Netflix Movie Recommendation Engine

Hetu Feng, Jiacheng Liu, Linle Jiang, Xiao Yang

1. INTRODUCTION

Netflix is the leading provider of online video streaming services. Its business model allows users to subscribe to its monthly service and to cancel at any time. Therefore, in order to keep users on the platform and not to churn, building a robust recommendation system is the key to the business. Generally, a recommender system provides users with suggested products or services that best suit their needs. For Netflix, having a recommendation system that offers personalized content for users can increase user stickiness.

Therefore, this project aims to apply machine learning algorithms to develop a recommendation system for Netflix. This project originated from the Netflix Prize Open Competition in 2006. The data is gathered from kaggle's competition section. There are 4 training sets in the dataset and each contains 20M rows of ratings. In total, there are over 17K movies and 500k users. Among many available algorithms, the main approaches used in the current project are Popularity and Collaborative Filtering.

2. DATA PREPARATION

Each training set contains user id, movie id, user rating of the movie and the date the rating was given. Each movie id is in its own line followed by a colon. And the subsequent lines underneaths the movie id are all the ratings of that movie with user id and date. In one training set, there are more than 24,000,000 rating entries for a total of 4,500 movies. Although incomplete data may decrease the model accuracy, the fact that a memory error occurred in the subsequent data manipulation stage as we combined all 4 training sets, we decided to retain only dataset 1 for the demonstration of this project.

We extracted all movie ids from its row and mapped them to their corresponding rating rows, so that we could organize all data into a table with 4 columns containing user_id, rating, rating_date and movie_id.

3. EXPLORATORY ANALYSIS

For the 4,500 movies and 24,053,764 ratings given by 470,758 users, the distribution of ratings are as follows. First, not every user rated every movie. From the distribution (Figure 3.1), only 15% of the ratings were lower than 3, while more than half of the ratings are 4 or 5. However, this statistic might be biased since most people only rated the movies they watched, and these movies may skew towards their own interests, leading to a negative distribution. This could lead to a result that ‘good’ movies received more and higher ratings, while ‘bad’ movies received less and lower ratings. This was taken into account at the modeling stage.

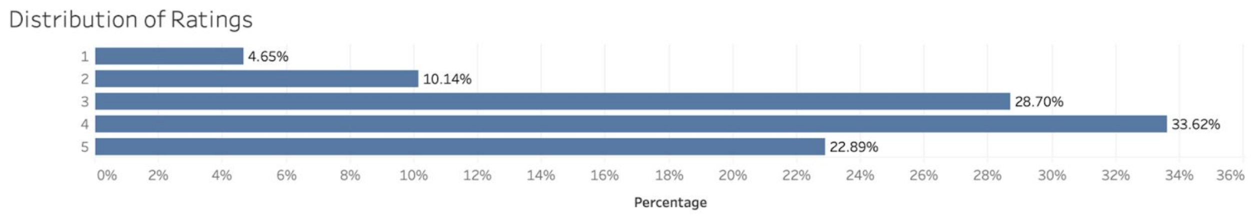


Figure 3.1. Distribution of Movie Ratings

An interesting finding (Figure 3.2) was that the number of movie ratings given by users increased over the years and reached 2 peaks in 2005 with a sudden drop afterwards. This might suggest that more recent movies tend to receive more ratings, compared to older movies.

