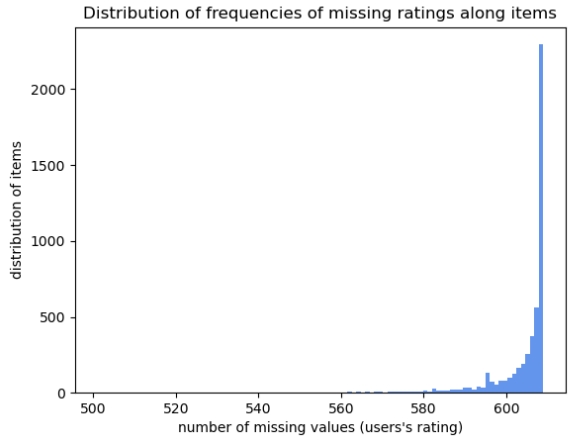


Figure 1: Frequency distribution of missing values by items.

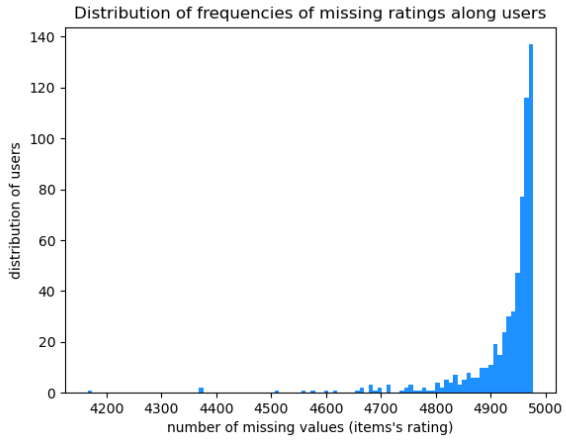


Figure 2: Frequency distribution of missing values by users.

The matrix is indeed essentially composed of missing ratings despite its huge size.

Through the entire problem resolution, the rating matrix is split into two matrices of same size where ratings have been randomly distributed between both. One, the `train` matrix, is used to do the training, and the other, the `test` matrix, is used to do the testing.

2 Matrix factorization

The technique we have chosen to implement and present is the matrix factorization method. This will be the main focus of this report.

2.1 Formalism

To quickly recall the principle of matrix factorization, the idea is to approximate the rating matrix R by the following :

$$R \approx I \times U^T$$

with $R \in \mathbb{R}^{m \times n}$, $I \in \mathbb{R}^{m \times k}$ and $U \in \mathbb{R}^{n \times k}$. R is then approximated at the rank k.

2.2 First approach : Gradient Descent

The first approach implemented has been gradient descent, in order to find the minimum of the function $C(I,U) = \|R - IU^T\|$. As seen in class, in this case we end up with the following formulas for gradient descent algorithm:

$$I_{t+1} = I_t - \eta_{U,t}\frac{\partial C}{\partial I}(I_t, U_t)$$
$$U_{t+1} = U_t - \eta_{I,t}\frac{\partial C}{\partial U}(I_t, U_t)$$

With the following gradients computed at each step:

$$\frac{\partial C}{\partial U} = -2 \cdot (R^T - UI^T) \cdot I$$
$$\frac{\partial C}{\partial I} = -2 \cdot (R - IU^T) \cdot U$$

2.3 Implementation, results and analysis

2.3.1 Classical Gradient Descent

The first implementation was the classical gradient descent method, improved over time with different methods. The principal performance criterion for the choice of parameters was the RMSE (Root Mean Square Error) and the accuracy. For the classical gradient descent, the following choices have been made:

Both the learning rates of the two matrices in the factorization η_U and η_I have to be determined in order to maximize the convergence rate. In order to identify them, they were chosen too big and diminished until the norm of the gradient ceased to diverge. This method allowed us to obtain $\eta_I = 1e-4$ and $\eta_U = 1e-3$.

