Final Report: Matrix Imputation Based on Restaurant Survey

Chunru Yu, Himanshu Kumar, Lexuan Ye, Ran Jiang
Xin Fan, Xutao Cao, Yufei Ruan, Zhengyu Zheng

May 2023

1 Introduction

Matrix imputation is a critical problem in data science and machine learning that aims to fill in missing or incomplete entries within datasets, a common issue in real-world scenarios such as user surveys, sensor readings, or online platforms.

In our project, we investigated the effectiveness of various imputation techniques in addressing missing data in a restaurant survey completed by our classmates. Using the MovieLens dataset as a benchmark, we tested a wide range of algorithms, including those from the Python surprise package and the R recommenderlab package, as well as deep learning approaches and traditional soft/hard impute approaches. By evaluating the performance of each algorithm based on root mean square error (RMSE), we aimed to provide valuable insights into the most effective imputation techniques for handling missing data in restaurant surveys and similar contexts.

1.1 Task

In this project, our task is to impute missing data from a restaurant survey involving 41 classmates rating 15 restaurants, selected from the list of *15 great Campustown places to eat* featured on ChambanaMoms. With half of the entries intentionally removed, our objective is to accurately impute the missing values while minimizing the root mean square error (RMSE) in the process. An excerpt of the matrix that needs to be imputed is as follows:

C \ R	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15
1	3	NA	NA	3	NA	3	3	NA	NA	3	3	NA	4	NA	3
2	NA	3	3	NA	NA	NA	3	3	NA	NA	3	NA	5	NA	3
3	NA	NA	3	3	3	3	3	3	NA	NA	NA	4	3	3	NA
4	4	4	NA	NA	3	NA	NA	3	NA	2	NA	4	NA	4	NA
5	5	NA	5	5	NA	NA	5	NA	5	NA	NA	5	NA	NA	5
6	1	NA	4	2	NA	NA	4	NA	4	2	NA	3	NA	NA	NA

Table 1: Excerpt of task matrix. C\R stands for Classmates\Restaurants.

1.2 Dataset

To evaluate the performance of the algorithms, we used the MovieLens 100K dataset [1], a well-known benchmark dataset for collaborative filtering and recommendation systems. We will select the most effective algorithms based on their performance on the test dataset and apply them to our restaurant survey imputation task. The MovieLens 100K dataset consists of a sparse matrix containing 943 users, 1682 movies, and 100,000 ratings (ranging from 1 to 5).

	1	2	3	4	5	6	7	8	9	10
1	5	3	4	3	3	5	4	1	5	3
2	4									2
5	4	3								
6	4						2	4	4	
7				5			5	5	5	4
8							3			
9						5	4			
10	4			4			4		4	

(a) Excerpt of MovieLens 100K

	1	2	3	4	5	6	7	8	9	10
1	5	3	4	3	3	5	4	1	5	3
2	4									2
5	4	3								
6	4						2	4	4	
7				5			5	5	5	4
8							3			
9						5	4			
10	4			4			4		4	

(b) Randomly mask half the ratings and predict them

Figure 1: Workflow Illustration: calculate RMSE by original masked ratings and their predicted values.

As shown in Figure 1, to simulate the missing data scenario in our restaurant survey, we randomly masked half of the ratings in the MovieLens dataset. We then utilized different algorithms to predict the masked ratings and calculated the root mean square error (RMSE) between the original masked ratings and their predicted values.

In addition to the MovieLens 100K dataset, we also employed the MovieLens 1M dataset to further validate the performance of our selected algorithms. The MovieLens 1M dataset is an extended version of the original 100K dataset, containing 1 million ratings from 6,040 users across 3,952 movies.

1.3 Data Manipulation

Nevertheless, one difficulty we are facing is that the MovieLens dataset may not serve as an optimal reference for our task. In MovieLens, users only rate the movies they have watched, and some users rate as few as 20 to 30 movies out of the entire collection of 1,600+ movies. In contrast, our task involves many restaurants that are unfamiliar to our classmates. We conducted a survey to understand how our classmates would rate unknown restaurants and discovered that the most common approach was to assign a rating of 3 or a value below their average rating.

