

Big Data Intelligence Assignment 2

Gausse Mael DONGMO KENFACK (董沫高斯)
Student ID: 2024403346

November 2024

Contents

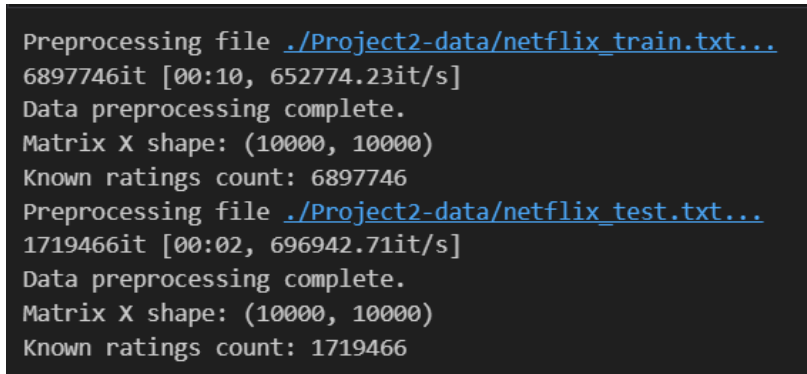
Introduction	1
1 Data Preprocessing	2
2 Collaborative Filtering	2
2.1 Implementation	2
2.2 Analysis	3
3 Matrix Decomposition	4
3.1 Implementation and Inputs	4
3.2 Results for $k = 50$ and $\lambda = 0.01$	5
3.3 Parameters Selection	5
4 Comparative Study	8
Conclusion	8
References	8

Introduction

Personalized recommendation systems have become an essential application of big data, enabling platforms to tailor their services to individual users. These systems leverage user behavior to predict preferences and recommend new items. The primary goal is to address the challenge of predicting missing values in a user-item behavior matrix. In this project, we tackle the movie recommendation task using a subset of the Netflix dataset, which consists of 10,000 users and 10,000 movies. The task involves implementing two classic recommendation algorithms: **user-based collaborative filtering** and **Adam optimizer for matrix decomposition**. By evaluating and comparing these methods using metrics such as Root Mean Square Error (RMSE), we aim to explore their strengths, limitations, and optimal configurations. This document is accompanied by a *Jupyter Notebook* file containing the complete code for the data processing and the algorithms implementations.

1 Data Preprocessing

The first step in building a recommendation system is to pre-process the raw data into a format suitable for analysis. The dataset consists of four files: user IDs, movie titles, a training set of ratings, and a test set of ratings. Since the user IDs are non-sequential, a mapping is created to assign each user a 0-based index. The training and testing datasets are then parsed to populate two sparse matrices: the rating matrix (\mathbf{X}), which stores user ratings for movies, and the indicator matrix (\mathbf{IM}), which records whether a rating is known. By leveraging sparse matrix representations [2], we efficiently handle the large but sparse nature of the dataset, reducing memory usage and preparing the data for subsequent analysis. The figure 1 is a screenshot of the result after data preprocessing. We also wrote some unit tests to ensure we preprocessed the data correctly.



```
Preprocessing file ./Project2-data/netflix_train.txt...
6897746it [00:10, 652774.23it/s]
Data preprocessing complete.
Matrix X shape: (10000, 10000)
Known ratings count: 6897746
Preprocessing file ./Project2-data/netflix_test.txt...
1719466it [00:02, 696942.71it/s]
Data preprocessing complete.
Matrix X shape: (10000, 10000)
Known ratings count: 1719466
```

Figure 1: Preprocessing Results, first the training then the test dataset

2 Collaborative Filtering

Collaborative filtering (CF) is a widely used technique for recommendation systems, where user preferences are predicted based on the preferences of other similar users. In this part, we implemented **user-based collaborative filtering** using **cosine similarity** to identify user similarities and matrix operations to predict ratings efficiently. The performance of the model is evaluated using the Root Mean Square Error (RMSE) on the test set.

