Advances in Data Mining: Fall 2022 – Assignment 1

Chao Zhao[1], Qingzhi Li[1], and Weiji Wang[1]

[1]Leiden Institute of Advanced Computer Science, Leiden University
{*s3645002,s3050947,s3644677*}*@vuw.leidenuniv.nl*

Last updated on: October 25, 2022

1 Introduction

Netflix is an American subscription streaming service and production company with 222 million subscribers worldwide with thousands of movies by September 2022 [1]. And each user might not watch every movie and it would be important to recommend proper movies or series to different users based on available data. So how to select movies and series that users prefer would be challenging.

In this assignment, we use three methods Naive Approach, UV Matrix Decomposition and Matrix Factorization to recommend different context to different users. If we know that user A and user B have the same preferences, we recommend a movie that user B has seen that user A has not seen to user A will boost user A's click-through rate, thus generating revenue for the commercial company.

For this project, we introduce the data and data source in Section 2. We describe our methods for prediction in Section 3. Then we describe the results and visualize the matrix by dimensionality reduction in Section 4 and 5.1 respectively. Finally, we make a discussion and draw a conclusion in Section 6 and 7 individually.

2 Data

We use MovieLens 1M Dataset from Grouplens [2], consisting of ratings, users and movies data files, the last two files describe about basic information about users and movies. In the ratings file, formed with (users :: movies :: ratings :: timestamp), there are 6040 users with 3952 movies but only with 1000209 ratings(nearly 4% saturation, and each user make at least 20 ratings. In practice, we can fulfill the missing ratings based on methods described in Section 3 for recommendation. But in this assignment, our main goal is to evaluate the three methods' performance. Thus, we use Cross Validation to evaluate the accuracy of recommendation of these methods.

3 Methods

3.1 Cross Validation

We use cross validation to evaluate the performance of Naive Approach, UV Matrix Decomposition and Matrix Factorization. The idea of cross validation is: split the data into N parts of equal size. And the fitting the model with N times. One of the N parts as test data, remaining part is used as the training set. The final performance data of the model is obtained by averaging the results of N trials.

Cross validation is a very common method of measuring the goodness of a model in machine learning. It enables training using all data. It's result also more accurate than simply dividing the data into a training data and a test data. The disadvantage is that it requires N times of training, which increases the training time by a factor of N.

N can take any value. When N is equal to the size of the whole dataset, this is leave-one-out cross validation (LOOCV). In this assignment, if not specified, N takes the value of 5, which means that we use 5-fold cross validation to measure the performance of different models.

3.2 Measurements

We divide the data into a training set as well as a test set, and our aim is to predict the user's rating of a particular movie. Therefore we use Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) to measure the performance of models.

Assuming the size of the dataset is N, and denote the true value as y_i, the predicted value as $\hat{y}_i$ for the i-th sample in the dataset.

RMSE equals to the square root of mean value of all the error square, and error is the difference between true and predicted values, the equation could be seen as follow :

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{N}} \tag{1}$$

MAE equals to the sum of the absolute error which means the difference between true and predicted values, and then take the average value, the equation could be seen as follow :

$$MAE = \frac{\sum_{i=1}^{N} |y_i - \hat{y}_i|}{N} \tag{2}$$

3.3 Naive Approaches

There are five Naive Approaches to predict the ratings [3]:

Global Average: Global Average means that for all unknown ratings, we use the average of all ratings present in the training set to predict. That is, the predicted ratings are identical for any item for any user.

$$R_{global}(User, Item) = mean(all\ rating)$$

Item Average: Item Average means for an item, the unknown ratings are the average of the existing ratings for that item. For example, if both A and B are not rated for the item C, then their ratings are predicted to be the average rating for item C.

$$R_{Item}(User, Item) = mean(all\ ratings\ for\ Item)$$

User Average: User Average means for an user, the unknown ratings are the average of the existing ratings for that user. For example, if user A are not rated for an item C, then the rating is predicted to be the average rating for user A.

$$R_{User}(User, Item) = mean(all\ ratings\ for\ User)$$

Linear Regression without Intercept: A combination of Item Average and User Average. But do not use intercept term. We can use least squares method to solve this problem.

$$R_{User-Item}(User, Item) = \alpha * R_{User}(User, Item) + \beta * R_{Item}(User, Item)$$

Linear Regression with Intercept: A combination of Item Average and User Average and intercept. We can use least squares method to solve this problem too.

$$R_{User-Item}(User, Item) = \alpha * R_{User}(User, Item) + \beta * R_{Item}(User, Item) + \gamma$$

Since we split the dataset into training and test dataset, when we compute Item Average or User Average, some users or movies do not have any ratings. Thus, we cannot get the average ratings. At this time, we use the Global Average as replacement. This value called fall-back value.