user \ movie	1	4	7	8	9	11	12	13	14	15	22	23	24	25	28	31	50	56
1	5	3	4	1	5	2	5	5	5	5	4	4	3	4	4	3	5	4
5	4	3	3	3	3	3	3	3	3	3	3	3	4	3	3	3	4	3
6	4	3	2	4	4	3	4	2	5	3	3	4	3	3	2	3	4	4
7	3	5	5	5	5	3	5	3	3	3	5	3	3	3	5	4	5	5
10	4	4	4	3	4	4	5	3	3	3	5	5	3	3	3	3	5	5
11	3	3	3	4	5	2	2	3	3	5	4	3	3	3	5	3	3	4
13	3	5	2	4	3	1	5	5	4	3	4	5	1	1	5	3	5	5
18	5	3	3	5	5	3	5	5	5	4	5	4	3	3	3	3	4	5

Table 2: Example of Manipulated Matrix Excerpt: fill sparsity with 3

To address this discrepancy, we will preprocess the MovieLens dataset to create a noisier version that better mimics our restaurant survey. We extracted a 200×200 submatrix from the MovieLens 100K dataset with a 50% sparsity, which we believe reflects the proportion of noisy ratings based on random guesses. Next, we filled the sparse "NA" entries with either a 3 or the row mean minus a constant c, where $c = 0.1, 0.2, 0.3, 0.5, 0.75, 1$. An example of filling sparsity with 3 is shown in Table 2 and all other manipulated matrices can be found in Github. After creating the manipulated complete matrices, we test all algorithms on these matrices to check their performance under more realistic conditions.

2 Result Overview on MovieLens Dataset

2.1 Algorithms

To evaluate the performance of various algorithms on MovieLens 100K, MovieLens 1M, and the manipulated MovieLens dataset detailed in Section 1.3, we will conduct simulations on these datasets with the following bunch of algorithms:

- Classical imputation algorithms: **Soft impute, Hard impute.**
- Algorithms from Python surprise package: **SVD, SVD++, NMF, KNN Baseline.**
- Algorithms from R recommenderlab package: **UBCF, IBCF, SVD, SVDF, ALS, LIBMF.**
- Deep learning algorithm: **Neural network-based Collaborative Filtering (NCF).**

Soft impute and hard impute are matrix completion algorithms that fill in missing data by employing low-rank approximations, with soft impute employing an iterative, shrinkage-based approach, while hard impute utilizes a simple thresholding mechanism. The Python Surprise package [2] is a user-friendly, open-source library tailored for developing and analyzing recommendation systems with diverse collaborative filtering algorithms. The R recommenderlab package [3] is a comprehensive framework for creating and testing recommendation algorithms, offering a rich collection of tools for data preprocessing and evaluation. Neural network-based Collaborative Filtering (NCF) [4] is a deep learning approach that combines the strengths of both matrix factorization and neighborhood-based collaborative filtering methods to enhance recommendation performance.

2.2 Simulation Result

As mentioned in Section 1.2, we will evaluate the previously described algorithms on the MovieLens 100K and MovieLens 1M datasets. The results are presented in Figure 2. We can see that SVD/SVD++ from Python surprise package and Deep learning algorithm NCF perform the best. And generally Python surprise algorithms and NCF perform better than soft/hard impute and R recommenderlab algorithms.

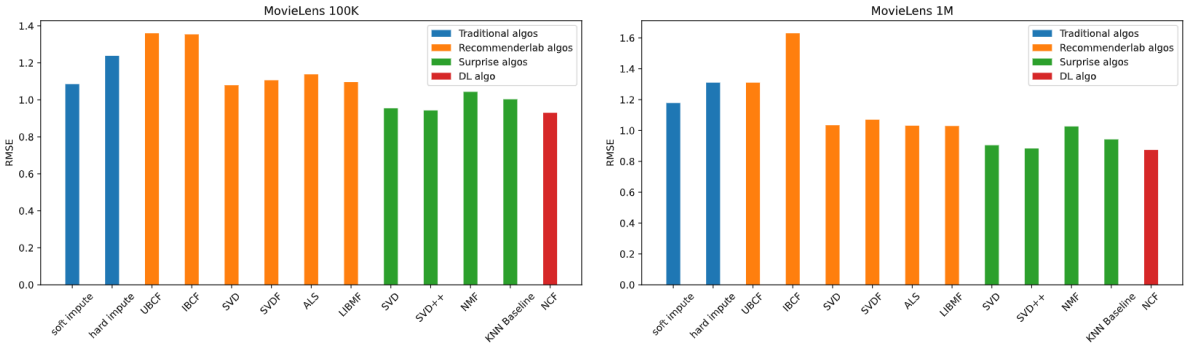

Figure 2: Simulation Result on MovieLens 100K and MovieLens 1M.