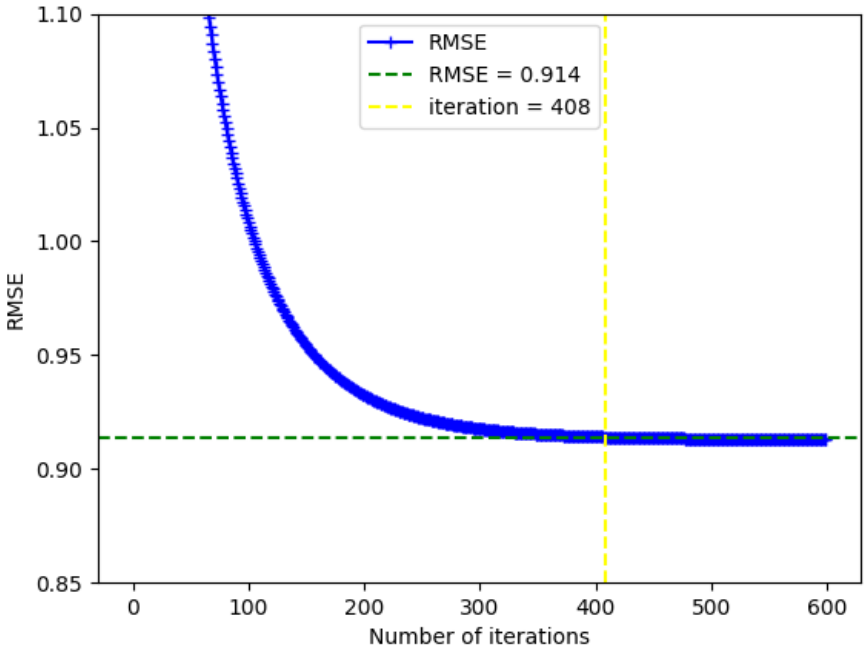


Figure 3: Frequency distribution of missing values by items.

The loop of the gradient descent is stopped by a chosen number N of iterations. This number N is found by plotting the RMSE between the predicted matrix $R_{prediction} = U \cdot I^T$ and the test matrix as a function of

the iterations. The number of iterations N is identified when the RMSE begins to stagnate. In this particular situation 3, the gradient descent is stopped at the 408th iteration.

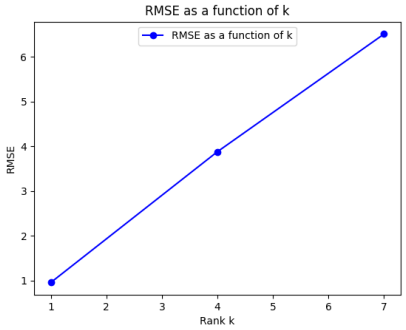
Besides, it's necessary to choose k, the rank of both of the matrixes I and U. The prediction with different values of rank k is computed and the associated RMSE calculated. The figure 4a provides a visualization of the RMSE as a function of the rank. For the classic matrix factorization, the best rank of factorization was found to be $k = 1$.

The gradient descent computed with these parameters gave an RMSE of 0,914 and an accuracy of 0.00% after 408 iterations.

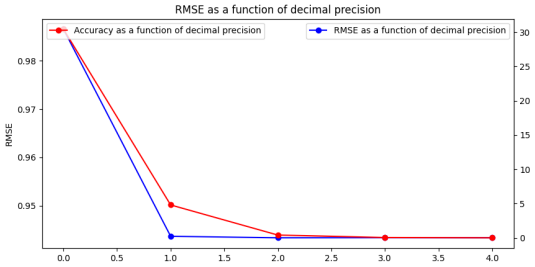
The reason why the accuracy is null is that the algorithm creates a matrix $R_{predicted} = I \cdot U^T$ which is composed of floats. When it is compared with the test matrix, which is composed of integers, it is normal that there are barely none of the ratings that are perfectly predicted.

In order to avoid this, the ratings in the final predicted matrix $R_{predicted}$ are rounded. To know if this creates a huge impact on the RMSE, the RMSE is displayed as a function of the decimal precision of ratings figure 4b.

This plot 4b shows that the rounding to the integer value causes a high increase of the RMSE, on the contrary the rounding to the tenth results in a negligible increase in RMSE, while it allows a 5% increase of the accuracy.



(a) Frequency distribution of missing values by items.



(b) Frequency distribution of missing values by items.

Figure 4: Frequency distribution of missing values by items.

2.3.2 Regularized Gradient Descent

In order to be sure that the model doesn't overfit, a ridge regularization is implemented to the gradient descent.