3.4 UV Matrix Decomposition

UV matrix decomposition is a instance of Singular-value decomposition. Assuming the size of item-rating matrix R is $M \times N$. And U is a $M \times K$ matrix and V is a $K \times N$ matrix. The goal of matrix factorization is to find two matrices U and V from original user-item rating matrix R which satisfies:

$$R \approx U \times V \tag{3}$$

The main idea of UV-Decomposition is to adjust U and V to make RMSE smaller [4]. We update the elements of U and V in round-robin fashion. It means that we update of the elements row-by-row.

The update rules as follows [4]:

$$V_{r,s} = \frac{\sum_j V_{s,j}(R_{r,j} - \sum_{k \neq s} U_{r,k} V_{k,j})}{\sum_j V_{s,j}^2}$$
$$U_{r,s} = \frac{\sum_i U_{i,r}(R_{i,s} - \sum_{k \neq r} U_{i,k} V_{k,s})}{\sum_i U_{i,r}^2}$$

Note, that we only sum over all i that $R_{i,s}$ exists for $\sum_i R_{i,s}$.

3.5 Matrix Factorization

Assuming the size of item-rating matrix R is $M \times N$. And U is a $M \times K$ matrix and V is a $K \times N$ matrix. The goal of matrix factorization is to find two matrices U and V from original user-item rating matrix R which satisfies:

$$R \approx U \times V \tag{4}$$

The rows of U represent the features for each user. And the column of V represent the features for each item (movie). Parameter K is important which means the number of factors.

When we get U and V, it's easy to predict the ratings for each pair of User-Item:

$$\hat{R} = U \times V \tag{5}$$

The i-th row of $\hat{R}$ means the predicted ratings of user i for each item. Also, the j-th column of $\hat{R}$ means the predicted ratings of item j for each user. We initial U and V with sampling in a normal distribution $N(\mu = 0, \sigma = 1)$.

We use the stochastic gradient descent (SGD) to update the matrices U and V. Assuming i as the index of the user, and j means the index of the item. Update rules as follows [3]:

$$error = R_{i,j} - U_{i,} * V_{,j}$$
$$U_{i,} = U_{i,} + \eta * (error * V_{,j} - \lambda * U_{i,})$$
$$V_{,j} = V_{,j} + \eta * (error * U_{i,} - \lambda * V_{,j})$$

where η means the learning rate and λ means the regularization coefficient which prevents overfitting. The detailed derivation of the update formula can be referred to [5].

4 Results

4.1 Experimental Environment

All experiments in Section 4 were run in the same environment. Operation System: OS X Monterey. Memory: 32 G. Language: python 3.9.12. Detailed information about how to reproduce our results please refer to *README.md*.

4.2 Data Preprocess

Converts the raw data into a User-Item matrix. And we round all ratings values bigger than 5 to 5 and smaller than 1 to 1. Then, we split data into 5 parts. The following part are the results of 5-fold cross validation if not specified.

4.3 Naive Approaches

The results of the Naive Approaches are shown in the Table 1. Note that LR mean Linear Regression and MyMediaLite is the results from [6] for reference. Our results are very close to those of MyMediaLite, indicating that our implementation is correct.

We can see that Global Average has worst performance. This makes sense because the Global Average is simply one value to predict all cases. Item Average has better performance than User Average. The reason is that it is impossible for a user to rate all movies the same. But the ratings of a movie may be more convergent.

Item Average has the shortest running time, which is unexpected. This is because Global Average only needs to calculate one value while Item Average need to calculate the average ratings of all items. This may be related to the code implementation.