As mentioned in Section 1.3, we will also evaluate the algorithms on the 200 × 200 manipulated MovieLens datasets. The results are presented in Figure 3. We can see that SVD/**SVD++** [1] from Python surprise package and Deep learning algorithm **NCF** and **soft impute** [2] perform the best under manipulated MovieLens datasets. As a result, these 3 algorithms in bold will be utilized as a reference for our final restaurant survey imputation task. Details will be discussed in Section 4.

The detailed simulation result in float format can be found in the appendix of our group project's final presentation slides here.

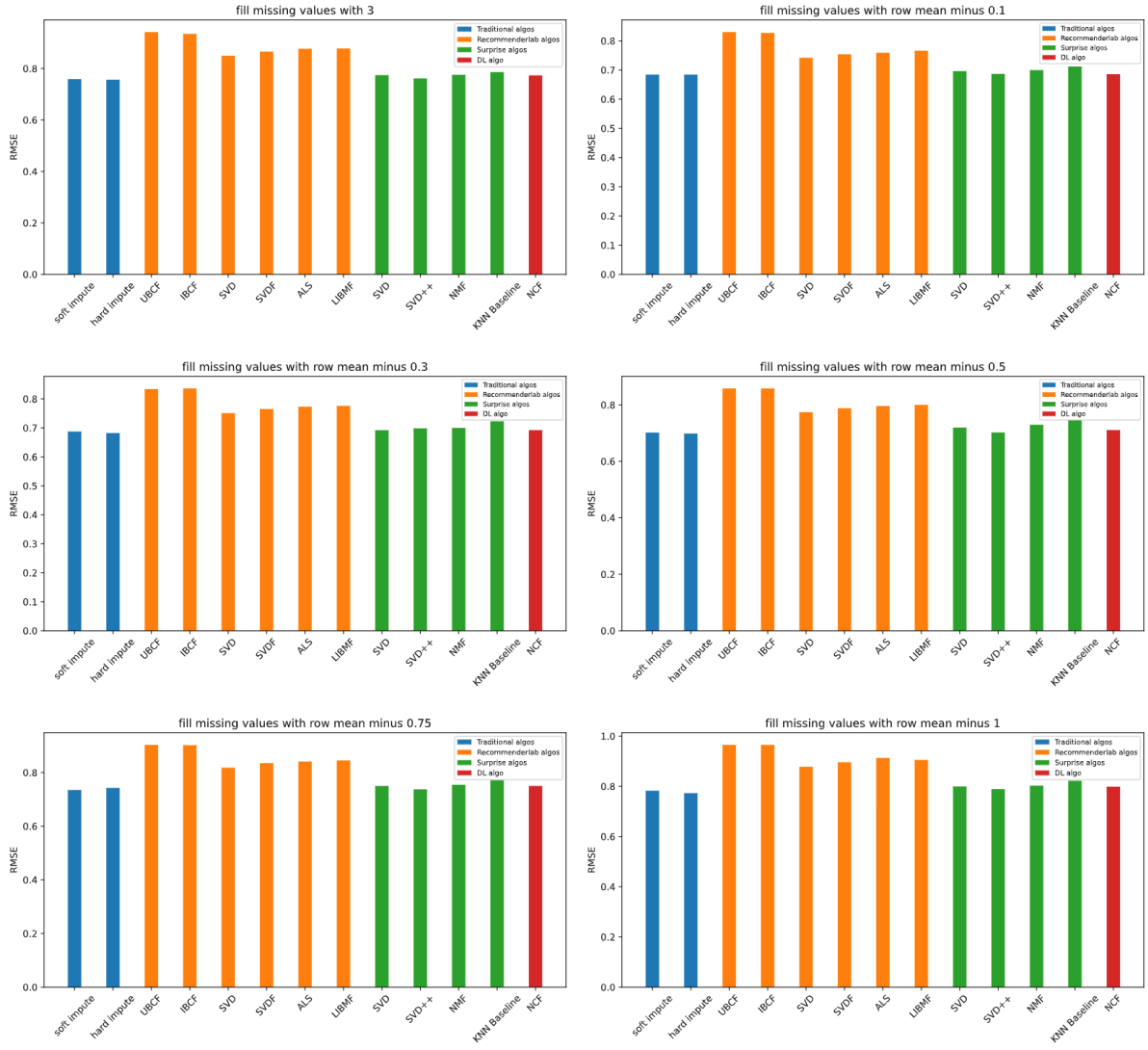

Figure 3: Simulation Result on 200 × 200 manipulated MovieLens datasets.