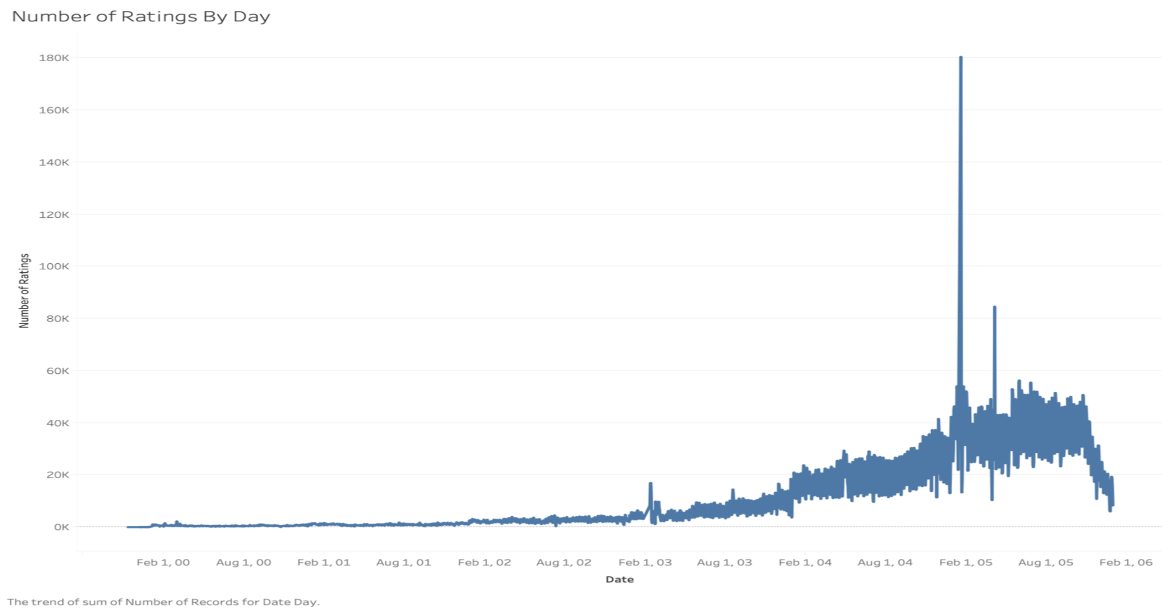


Figure 3.2. Distribution of Movie Ratings Grouped By Day

Then, we analyzed the distribution of ratings per movie. In Figure 3.3, the x-axis represents the bin of number of ratings per movie and we used 200 ratings as a bin. The y-axis represents the number of movies. The distribution was nearly exponential. Most movies were distributed in the first several bins. About 56% of the movies were rated less than 1,000 times. For a movie, 1,000 ratings are relatively small and may be insufficient to represent the general performance of the movie. Thus, we need to exclude movies without enough ratings because they could be biasing in the following algorithm implementation.