	RMSE			MAE			Running Time(s)
Model	Train	Test	MyMediaLite	Train	Test	MyMediaLite	
Global Average	1.117	1.117	1.117	0.934	0.934	0.934	3.402
User Average	1.028	1.036	1.036	0.823	0.829	0.827	5.940
Item Average	0.974	0.979	0.983	0.778	0.782	0.783	**3.345**
LR without Intercept	0.947	0.952	None	0.759	0.763	None	188.679
LR with Intercept	**0.915**	**0.924**	None	**0.725**	**0.733**	None	188.480

Table 1: Result of five Naive Approaches

4.4 UV Matrix Decomposition

We initialize the elements of U and V as:

$$\sqrt{\frac{a}{d}} + \epsilon$$

where a means the global average of matrix R (the average nonblank element). And ϵ is random noise and it follows a uniform distribution, $\epsilon \sim U(-1, 1)$.

Since it is very computationally intensive to calculate RMSE for each element update. Therefore, we set the algorithm termination rule to terminate the training after one iteration if the RMSE boost is less than *threshold*.

The configuration for experiments can be seen as Table 2. We tried setting the *num_factor* larger, but with a setting of 5, the training still did not end after more than 12 hours. Therefore we only conducted two sets of experiments.

Configuration	num_factor	threshold	num_iter
A	2	0.001	100
B	1	0.001	100

Table 2: The Configuration of UV Matrix Decomposition

4.4.1 Results of Different Configuration

Results of different configuration for UV Matrix Decomposition can be seen in Table 3. We can find that the performance of Configuration A and Configuration B is not much different. Configuration A has higher performance on the training set than Configuration B due to the number of features of 2. Its RMSE is slightly higher than that of Configuration B on the test set. Its MAE is lower than that of Configuration B.

Configuration	Train RMSE	Train MAE	Test RMSE	Test MAE	Running Time(s)
A	0.887	0.700	0.916	0.716	38.22
B	0.901	0.713	0.913	0.722	14.24

Table 3: Comparison of RMSE and MAE for train and test data for different configuration

4.4.2 Loss vs. Epoch for different Configuration

In this subsection, we want to explore how the loss changes during the training. Therefore, we divided the data into training and test sets (split ratio: 4 / 1), and did not perform 5-fold cross validation.

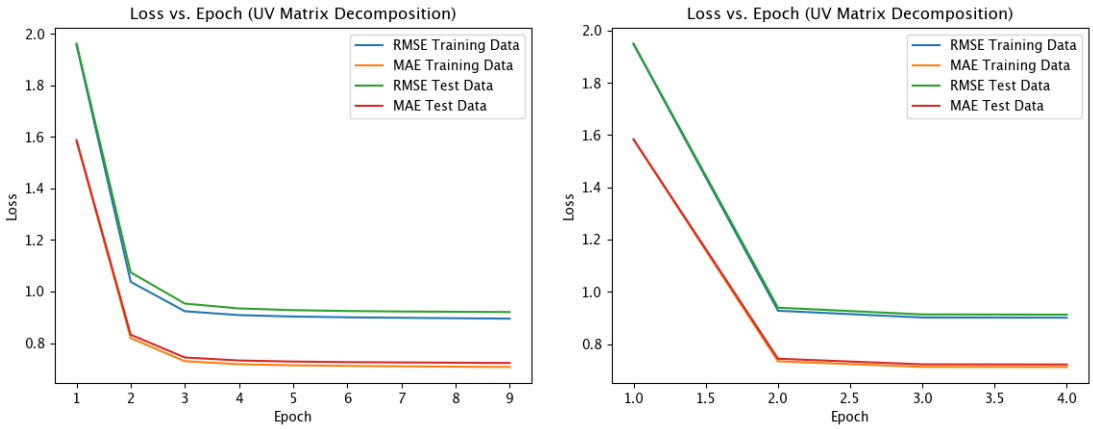


Figure 1: Left for Configuration A and right for Configuration B

As shown in Figure 1. When the *num_factor* increases, it takes longer time to converge. For example, when the *num_factor* is 2 (i.e. Configuration A), it takes 3 epochs before it is basically stable. Whereas, when *num_factor* is 1 (i.e. Configuration B), it takes only 2 epochs.