2.1 Implementation

The implementation is divided in few steps:

- First we computed the similarities as a matrix \mathbf{S} .

$$sim(x, y) = \frac{x \cdot y}{\|x\| \cdot \|y\|}$$

To compute similarities for all user pairs, the rows of the training matrix are normalized, and the similarity matrix is obtained via matrix multiplication

$$norm_X[i, :] = \frac{\mathbf{X}[i, :]}{\|\mathbf{X}[i, :]\|}$$
$$\mathbf{S} = norm_X \cdot norm_X^T$$

- For each user, only the top $k_neighbors$ most similar users are considered for predictions. This parameter will be discussed in the next subsection.
- Missing ratings are predicted by averaging ratings from similar users, weighted by their similarity.

$$\hat{\mathbf{X}}[i, j] = \frac{\sum sim(\mathbf{X}[i], \mathbf{X}[k]) \cdot \mathbf{X}[k, j]}{\sum sim(\mathbf{X}[i], \mathbf{X}[k])}$$

This calculation is efficiently implemented using matrix multiplication between the similarity matrix and the training matrices (ratings and indicator matrix)

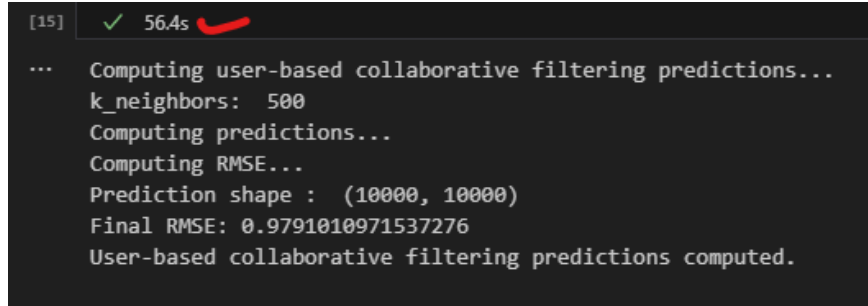
$$\begin{aligned} weighted_sum &= \mathbf{S} \cdot \mathbf{X}_{train} \\ sim_sum &= \mathbf{S} \cdot \mathbf{IM}_{train} \\ \hat{\mathbf{X}} &= \frac{weighted_sum}{sim_sum} \end{aligned}$$

- The predicted matrix is then compared with the test matrix and the RMSE is calculated for evaluations.

$$RMSE = \sqrt{\frac{1}{|\text{Test}|} \left(\sum_{(i,j) \in \text{Test}} (X_{ij} - \hat{X}_{ij})^2 \right)},$$

2.2 Analysis

- Execution Time: The first aspect to evaluate is the execution time of the algorithm. Since predictions are computed through matrix multiplication, the execution time is directly influenced by the size of the matrices rather than the number of neighbors $k_neighbors$. This is because the matrix dimensions remain constant regardless of the $k_neighbors$ value. As shown in Figure 2, the execution time on my PC for $k_neighbors = 500$ was approximately 56 seconds, which is significantly faster than using nested loops. This efficiency highlights the advantage of leveraging matrix operations over iterative approaches.



```
[15] ✓ 56.4s
... Computing user-based collaborative filtering predictions...
k_neighbors: 500
Computing predictions...
Computing RMSE...
Prediction shape : (10000, 10000)
Final RMSE: 0.9791010971537276
User-based collaborative filtering predictions computed.
```

Figure 2: Output of the notebook for a prediction with 500 neighbors

- RMSE Evolution with $k_neighbors$: The second point of analysis focuses on how the RMSE evolves as $k_neighbors$ changes, as illustrated in Figure 3. The RMSE measures the average deviation between the predicted and actual values in the test set. Lower RMSE values indicate

higher prediction accuracy. For a rating scale ranging from 1 to 5, an RMSE lower than 1 is considered reasonably good, although there is still room for improvement. The results suggest that our collaborative filtering implementation captures meaningful patterns but can be further optimized for better accuracy.