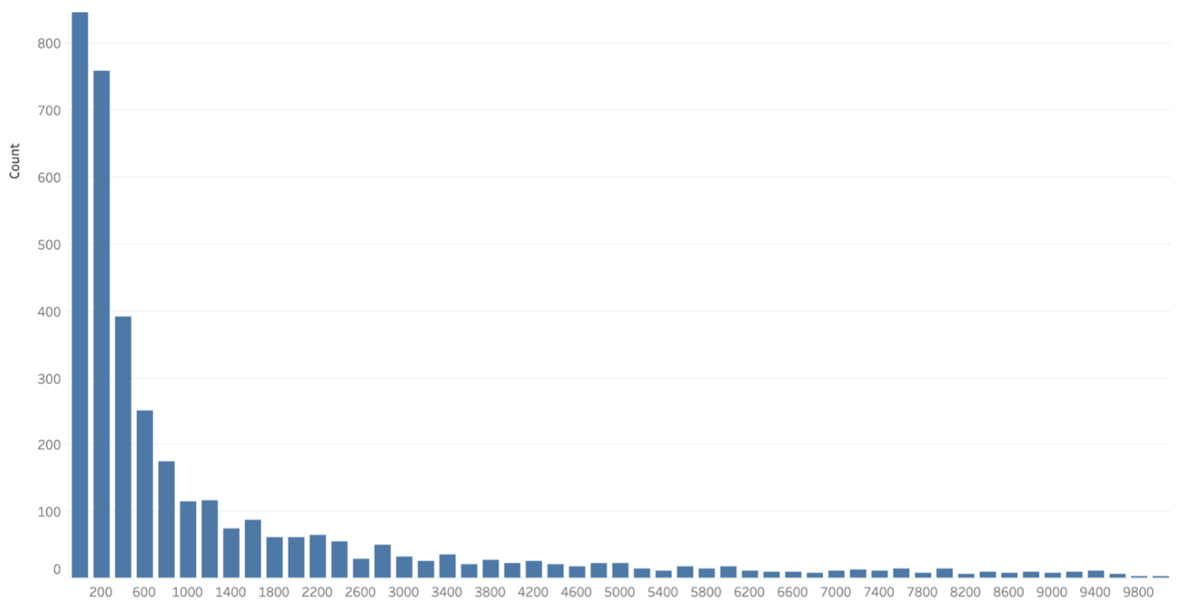


Figure 3.3. Ratings Per Movie

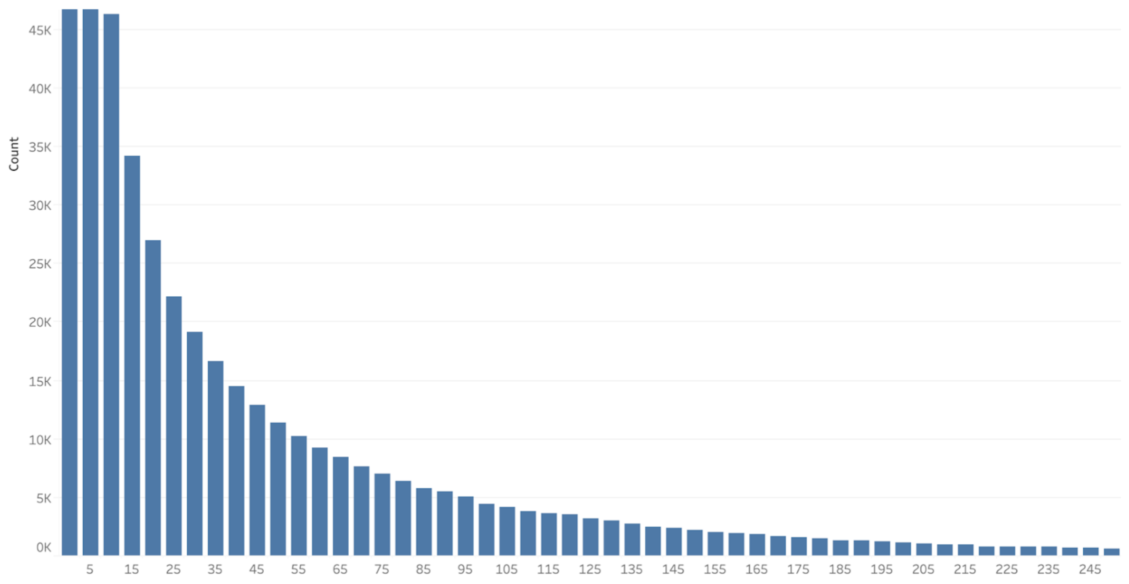


Figure 3.4. Ratings Per User

Similarly, we analyzed the rating in terms of users. In Figure 3.4, the x-axis represents the bin of number of ratings per user and we used 5 ratings as a bin. The y-axis represents the count of each bin. It showed that most users were distributed in the first several bins as well. The distribution was also strongly positively skewed. About 30% of the users rated movies less than 15 times and about 64% of the users rated the movies less than 50 times. For the same reason mentioned before, we excluded the outliers.

As a result, to reduce the dimensionality of the dataset, we decided to filter out sparse data. Based on the above analysis we used movies with at least 5,000 ratings and users who rated more than 400 times to build the models. After the filtering procedure, 2,607 users and 774 movies with a total of 939,490 ratings were retained for modeling.

4. MACHINE LEARNING MODELS

There are several popular approaches that can be used to develop a recommender system. When choosing from Popularity-based, Collaborative Filtering, Content-Based and other approaches, we considered several factors (Ajitsaria, 2019). The data for this project was not sufficient to perform Content-based filtering, because the only information that was relevant to content was movie titles, yet some of them were not even complete. For Collaborative filtering, both Memory-based and Model-based techniques were applied. In terms of memory-based technique, we prefered item-based over user-based filtering for its potential accuracy and processing speed. Considering the fact that the number of users was way beyond the number of movies in our data, there were more potential neighbors for a single user (user-based), while less ratings for similarity calculation than for a single movie (item-based). In this case, it might be better to choose the Item-based method as it used a small number of highly confident neighbors, which could technically achieve more accurate predictions while costing less computational power. However, we still implemented both methods to test our hypothesis.

Given the large and sparse nature of the user-movie rating matrix, scalability and sparsity become the challenges to construct an effective recommendation system (Thorat, Goudar, & Barve, 2015). In this case, the goal of this project is to build a recommendation system upon a training dataset with nearly a million users and thousands of movies, yet only a very small fraction of ratings was actually available. To tackle these challenges, we turned to the model-based collaborative filtering approach. Specifically, the Singular Value Decomposition (SVD) was applied. This method was proposed by Simon Funk in the Netflix Prize Competition, and it is recognized as a recommendation system algorithm that works better in practice with respect of prediction accuracy and computational efficiency (Funk, 2006).