4.5 Matrix Factorization

We experiment with different parameter configurations. The configurations are shown in the Table 4. Baseline is the recommended configuration by MyMedialite website. The rest of the configuration has only one parameter changed compared to baseline. We explore the effect of different parameters on the model performance in this way.

Configuration	num_factor	regularization	learn_rate	num_iter
Baseline	10	0.05	0.005	75
A	10	**0.1**	0.005	75
B	10	**0.01**	0.005	75
C	**5**	0.05	0.005	75
D	**100**	0.05	0.005	75
E	10	0.05	**0.001**	75
F	10	0.05	**0.01**	75
G	10	0.05	0.005	**50**
H	10	0.05	0.005	**100**

Table 4: The Configuration of Matrix Factorization

4.5.1 Results of Different Configuration

Configuration	Train RMSE	Train MAE	Test RMSE	Test MAE	Running Time(s)
Baseline	0.807	0.640	0.881	0.694	2205.32
A	0.845	0.675	0.886	0.705	2193.53
B	0.803	0.631	0.912	0.709	2172.19
C	0.835	0.662	0.879	0.694	2183.45
D	0.637	0.498	0.941	0.725	2221.82
E	0.875	0.695	0.934	0.735	2166.83
F	0.793	0.628	0.877	0.690	2188.76
G	0.828	0.657	0.894	0.705	1457.85
H	0.795	0.630	0.873	0.687	2924.55

Table 5: Comparison of RMSE and MAE for train and test data for different configuration

Results of different configuration can be seen in Table 5. Configuration G and H has different running time due to their number of iteration lower or higher than others. Except for G and H, the running time for all other configurations are not very different. Configuration H has better performance than baseline, it means that we can obtain better performance by increasing number of iterations. In contrast, the performance of Configuration G is worse than that of Baseline.

Result of Configuration A indicates that when we increase the regularization coefficient, the performance is reduced compared to Baseline on both the training and test datasets. While result of Configuration B shows that when we decrease the regularization coefficient, the performance increases on the training dataset and decreases on the test dataset.

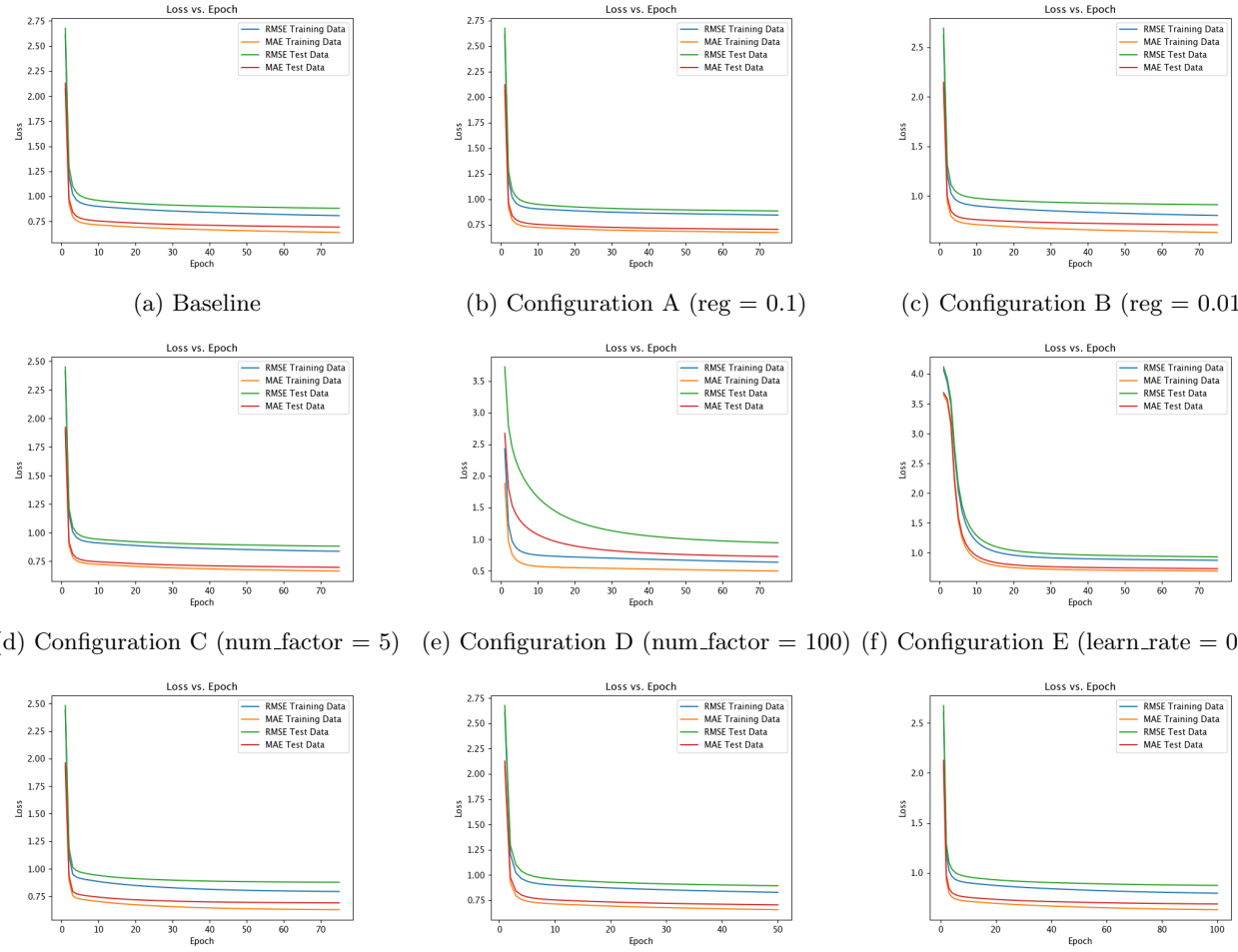
When we reduce the number of factors, as shown in the results of Configuration C, the performance on the training dataset as well as on the test set is reduced. But when the number of factors goes to 100, i.e., Configuration D, it causes the model to overfit.