The gradient formulas then becomes:

$$\frac{\partial C}{\partial U} = -2 \cdot (R^T - UI^T) \cdot I + 2\lambda_I U$$
$$\frac{\partial C}{\partial I} = -2 \cdot (R - IU^T) \cdot U + 2\lambda_U I$$

with λ_I and λ_U positive real coefficients.

The optimal coefficients of the ridge method λ_I and λ_U that gives the best results have to be determined. To do this, the RMSE for several values of λ_I and λ_U are calculated and those that give the lowest RMSE are kept. These are : $\lambda_I = 1.00$ and $\lambda_U = 1e-1$. The results are then as follows: with a RMSE of 0.917, and an accuracy of 4,8 after 1381 iterations.

For the classic matrix factorization, the penalty ridge doesn't give very conclusive results. Indeed, similar RMSE and accuracy values are obtained with a much higher number of iterations. The algorithm is therefore less efficient with the penalty ridge.

Finally, a learning rate scheduling is set up, by decreasing the stepsize by $x\%$ every 100 iterations. The aim is to reduce the learning rate so that it adjusts optimally as it approaches the minimum. The display of the

RMSE and accuracy as a function of the decay rate can be observed figure 5, the decay rate is the percentage decrease in the gradient descent coefficient every 100 iterations.

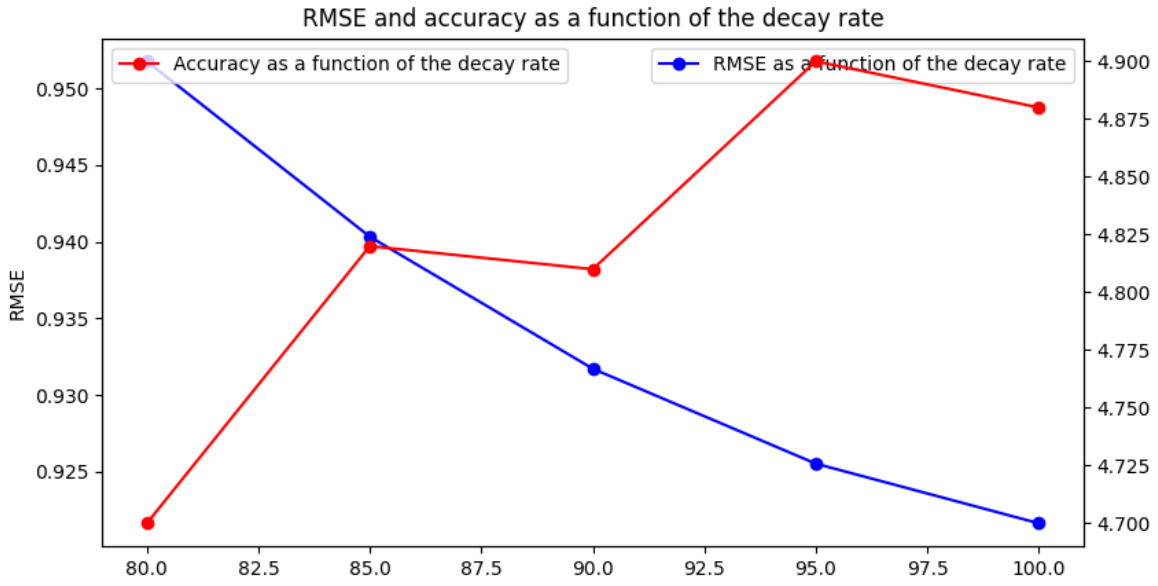


Figure 5: Frequency distribution of missing values by items.

The more the decay rate decreases, the higher the RMSE and the lower the accuracy. Finally the implementation of learning rate scheduling has only led to a decrease in the algorithm's efficiency.

The code for the Regularized Gradient Descent can be found in the `MF_by_GD.py` file.

2.4 Marginal approach : SVD

Marginally, a technique developed in paper (1) has been studied and implemented under the `svdGD.py` file. The idea of the algorithm presented in the paper is to approximate the rating matrix with the singular values decomposition and then create a h-neighborhood environment to predict the unknown ratings with the closest neighbors only. The singular value decomposition aims to find the eigenvalues of the covariance matrix in order to find the directions that have the highest variance i.e. that represent the most the original matrix. It is computed thanks to the `numpy` package but can't work with a sparse matrix, missing values are then completed in the paper with the columns average. A problem arises when the prediction matrix is very similar to the rating matrix filled with the averages, so a solution implemented in our work used the matrix predicted by the gradient descend from 2.3 as an input of the SVD. In face of the computation time and the lack of proper results, the approach has been abandoned.