[1] Notice that SVD from Python surprise package and SVD from R recommenderlab package are not the same algorithms. SVD from Python surprise package is the well known Simon Funk SVD algorithm [5]. While SVD from R recommenderlab package is more like the hard impute algorithm we introduced in class. We will illustrate the difference of these two in the following Algorithm Description Section 3.

[2] The significant improvement in the performance of soft/hard impute on the manipulated dataset may be attributed to the inherent characteristics of these algorithms, which aim to minimize the matrix rank. Our data manipulation approach (filling missing values with 3 and the row mean minus a constant) also has the effect of reducing the dataset's rank.

3 Selected Algorithms Description

In this section, we will provide a detailed introduction to some selected algorithms that demonstrate strong performance on the test dataset, along with SVD from R as a reference.

3.1 Soft Impute

3.1.1 Algorithm Description

Soft Imputed-based matrix completion is on the basis of the concept of nuclear norm minimization, which promotes low-rank solutions while maintaining the original structure of the matrix [6]. This technique is particularly useful in situations where the underlying data is assumed to have a low-rank structure, such as in collaborative filtering, image inpainting, and various other applications in machine learning and data analysis.

Soft Impute has several advantages over other matrix completion methods. It is robust, scalable, and has strong theoretical guarantees for recovering the true underlying low-rank structure under certain conditions. Furthermore, it can be easily adapted to handle various types of noise and constraints on the data, making it a versatile choice for a wide range of real-world applications.

This algorithm basically contains the following steps:

1. Initialize missing entries with a guess (e.g., column means or zeros).
2. Compute the Singular Value Decomposition (SVD) of the matrix.
3. Apply soft-thresholding to singular values, setting small ones to zero.
4. Reconstruct the matrix using remaining singular values and vectors.
5. Update known entries with their original values.
6. Repeat step 2-5 until convergence.

3.1.2 Algorithm Implementation

In the `fancyimpute` package, the Funk SoftImpute algorithm is implemented as `SoftImpute` class. Our code is attached below:

```
from fancyimpute import SoftImpute
imputer = SoftImpute(max_iters=100, convergence_threshold=0.001)
```

Here `max_iters` represents the max number of iteration times, `convergence_threshold` denotes the upper limit of the change in the optimization objective function, beyond which the model should stop iterating. We set `max_iters=100, convergence_threshold=0.001` to obtain desirable results within a reasonable time. Also, we initialized the missing entries by default setting, zeros.

3.1.3 Results

Utilizing Soft Impute, the RMSE is 1.085 for MovieLens-100K dataset and 1.178 for MovieLens-1M dataset.

3.2 SVD from R Recommenderlab Package

3.2.1 Algorithm Description

SVD from R Recommenderlab Package is in fact the hard impute algorithm we introduced in class. In recommenderlab, the rating matrix R is represented as a sparse matrix using the `realRatingMatrix` class. The `SVD()` function in recommenderlab uses matrix factorization to decompose the rating matrix

into three matrices, U (users $\times$ latent factors), Σ (latent factors $\times$ latent factors), and V^T (latent factors $\times$ items). Then it selects a number of top latent factors (k) to keep, which are responsible for most of the variation in the data. This step reduces the dimensions of the U, Σ, and V^T matrices by keeping only the k largest singular values (diagonal elements of Σ) and their corresponding columns in U and V^T. Compute the reconstructed rating matrix by multiplying the reduced U, Σ, and V^T matrices: $\hat{R} = U\Sigma_k V^T$. This reconstructed matrix $\hat{R}$ is an approximation of the original matrix R with reduced dimensions.