The worse performance of Configuration E than Baseline indicates that the learning rate is too small and the model needs more iterations. And the performance of Configuration G is better than Baseline, which means we can turn up the learning rate to speed up the training process.

Compared with the results $RMSE = 0.857, MAE = 0.675$ of *MyMediaLite* (with the same parameters setting as our baseline model parameters), our results are approximately the same, indicating that our algorithm implementation is accurate.

4.5.2 Loss vs. Epoch for different Configuration

In this subsection, we want to explore how the loss changes during the training. Therefore, we divided the data into training and test sets (split ratio: 4 / 1), but did not perform 5-fold cross validation.



(a) Baseline (b) Configuration A (reg = 0.1) (c) Configuration B (reg = 0.01)

(d) Configuration C (num_factor = 5) (e) Configuration D (num_factor = 100) (f) Configuration E (learn_rate = 0.001)

(g) Configuration F (learn_rate = 0.01) (h) Configuration G (num_iter = 50) (i) Configuration H (num_iter = 100)

Figure 2: Loss vs. Epoch for different configuration

Result can be seen as Figure 2. When we turn down the learning rate, it takes more time to converge as shown in Figure 2 (f). And when we increase the number of factors, it also need more time to converge as shown in Figure 2 (e). The trend of change is similar between all other figures. Only the values of the final results differ.

When we increase the number of iteration, RMSE and MAE are still slowly declining, but not by much. Except for Figure 2 (d) and (f), In other configurations, the loss decreases most rapidly in the first 10 epochs. It indicates that 75 iterations is enough and if we want to improve our model most we can increase the number of iterations.

5 Data Visualization

The Matrix Factorization algorithm was employed in the first part to create artificial features that characterize the films and users. With these artificial features, can we cluster films and users? Is it possible to extract more potentially meaningful information from them? Visualizing the data of users and movies from the Matrix Factorization algorithm is necessary.