3 Deep matrix factorization (DeepMF)

3.1 General idea

The work conducted in this part of the assignment is based on a paper (2). The general idea behind this paper is to incorporate the philosophy of deep learning into the matrix factorization for collaborative filtering paradigm without making use of neural networks. Instead, the algorithm called *DeepMF*, transforms the matrix factorization process into a layered process that auto-correct itself layer after layer, by factorizing out the expected errors of the precedent layers.

This method aims at improving the quality of the predictions and the recommendations.

Consider the sparse matrix R of known ratings. The first step of *DeepMF* is a classical matrix factorization where $R \approx I_0 U_0^T$. Then the first degree error matrix $E_1 = R - I_0 U_0^T$ is factorized again in order to predict the expected error, such that $E_1 \approx I_1 U_1^T$. This gives us a second degree error matrix factorized again $E_2 = E_1 - I_1 U_1^T \approx I_2 U_2^T$. The factorization process is then repeated as many times as desired, the paper claiming that the sequence of error matrices $(E_s)_{s \in \mathbb{N}^*}$ converges to zero, providing preciser predictions layer after layer.

After N iterations, the predicted matrix $\hat{R}$ is recovered by summing the different factorizations :

$$\hat{R} = \sum_{s=0}^{N} I_s U_s^T$$

A visual explanation is provided figure 6 and comes from the paper (2). It should be noted that the report uses $I \cdot U^T$ as notation for the matrix factorization, while $P \cdot Q$ is used in the paper.

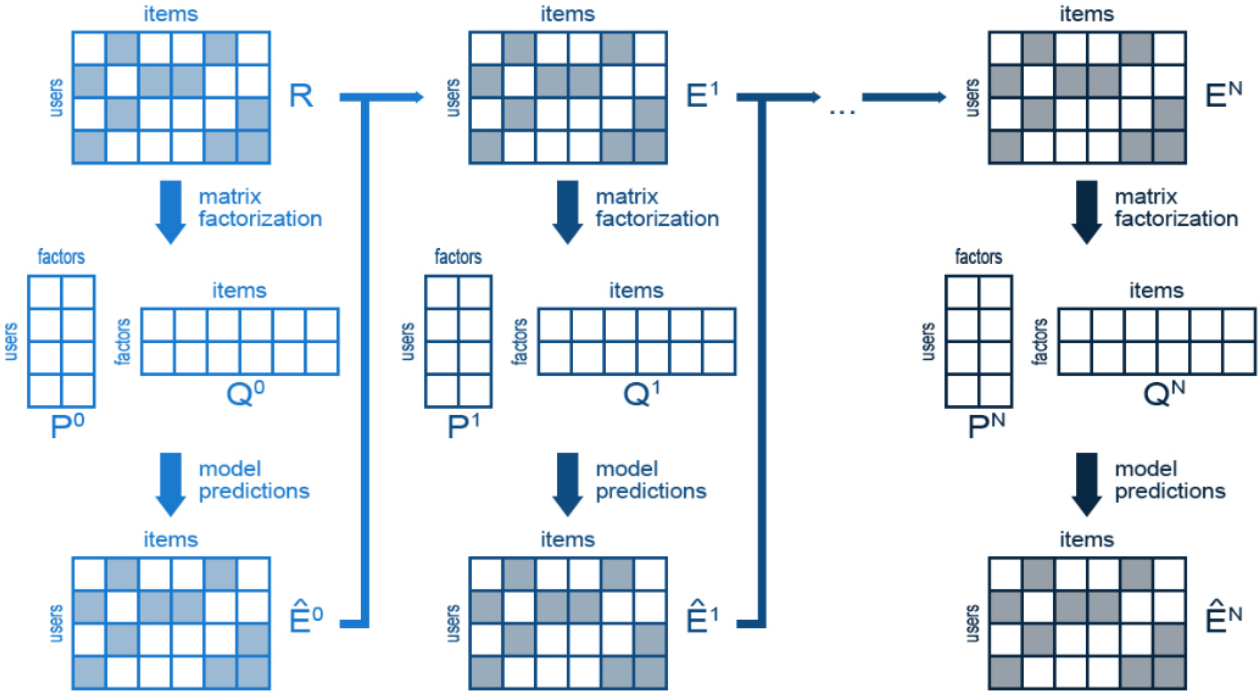


Figure 6: Illustrative explanation of proposed DeepMF method.