In order to build the optimal prediction model, the following approaches were implemented, evaluated and compared in the following section:

- Popularity method with Normal Mean Rating & Weighted Mean Rating
- Item-Based Collaborative Filtering with KNN using Cosine & Pearson Similarity
- User-Based Collaborative Filtering with KNN using Cosine & Pearson Similarity
- Model-based Collaborative Filtering with SVD

We will be using Rooted Mean Squared Error (RMSE) to evaluate the performance of the systems. Since we divided the dataset into train set and test set previously, RMSE was used to quantify the accuracy by comparing model estimated ratings against observed ratings. We chose RMSE over Mean Absolute Error (MAE) because it imposed a penalty over large errors. In other words, RMSE increases as the error magnitude increases, but this was not the case for MAE. The calculations are as follows:

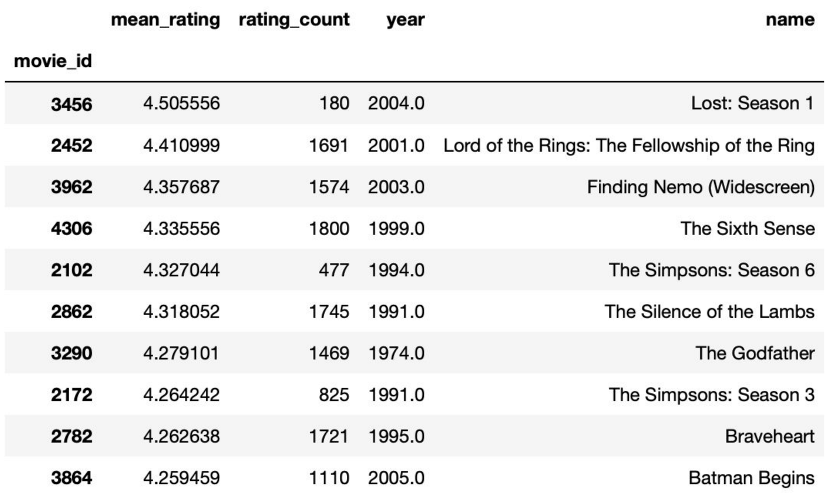
$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(y_{pred,i} - y_i)^2}{n}} \quad \text{(Equation 4.1)}$$

$$\text{MAE} = \frac{\sum_{i=1}^{n}\left|y_{pred,i} - y_i\right|}{n} \quad \text{(Equation 4.2)}$$

4.1 Popularity

There are cases when a user is new to the platform and we know nothing about his/her preference, and thus it is impossible to recommend any movies to users with Collaborative Filtering. Therefore, we could recommend the most popular movies to the user by computing mean ratings of all movies for a ranking. One thing to note is that the recommendations here stay the same for every user with no variations and personalization.

The first approach was to use a mean rating for each movie. Table below (Table 4.1.1) displays the top 10 recommended movies based on this method. This model had a RMSE of 1.0423. One drawback of this mean rating approach was that it didn't account for the amount of ratings for movies. For instance, 'Lost Season 1' in the table became the highest ranking movie. Alternatively, movies with more ratings typically retained less extreme scores after averaging and thus could be ranked lower.