5.1 Methods for Data Visualization

To visualize a dataset, we must first ensure it is within three dimensions. However, data mining problems can involve hundreds or higher of data features. As a result, we need to reduce the dimensionality of the data first. PCA, t-SNE and UMAP are the commonly used dimensionality reduction algorithms. By this way, we can see how the data we obtained from Section 3.5 is distributed and whether it has some meaningful manner. A brief description about these algorithms as follows:

5.1.1 Principal Component Analysis (PCA)

Dimensionality reduction aims to retain as much of the information and structure of the high-dimensional data in the low-dimensional space [7]. PCA is a linear dimensionality reduction method. With PCA, data points from the high-dimensional space are projected into the low-dimensional space, preserving the distances between the data points. However, using PCA is not a good choice for data visualisation if the projection into two or three dimensional space does not preserve the major information of the original data.

5.1.2 t-distributed Stochastic Neighbor Embedding (t-SNE)

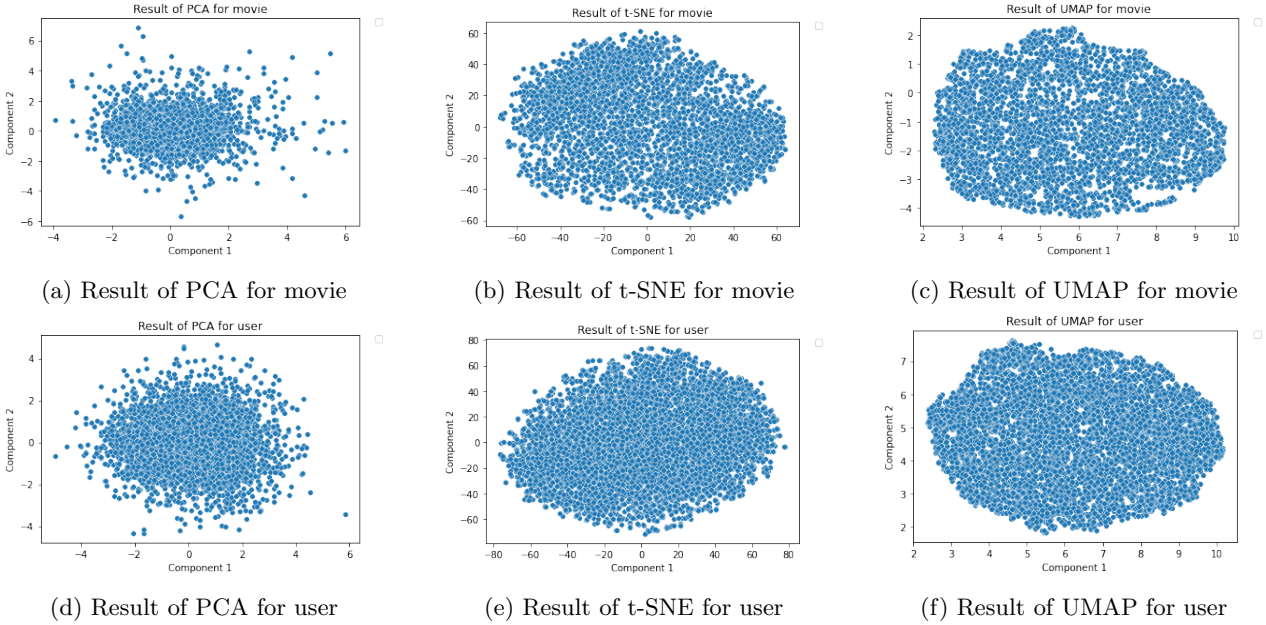
t-SNE created in 2008 is a non-linear dimensionality reduction. It projects high-dimensional data points into two- or three-dimensional space by focusing on keeping the similarities between original data points. In detail, t-SNE first constructs a probability function P to represent the neighbour relationship between two high-dimensional points x_i and x_j. Then, it constructs another probability function Q to represent the neighbour relationship between two low-dimensional points y_i and y_j. Finally, it builds a KL(Kullback-Leibler) divergence as the loss function and use the gradient descent learning method to make P distribution and Q distribution as close as possible. In this way, high-dimensional points x_i and x_j are projected into low-dimensional space by points y_i and y_j [7].