3.2.2 Algorithm Implementation

We create an evaluation scheme using the `evaluationScheme()` function, specifying the method parameter as `split` to create a random split of the rating matrix into training and test sets. We also specify the `train` parameter as 0.5 to allocate 50% of the data to the training set, and the `given` parameter as 5 to ensure that at least 5 ratings are available for each one in the test set. We then use the `evaluate()` function to fit a recommendation model using the SVD algorithm and evaluate its performance.

```
library("recommenderlab")
data <- read.csv("path/ml-1m.dat", sep = ' ', header=FALSE)
ratings <- as(data, "realRatingMatrix")
scheme <- evaluationScheme(ratings, method = "split", train = 0.5, given = 5)
results <- evaluate(scheme, method="SVD", type = "ratings")
```

3.2.3 Results

Utilizing SVD in R recommenerlab, the RMSE is 1.079 for MovieLens-100K dataset and 1.034 for MovieLens-1M dataset.

3.3 SVD/SVD++ from Python Surprise Package

3.3.1 Algorithm Description

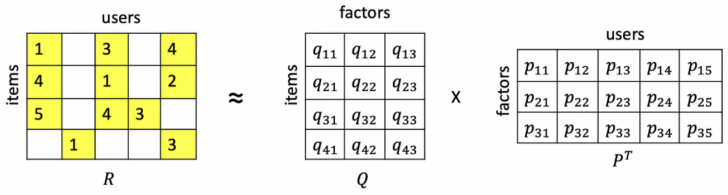

Figure 4: SVD from Python Surprise Package.

SVD from Python Surprise Package uses the Funk SVD algorithm by Simon Funk during the Netflix Prize competition [5], which aims to predict missing ratings in a user-item interaction matrix by minimizing a regularized mean squared error loss function. Details illustrated as follows:

- Matrix factorization: The user-item matrix is factorized into two matrices, Q and P, such that the original matrix R can be approximated as the product of Q and $P^\top$.

  Each row of Q measures the extent to which the item possesses those factors, positive or negative, and each row of P measures the extent of interest the user has in items that are high on the corresponding factors (again, these may be positive or negative).

- Optimization: The latent features in Q and P are optimized by minimizing the sum of squared errors between the predicted and actual ratings. This is typically done using gradient descent, which involves iteratively updating the values of Q and P to minimize the error:

- Minimize the objective function $J(Q, P) = \|R - QP^\top\|_F^2$, where $\|\cdot\|_F$ denotes the Frobenius norm, which is the square root of the sum of squares of all the elements of the matrix.
- Use gradient descent to minimize the objective function $J(Q, P)$. The updated equations are as follows:

$$Q = Q + \text{lr} \cdot (R - QP^\top)P^\top, \qquad P = P + \text{lr} \cdot Q^\top(R - QP^\top),$$

where lr is the learning rate, which controls the step size of each update.

These equations update the values of Q and P iteratively until the objective function converges to a minimum. At each iteration, the predicted ratings are recomputed as $\hat{R} = QP^\top$, and the error is evaluated using the objective function $J(Q, P)$.

- Prediction: Once the optimization is complete, the predicted ratings can be computed by taking the dot product of the corresponding user and item latent feature vectors in Q and P, respectively.
- Regularization: To prevent overfitting, regularization is often applied to the optimization process. This involves adding a penalty term to the error function that encourages the latent feature vectors to be as small as possible. The strength of the regularization can be controlled using a hyperparameter.

SVD++ is an extension of SVD that takes into account implicit feedback in addition to explicit ratings. Implicit feedback refers to user-item interactions such as clicks, views, and purchases, that can be used to infer a user's preferences even if they did not explicitly rate the item. SVD++ incorporates this information by adding a user and an item bias term, which are used to estimate the user's preference for an item even if the user has not rated it.

3.3.2 Implementation

In the Surprise package, the Funk SVD algorithm is implemented as `SVD` class. To use it, we imported the package and create an instance of the `SVD` class with the default parameters:

```
from surprise import SVD
algo = SVD(n_factors, n_epochs, lr_all, reg_all)
algo.fit(trainset)
# for SVD++ algorithm
# algo = SVDpp(cache_ratings=False)
```

Here `n_factors` represents the number of latent factors, `n_epochs` denotes the number of iterations for stochastic gradient descent, `lr_all` specifies the learning rate, and `reg_all` sets the regularization term. We kept all the parameters as default settings.

3.3.3 Result

Utilizing SVD python package, the RMSE is 0.954 for MovieLens-100K dataset and 0.904 for MovieLens-1M dataset.

For SVD++ Algorithm, We observed that the RMSE value for MovieLens-100K is 0.943 and for MovieLens-1M is 0.883, which is slightly better than SVD.