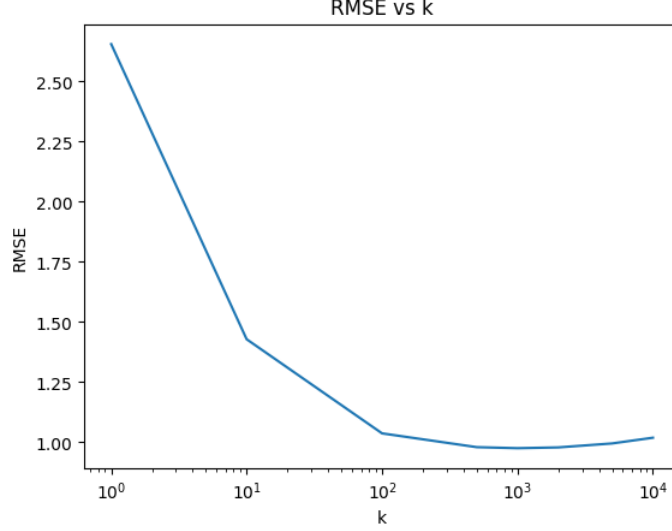


Figure 3: Evolution of the RMSE with $k_neighbors$ (the x axis is log scaled)

3 Matrix Decomposition

In this section we try another approach to solve the recommendation problem. We are going to decompose our rating matrix \mathbf{X} into 2 matrices \mathbf{U} and \mathbf{V} representing the user and movie in the latent space. To do that, we choose to implement an optimization algorithm to minimize the target function \mathbf{J} defined as:

$$\mathbf{J} = \frac{1}{2} \|\mathbf{IM} \odot (\mathbf{X} - \mathbf{UV}^\top)\|_F^2 + \lambda \|\mathbf{U}\|_F^2 + \lambda \|\mathbf{V}\|_F^2$$

To implement this optimization, since the dataset is very large, we decided to code a ***ADAM Optimizer*** [1]. It is an optimization algorithm based on gradient descent and involving Momentum and Root Mean Square Propagation.

3.1 Implementation and Inputs

To implement the Adam optimization algorithm for matrix decomposition, we carefully configured several parameters to ensure effective training and convergence. The primary input to the algorithm is the user-item interaction matrix (\mathbf{X}_{train}), which we aim to decompose into two lower-dimensional matrices. A binary indicator matrix (\mathbf{IM}_{train}) is also used to distinguish observed entries from missing values during training. The number of latent factors, k , controls the complexity of the decomposition, allowing the model to capture intricate patterns in the data. Key hyper-parameters, such as the learning rate (α) and Adam’s decay rates for the first and second

moment estimates (β_1 and β_2), influence the stability and speed of optimization. Regularization (λ) is incorporated to prevent overfitting by penalizing large values in the latent matrices. Convergence is monitored through a threshold on the change in the objective function, ensuring computational efficiency while maintaining accuracy. Overall, these parameters enable the Adam-based matrix decomposition to adaptively balance model performance and efficiency.

3.2 Results for $k = 50$ and $\lambda = 0.01$

We ran the Adam optimizer with $k = 50$ and $\lambda = 0.01$ to begin and the results are presented by Figure 4. This ran for 1000 iterations and produces a final RMSE of 0.902. It important to note that after 250 iterations, the RMSE was already 0.94 which is better than the Collaborative filtering or random guess, but it is perfectible. In the next part we are going to study the parameters so see if we can reduce the RMSE and the number of iterations.