5.1.3 Uniform Manifold Approximation and Projection (UMAP)

UMAP is created in 2018. In application it is similar to t-SNE: UMAP also constructs a probability function to represent the relationship between two high-dimensional points, another probability function used to represent the relationship between the low-dimensional points, and finally constructs a loss function and uses a gradient descent learning method to make the above two functions as close as possible. As a result, UMAP can be seen as an improved version of the t-SNE method. Compared to t-SNE, it preserves local information better, improves calculation efficiency and allows for dimensionality reduction in arbitrary dimensions [8].

5.2 Results for Data Visualization

As shown in Figure 3, the scatter plots of data visualization using PCA are less dispersive than the t-SNE and UMAP algorithms. This is due to the fact that when data is compressed to two dimensions by PCA, less than 30 % (25.2 % for movie and 29.8 % for user) of the original information is maintained. PCA is unable to distribute data points on a two-dimensional plot adequately due to the significant information loss. The scatter plots of t-SNE and UMPA are similar. UMAP retains more local information than t-SNE and is a more effective data visualization tool. In order to whether if any meaningful cluster is present, we labeled the results of UMAP for movie by year and genre, user by age, gender and occupation.

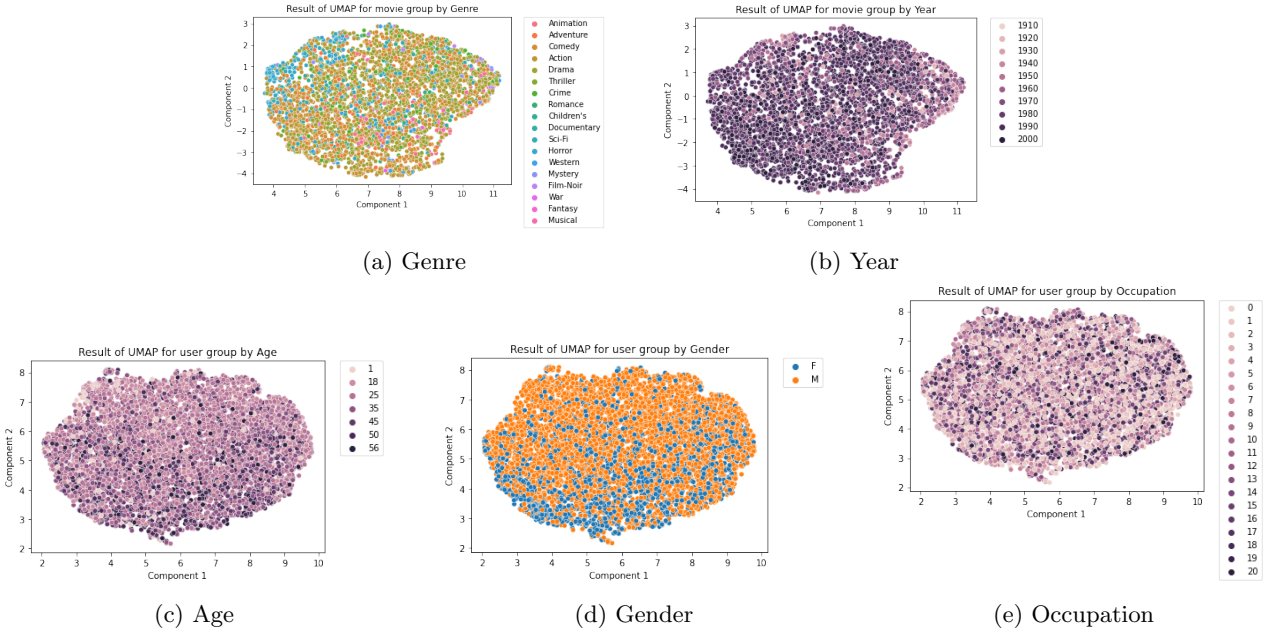


(a) Result of PCA for movie (b) Result of t-SNE for movie (c) Result of UMAP for movie

(d) Result of PCA for user (e) Result of t-SNE for user (f) Result of UMAP for user

Figure 3: Results of Data Visualization for movie and user

After dimensionality reduction and data visualization of the movie and user data like Figure 4 showing, no meaningful clusters are present, and the points of various groups are mixed. Possible reasons for this result are:
Firstly, we created ten artificial features in the 4.5 Matrix Factorization, but ten features are too few for movie and user data to mine the characteristics of original data significantly. As a result, no clusters are observed after the data visualization.
Secondly, the points after coloring are mixed, probably because age, and occupation have little effect on the differentiation of users. The genre should be influential for the movie differentiation, but there are too many categories of genre, resulting in insufficient differentiation. If the number of genre categories was reduced, it might be possible to obtain valid information.