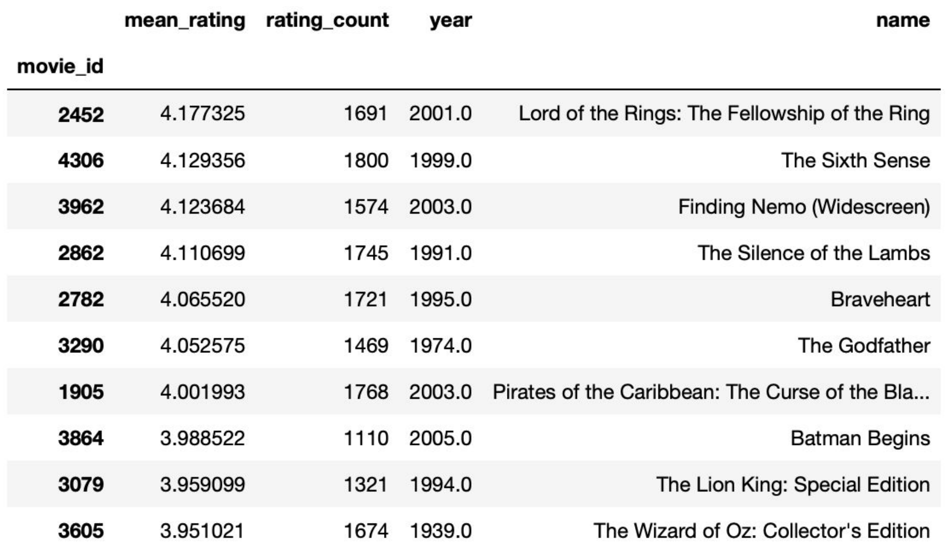
	mean_rating	rating_count	year	name
movie_id				
3456	4.505556	180	2004.0	Lost: Season 1
2452	4.410999	1691	2001.0	Lord of the Rings: The Fellowship of the Ring
3962	4.357687	1574	2003.0	Finding Nemo (Widescreen)
4306	4.335556	1800	1999.0	The Sixth Sense
2102	4.327044	477	1994.0	The Simpsons: Season 6
2862	4.318052	1745	1991.0	The Silence of the Lambs
3290	4.279101	1469	1974.0	The Godfather
2172	4.264242	825	1991.0	The Simpsons: Season 3
2782	4.262638	1721	1995.0	Braveheart
3864	4.259459	1110	2005.0	Batman Begins

Table 4.1.1. Mean Rating Top 10 Recommended Movies

To improve the mean rating method, we took the number of ratings into account by using weighted mean rating (Equation 4.1.1), a method used by IMDb.

$$\text{weighted rating (WR)} = \frac{v}{v+m} \times R + \frac{m}{v+m} \times C \quad \text{(Equation 4.1.1)}$$

It takes into account the number of votes each movie received (v), minimum ratings required to be on the list (m), and the mean rating for all movies (C). This model yielded a RMSE of 1.1240 and below showed the top recommendations.



	mean_rating	rating_count	year	name
movie_id				
2452	4.177325	1691	2001.0	Lord of the Rings: The Fellowship of the Ring
4306	4.129356	1800	1999.0	The Sixth Sense
3962	4.123684	1574	2003.0	Finding Nemo (Widescreen)
2862	4.110699	1745	1991.0	The Silence of the Lambs
2782	4.065520	1721	1995.0	Braveheart
3290	4.052575	1469	1974.0	The Godfather
1905	4.001993	1768	2003.0	Pirates of the Caribbean: The Curse of the Bla...
3864	3.988522	1110	2005.0	Batman Begins
3079	3.959099	1321	1994.0	The Lion King: Special Edition
3605	3.951021	1674	1939.0	The Wizard of Oz: Collector's Edition

Table 4.1.2. Weighted Mean Rating Top 10 Recommended Movies

4.2 Collaborative Filtering Memory-Based

The principle of Collaborative filtering algorithms is to discover the user's preference by mining the users' historical behavior data, divides users into groups based on different preferences and recommends products with similar taste. Collaborative filtering recommendation algorithms are divided into two categories, memory-based collaborative filtering and model-based collaborative filtering. User-based and item-based algorithms are both memory-based types.

4.2.1 User-Based Collaborative Filtering

The basic idea of user-based algorithms is similar to that recommended by a friend. For example, suppose that by analyzing the items the user likes, it is found that user A and user B are very similar. If user B likes an item, while user A has not liked it, the algorithm recommends this item to user A. This algorithm uses the nearest-neighbor algorithm to find a user's neighbor set. The users in this set have similar preferences as the target user. The algorithm predicts the user based on the neighbor's preferences. In this case, the two metrics we used to find the nearest neighbors of a user and calculate the similarity are cosine similarity and Pearson correlation. The cosine similarity (Equation 4.2.1) calculates the cosine of the angle between two user vectors:

$$\mathrm{sim(i, j) = cos(i, j)} = \frac{\sum_{k=1}^{n} R_{ik} R_{jk}}{\sqrt{\sum_{k=1}^{n} R_{ik}^{2} \sum_{k=1}^{n} R_{jk}^{2}}} \quad \text{(Equation 4.2.1)}$$