3.2 Method

In order to implement the *DeepMF* method, it was decided to opt for four successive factorization layers, as suggested in (2), and to use the gradient descend method described in the section 2.2 for each matrix factorization. However, the latter method was modified in order to fix the learning rate η_I and η_U, the stopping criterion `max_iteration` also being fixed from the start (and set to 1000).

For each layer of *DeepMF*, a grid-search hyperparameter optimization was put in place to best fit the model to the dataset. To deal with the important computing time this hyperparameter optimization method requires, the performance of each hyperparameter combination was computed using the following grid :

- for the ridge penalization parameters λ_I and λ_U: $0.01; 0.1; 1$
- for the latent factor k: $1; 3$

This resulted in the computation of a total of 18 combinations per factorization layer, which equals to 72 hyperparameter combinations to assess. For 1000 iterations per gradient descent, the total execution time was

roughly three hours, which explains the reduced number of hyper-parameters fitted in the grid. Moreover, the best hyper-parameters found in (2) on the *MovieLens100K* dataset helped in the choice of the latter, as the dataset dealt with is similar.

Concerning the choice of the learning rates, at the start of each factorization layer, η_U and η_I were chosen by hand, by executing several gradient descent and selecting the biggest learning rates that did not make the algorithm diverge. These were then rounded to the nearest power of ten.

Finally, the optimization metric used to calculate the performance was the RMSE, rather than the MAE (*Mean Average Error*) chosen in (2). The grid-search hyper-parameter optimization was thus performed in order to minimize the RMSE in the train split at first, and in test split at last. Those two cases are described in the section 3.3.

3.3 Results

In order to best present the results obtained from the *DeepMF* method, it is important to note that two approaches were led in parallel, as stated in the section 3.2:

- The first approach consisted of optimizing the hyperparameters via grid search by minimizing the RMSE on the train matrix, knowing the input matrix of the *DeepMF* was the same train matrix.
- The second approach consisted of optimizing the hyperparameters via grid search by minimizing the RMSE on the test matrix, knowing the input matrix of the *DeepMF* was the train matrix.

3.3.1 First approach

In this approach, the train matrix in question is the sum of two matrices, `train` and `test` named `train+test` in the following.

	η_I	η_U	λ_I	λ_U	k	RMSE
1^{st} MF `train+test`$\approx I_0 \cdot U_0^T$	$1e-4$	$1e-3$	$1e-1$	$1e-2$	1	RMSE(`train+test`,$I_0U_0^T$)=0.79
2^{nd} MF E_1 =`train+test`$-I_0 \cdot U_0^T \approx I_1U_1^T$	$1e-3$	$1e-2$	$1e-2$	$1e-2$	3	RMSE(E_1,$I_1U_1^T$)=0.64
3^{rd} MF $E_2 = E_1 - I_1 \cdot U_1^T \approx I_2U_2^T$	$1e-3$	$1e-2$	$1e-2$	$1e-2$	3	RMSE(E_2,$I_2U_2^T$)=0.56
4^{th} MF $E_3 = E_2 - I_2 \cdot U_2^T \approx I_3U_3^T$	$1e-2$	$1e-2$	$1e-2$	$1e-2$	3	RMSE(E_3,$I_3U_3^T$)=0.49

Table 1: Results of the grid search on 1000 iterations of gradient descent.

This approach, summerized in the table 1, is aimed at reproducing the results of paper (2), showing that the more layers are added in the *DeepMF*, the more the predictions are precise. This is true for a model optimized for the exact data it is supposed to predict. The results are indeed well reproduced.

It can be noted, as shown in figure 7, that the performance stays the same whatever the number of iterations in the gradient descent, showing that the model is well optimized for the prediction of the existing ratings. However, it is very bad at generalizing, as the optimization does not take into account ratings that are not in the matrix used for optimizing the model.