3.4 Deep Learning Algo: Neural Collaborative Filtering (NCF)

3.4.1 Algorithm Description

Deep matrix factorization is an advanced approach that leverages the power of deep learning and traditional matrix factorization for recommendation systems. Traditional matrix factorization simply decompose the user-item interaction matrix into lower-dimensional matrices to reveal the underlying latent factors that govern user preferences and item characteristics. Deep matrix factorization addresses

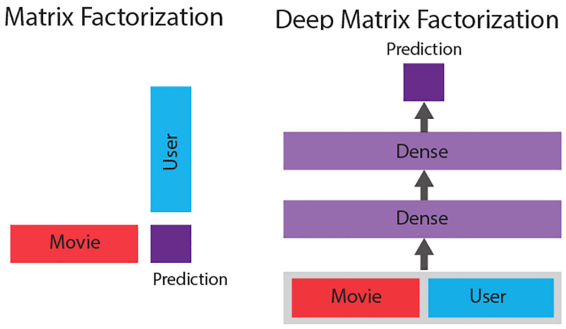

Figure 5: Traditional Matrix Factorization vs Deep Matrix Factorization, source from O'Reilly [7].

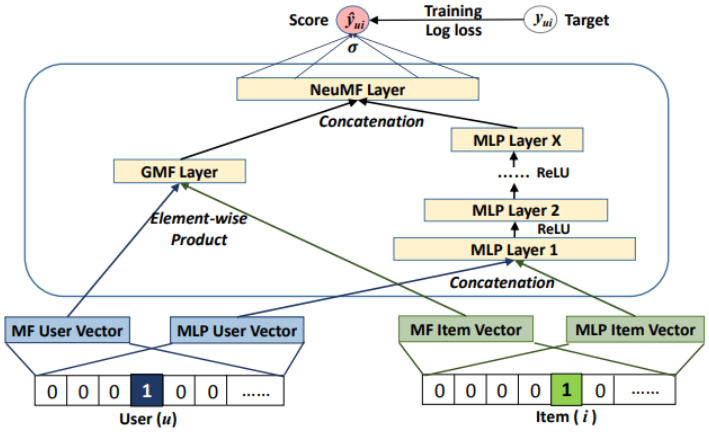

Figure 3: Neural matrix factorization model

Figure 6: Neural Collaborative Filtering Network, source from [4].

its limitation in capturing complex and nonlinear relationships by incorporating the neural network structure, which allows for the learning of more intricate patterns in user-item interactions.

In this project, we adopt the Neural Collaborative Filtering (NCF) framework introduced by He et al in 2017 for deep matrix factorization [4]. The NCF model consists of three key components: embeddings, multilayer perceptron (MLP) and generalized matrix factorizaion (GMF), and the general workflow of the NCF is described below:

1. The embeddings representing the latent factors for users and items are learned and concatenated, which will be used as the input of both the MLP and GMF.

2. The GMF component focuses on learning latent factors for users and items and capturing linear relationships between them, which is similar to traditional matrix factorization.

3. On the other hand, the MLP component, which consists of multiple fully connected layers with activation functions, is used for capturing complex and non-linear interactions between user and item features.

4. Finally, the outputs from the GMF and MLP are concatenated and go through a final NeuMF layer, which produces the final rating or recommendation.

3.4.2 Algorithm Implementation

In this project, we use the `pytorch` library to write the NCF model from scratch, with the full code provided on Github. To select the best hyperparameters, we perform 3-fold cross-validation for the MovieLens relevant datasets, and 5-fold cross-validation for the restaurant dataset. For the restaurant dataset, specifically, the part of the matrix with ratings is used as the training-validation dataset. The hyperparameters optimized include the embedding size, MLP layer size, batch size and learning rate, with 200 training epochs. According to the 5-fold cross validation result for the restaurant dataset, the best hyperparamter combination is embedding size=64, MLP layer size=[128, 64, 32], batch size=100, learning rate=0.001.