(a) Genre (b) Year

(c) Age (d) Gender (e) Occupation

Figure 4: Label the Results of UMAP for movie and user

6 Discussion

In this section, we will discuss the time complexity and memory complexity of the algorithms we have implemented. Suppose there are M movies and U users, and R ratings. The dimension of the rating matrix is $U \times M$. Assuming the number of factor for both matrix factorization and UV matrix decomposition are K. The time complexity and memory complexity for each algorithm can be seen as Table 6.

Algorithms	Time Complexity	Memory Complexity
Global Average	$\mathcal{O}(R)$	$\mathcal{O}(1)$
Item Average	$\mathcal{O}(UM)$	$\mathcal{O}(M)$
User Average	$\mathcal{O}(UM)$	$\mathcal{O}(U)$
Linear Regression	$\mathcal{O}(R)$	$\mathcal{O}(1)$
UV Matrix Decomposition	$\mathcal{O}(UK + KM)$	$\mathcal{O}(UK + KM)$
Matrix Factorization	$\mathcal{O}(R)$	$\mathcal{O}(UK + KM)$

Table 6: Time Complexity and Memory Complexity of the Implemented Algorithms

Global Average needs to iterate the whole rating list. Thus its time complexity is $\mathcal{O}(R)$. And it only needs to memory the whole average so its memory complexity is $\mathcal{O}(1)$. Item Average and User Average need to iterate the rating matrix thus its time complexity is $\mathcal{O}(UM)$. And they need to store the average per user or movie, so the memory complexity are $\mathcal{O}(M)$ for Item Average and $\mathcal{O}(U)$ for User Average.

The time complexity of Linear Regression is $\mathcal{O}(R)$ since we need iterate all the ratings. While its memory complexity is $\mathcal{O}(1)$ because we only need to store and update the two features.

In UV Matrix Decomposition, we need to update the matrices U and V, thus the time complexity is $\mathcal{O}(UK + KM)$. And we need to store the two matrices, so the memory complexity is also $\mathcal{O}(UK + KM)$.

In Matrix Factorization, we need to iterate all ratings and update the matrices U and V too. So the time complexity is $\mathcal{O}(R)$ and the memory complexity is $\mathcal{O}(UK + KM)$.

7 Conclusion

In this assignment, we explored recommender system. Three classical algorithms were tried to be implemented: Naive Approaches, UV-Matrix Decomposition, and Matrix Factorization. Naive Approaches is the simplest idea, but also the least effective. Matrix Factorization is the best and can further improve performance when increasing the number of iterations (performance of Configuration H is better than other configurations). In the future, we can improve the code implementation to reduce running time, which is currently the main bottleneck.

Then we visualize the data using the features obtained by Matrix Factorization. The results are not satisfactory, probably because 10 features are not enough to summarize the required information. In the future, we will consider to increase the number of features and then examine whether the results will become better.

References

[1] "Netflix." [Online]. Available: https://en.wikipedia.org/wiki/Netflix

[2] "Grouplen." [Online]. Available: https://grouplens.org/datasets/movielens/

[3] W. Kowalczyk, "Lecture notes in advances in data mining (recommender systems)," October 2022.

[4] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of Massive Datasets, 2nd Ed.* Cambridge University Press, 2014. [Online]. Available: http://www.mmds.org/

[5] G. Takács, I. Pilászy, B. Németh, and D. Tikk, "Major components of the gravity recommendation system," *Acm Sigkdd Explorations Newsletter*, vol. 9, no. 2, pp. 80–83, 2007.

[6] Netflix. MyMediaLite: Example Experiments. [Online]. Available: http://mymedialite.net/examples/datasets.html

[7] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE." *Journal of machine learning research*, vol. 9, no. 11, 2008.

[8] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint arXiv:1802.03426*, 2018.