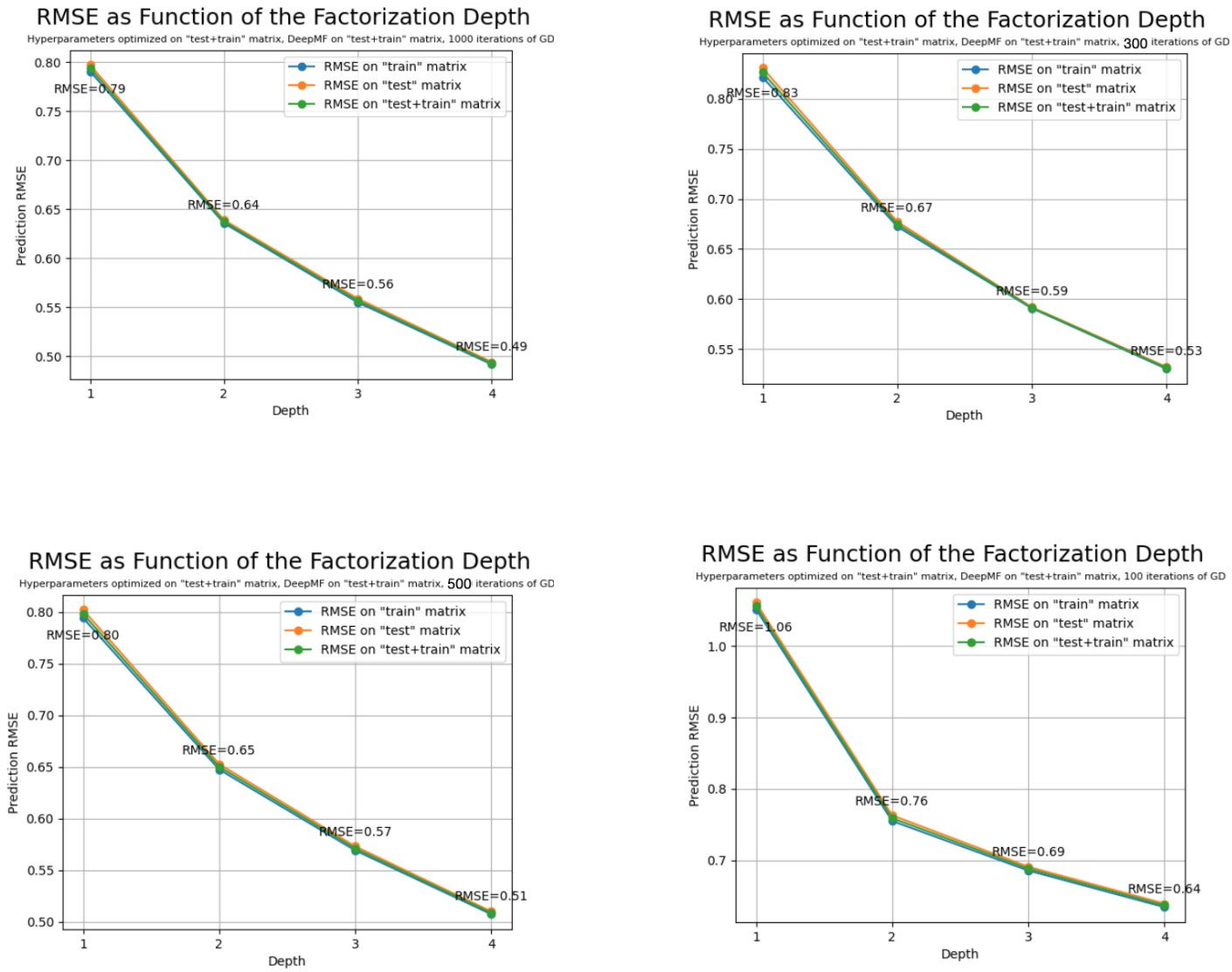


Figure 7: First approach results for DeepMF.

3.3.2 Second approach

In this approach, the generalization issue discovered in the first approach is tackled, and the grid search optimization of the hyperparameters is conducted using the `train` matrix as input of the *DeepMF*, and using the `test` matrix to minimize the RMSE of the factorization errors.