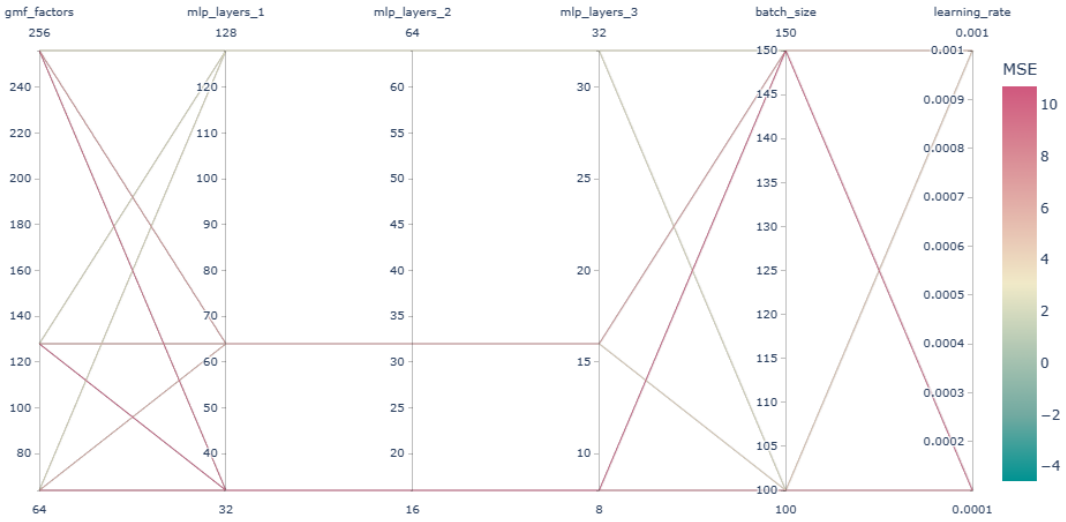

Figure 7: Restaurant dataset hyperparameter selection

3.4.3 Results and Discussions

Utilizing the NCF model written from scratch, the RMSE is 0.930 for MovieLens-100K dataset and 0.874 for MovieLens-1M dataset. In general, the NCF model performs better on larger datasets, which provides enough training data to ensure its performance. Also, it is better at capturing non-linear relationships between users and items, and can be more easily extended to incorporate additional features (e.g. item genres or user demographics, which are not included in this project), which may further improve the recommendation accuracy compared with traditional matrix factorization methods. Meanwhile, Deep Matrix Factorization may also have its drawbacks, including higher computational resource requirements, higher sensitivity to hyperparameter selections, as well as the tendency to overfitting when the training data is limited or the model complexity is high.

4 Final Imputation Result

To solve the final imputation task, we applied three top-performing algorithms: **Soft Impute**, **SVD++**, and **NCF**. When rounding these algorithms' predictions to the nearest integer (which can be interpreted as voting for the rounded integer), the variation between the three methods was found to be minimal. This is demonstrated in Figure 8, which displays a standard deviation heatmap of the integer outputs from the three algorithms. As illustrated in Figure 8, the majority of the predictions either match

across all three algorithms, which results in standard deviation 0; or match across two of them, with the other one's deviation being only a single unit, which results in a standard deviation of 0.47.