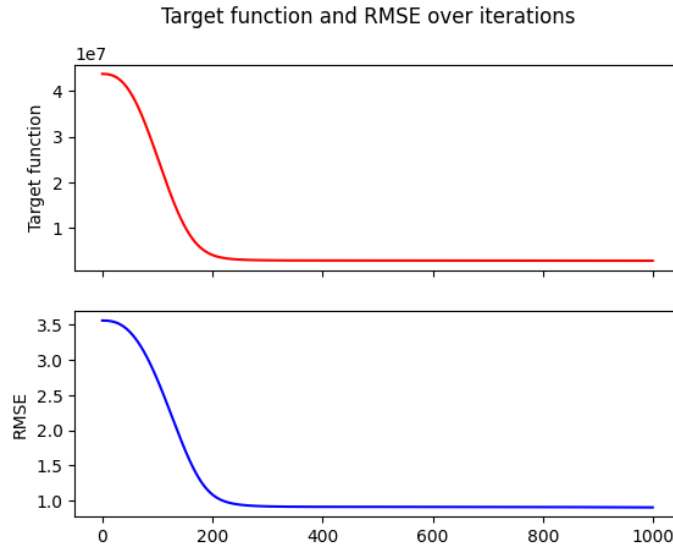


Figure 4: Evolution of the RMSE and the target function with the iterations ($k = 50$ and $\lambda = 0.01$)

3.3 Parameters Selection

We focus the studies of this part on 3 elements : the way we initialize \mathbf{U} and \mathbf{V} , the number of latent factors k and the regularization λ

1. Initialization Types

We implemented 4 different functions for matrix initialization, random values from a uniform distribution, small random values from a normal distribution, a matrix of ones and a sparse matrix with random values in $[1-5]$ at random positions. By looking at the Figure 5 we can see the impact of the Initialization Types on the RMSE after 100 iterations only. Among the tested the normal and sparse initialization yield the best RMSE values, so they should converge after fewer iterations than the others. In contrast, initializing with all ones will take

much more iterations to converge. This can be explained by the fact the the ratings matrix is very sparse, so it's the convergence is quicker when starting with values close to 0.

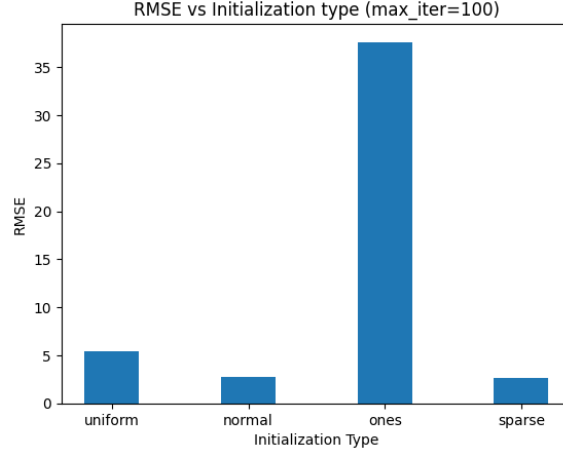


Figure 5: RMSE after 100 iterations for the different functions for matrix initialization ($k=50$)

2. parameter k

Next, we analyzed the RMSE after 100 iterations for different values of k , the latent feature dimension. You can see the plot on Figure 6. The results reveal that larger values of k facilitate faster convergence, as the model captures more latent relationships in the data. However, the improvements diminish after $k = 500$, where the RMSE is equal to 0.919 and begins to plateau. Notably, the model achieves an RMSE of 0.896 after 100 iterations with $k = 2000$.

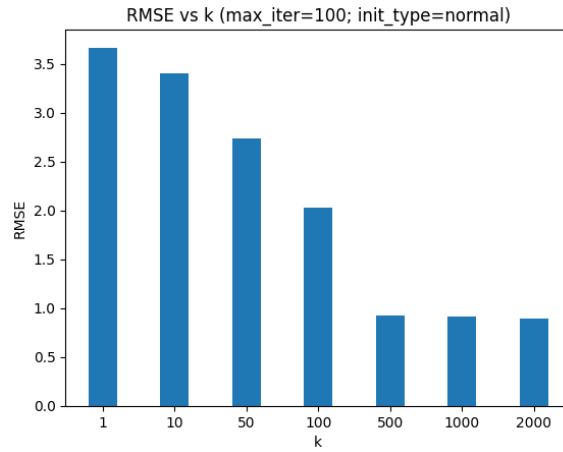


Figure 6: RMSE after 100 iterations for different values of k

3. parameter λ

The third plot was to check if with the other parameters fixed, the regularization would have a big impact on the RMSE; as the Figure 7 shows, it doesn't.