	η_I	η_U	λ_I	λ_U	k	RMSE
1^{st} MF $\texttt{train} \approx I_0 \cdot U_0^T$	$1e-4$	$1e-3$	$1e-2$	1	1	RMSE($\texttt{test}$,$I_0U_0^T$)=0.9143
2^{nd} MF $E_1 = \texttt{train} - I_0 \cdot U_0^T \approx I_1U_1^T$	$1e-3$	$1e-2$	1	1	1	RMSE($\texttt{test}-I_0U_0^T$,$I_1U_1^T$)=0.9219
3^{rd} MF $E_2 = E_1 - I_1 \cdot U_1^T \approx I_2U_2^T$	$1e-3$	$1e-2$	1	1	1	RMSE($\texttt{test}-I_0U_0^T - I_1U_1^T$,$I_2U_2^T$)=0.9457
4^{th} MF $E_3 = E_2 - I_2 \cdot U_2^T \approx I_3U_3^T$	$1e-2$	$1e-2$	1	1	1	RMSE($\texttt{test}-I_0U_0^T - I_1U_1^T - I_2U_2^T$,$I_3U_3^T$) =0.9731

Table 2: Results of the grid search on 1000 iterations of gradient descent.

As the table 2 shows, the RMSE on the test split are very close from one another, which suggests a better

generalization of the model. This result is observed on figure 8, where an increasing precision of the RMSE on `train+test` can be observed the deeper the matrix factorization layer.

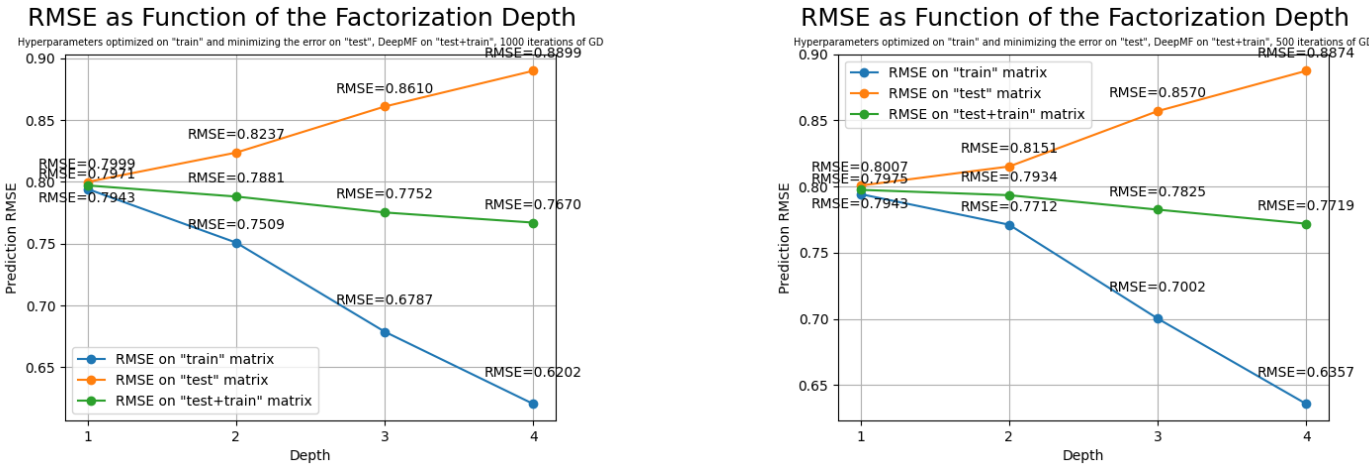


Figure 8: Second approach results for DeepMF.

The code for DeepMF using this hyperparameters setting can be found in the `generate.py` file.

4 Conclusion

Through the different methods assessed in this assignment report, it is clear that the most fruitful of all is the DeepMF factorization method. However, although the prediction precision improves itself with the number of factorization layers, it is not true that the recommendations get better, as (2) claims. The generalization issue still remains because of the choice of the matrix factorization method via gradient descent, which may not be optimal for collaborative filtering contexts. Nonetheless, it would still be interesting to look at the performance of this method on another broader datasest, to reevaluate this lack of generalization.

Finally, this work may be interestingly enhanced by looking at other - maybe better - matrix factorization methods to combine with DeepMF.

References

[1] M. G. Vozalis, A. Markos, and K. G. Margaritis, "Collaborative filtering through svd-based and hierarchical nonlinear pca," *Artificial Neural Networks – ICANN 2010*, 2010.

[2] R. Lara-Cabrera, González-Prieto, and F. Ortega, "Deep matrix factorization approach for collaborative filtering recommender systems," *Applied Sciences*, vol. 10, no. 14, 2020.