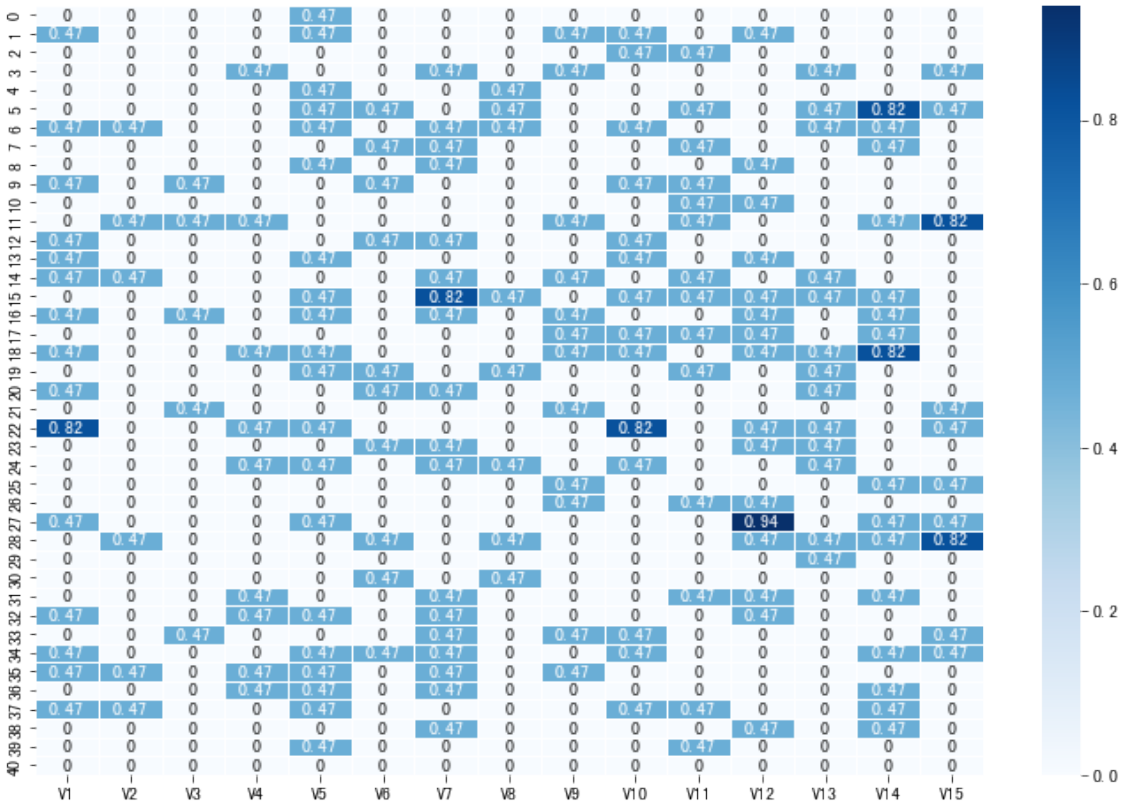

Figure 8: Standard deviation heatmap of the integer outputs from 3 best performing algorithms.

As a result, we finalize the imputation task by an ensemble method with the following steps:

1. Compute the imputed matrix by top-performing algorithms: **Soft Impute**, **SVD++**, and **NCF**.
2. Round the float ratings into integer values.
3. For each missing entry, choose the most common integer rating from the three methods. If there's no consensus, use the median of the three integer ratings.

The excerpt of the final imputed restaurant survey matrix is presented below, with blue values being our voted result of the 3 algorithms. The comprehensive final result is attached as final_result.csv.

C \ R	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15
1	3	3	3	3	3	3	3	3	3	3	3	3	4	3	3
2	4	3	3	3	4	3	3	3	3	3	3	3	5	3	3
3	3	3	3	3	3	3	3	3	3	2	3	4	3	3	3
4	4	4	3	4	3	3	3	3	3	2	3	4	4	4	3
5	5	4	5	5	4	4	5	4	5	4	4	5	5	4	5
6	1	3	4	2	2	2	4	2	4	2	3	3	3	2	2

Table 3: Excerpt of imputed restaurant survey matrix. C\R stands for Classmates\Restaurants.

5 Conclusion

In this project, we faced the difficulty that the MovieLens dataset is a sparse matrix, whereas our restaurant survey is a complete matrix containing numerous ratings for unfamiliar restaurants. To address this issue, we used a manipulated MovieLens dataset to mimic our restaurant survey.

In our simulation, we discovered that Soft Impute, SVD++, and NCF performed the best on these simulated dataset. Consequently, we applied these three algorithms to predict the missing values in the restaurant survey, rounded the results to the nearest integer, and took the median integer from these three algorithms as our final result.

Chunru Yu, Xutao Cao are responsible for exploring Soft/Hard Impute algorithms. Xin Fan, Yufei Ruan, Zhengyu Zheng are responsible for exploring algorithms in R Recommenderlab package. Himanshu Kumar, Ran Jiang are responsible for exploring algorithms in Python Surprise package. Lexuan Ye is responsible for exploring the deep learning NCF algorithm.

All code and manipulated matrices can be found in Github. Presentation slides can be found here.

References

[1] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 2015.

[2] Nicolas Hug. Surprise: A python library for recommender systems. *Journal of Open Source Software*, 5(52):2174, 2020.

[3] Michael Hahsler. *recommenderlab: A Framework for Developing and Testing Recommendation Algorithms*, 2020. R package version 0.2-6.

[4] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, pages 173–182. International World Wide Web Conferences Steering Committee, 2017.

[5] Simon Funk. Netflix update: Try this at home, 2006.

[6] Rahul Mazumder, Trevor Hastie, and Robert Tibshirani. Spectral regularization algorithms for learning large incomplete matrices. *J. Mach. Learn. Res.*, 11:2287–2322, 2010.

[7] O'Reilly Media. Deep matrix factorization using apache mxnet, 2023. Accessed: May 10, 2023.