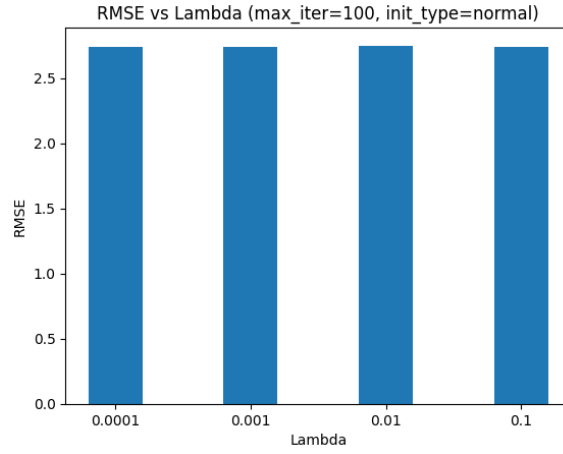


Figure 7: RMSE after 100 iterations for different values of λ , ($k=50$)

4. Best parameters

The Figure 8 present a run with $k = 500$, $\lambda = 0.01$, $init_type = normal$, $max_iter = 500$. The final RMSE is 0.864.

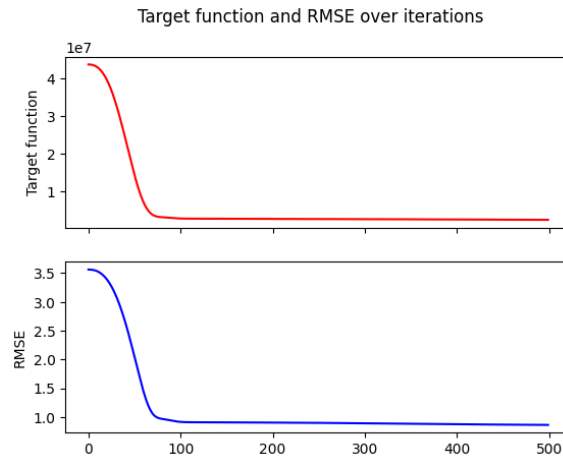


Figure 8: Evolution of the RMSE and the target function with the iterations

4 Comparative Study

In the previous two sections, we explored and implemented two distinct recommendation systems. Now, we shift our focus to a comparative study, evaluating their performances side by side. The performance and characteristics of the two recommendation approaches are summarized in Table 1 and Illustrated by Figure 9.

Criteria	Collaborative Filtering	Matrix Decomposition
Final RMSE	0.979	0.864
Execution Time	~ 1 minute	~ 30 minutes
Strengths	Better than a random guess, Simple to implement, Interpretable predictions	Better than a random guess, Captures latent relationships, Scalable for large datasets
Weaknesses	Computational cost increases with large datasets	Requires careful parameter tuning, slower convergence

Table 1: Comparison of Recommendation System Methods

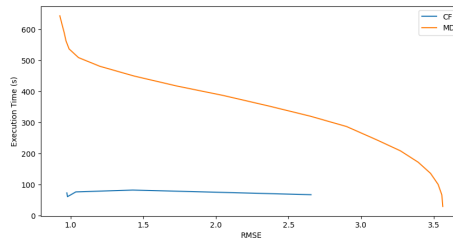


Figure 9: Execution time in seconds with the RMSE for the two algorithms

Conclusion

In conclusion, we explored the mechanics of recommendation systems by implementing and analyzing two distinct algorithms. Managing the challenges posed by the $10,000 \times 10,000$ matrix was a significant achievement, deepening our understanding of optimization and computational efficiency in recommendation systems.

References

- [1] *Adam Optimizer*. <https://www.geeksforgeeks.org/adam-optimizer/>. [Online].
- [2] *Scipy Sparse Matrix*. <https://docs.scipy.org/doc/scipy/reference/sparse.html>. [Online].