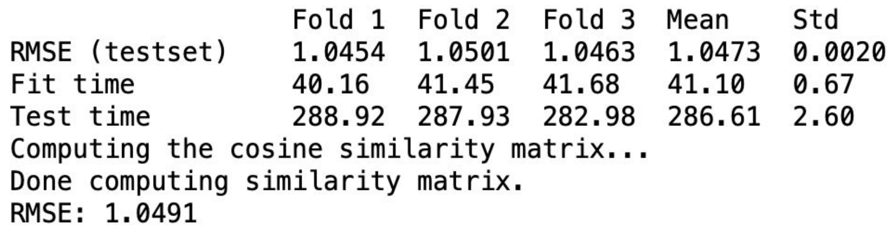
Correlation-based similarity (Equation 4.2.2) calculates the Pearson correlation between two user vectors:

$$\mathrm{sim(i, j) = corr(i, j)} = \frac{\sum_{k} (R_{ik} - \overline{R}_{i})(R_{jk} - \overline{R}_{j})}{\sqrt{\sum_{k} (R_{ik} - \overline{R}_{i})^{2}} \sqrt{\sum_{k} (R_{jk} - \overline{R}_{j})^{2}}} \quad \text{(Equation 4.2.2)}$$

Where i and j stands for two users, R_{ik} is the rating that user i gave to movie k, $\overline{R}_{i}$ is the mean rating given by user i. In this step, we get a m x m user similarity matrix, where m is the number of users calculated. Next step is to predict the items these similar users liked (Equation 4.2.3). Using the ratings of movies given by users to perform weighted summation, where the weight is the similarity between each two users, and then divided by the sum of similarities of all users to calculate the predicted ratings that the user would give to the movies:

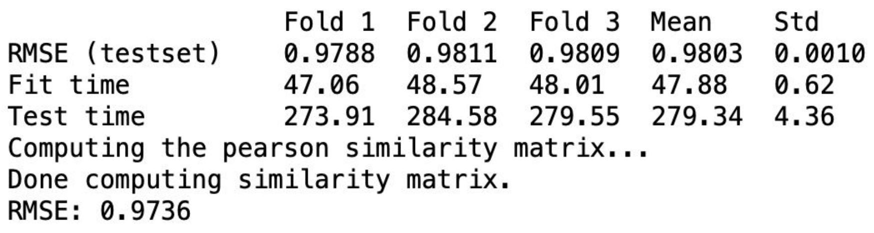
$$\mathrm{P(i, j)} = \frac{\sum_{i=1}^{m} (sim(i,j) * R)}{\sum_{i=1}^{m} sim(i,j)} \quad \text{(Equation 4.2.3)}$$

In our implementation, we used the Surprise KNNBasic library function to realize and test the performance of the user-based collaborative filtering algorithm. Comparing the results from using both cosine similarity (Figure 4.2.1) and Pearson correlation (Figure 4.2.2):



```
                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    1.0454  1.0501  1.0463  1.0473  0.0020
Fit time          40.16   41.45   41.68   41.10   0.67
Test time         288.92  287.93  282.98  286.61  2.60
Computing the cosine similarity matrix...
Done computing similarity matrix.
RMSE: 1.0491
```

Figure 4.2.1. User-Based Cosine Similarity RMSE



```
                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.9788  0.9811  0.9809  0.9803  0.0010
Fit time          47.06   48.57   48.01   47.88   0.62
Test time         273.91  284.58  279.55  279.34  4.36
Computing the pearson similarity matrix...
Done computing similarity matrix.
RMSE: 0.9736
```

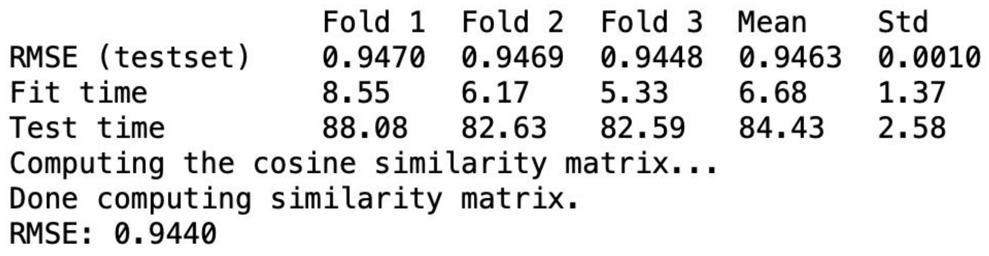
Figure 4.2.2. User-Based Pearson Correlation RMSE

We ran 3 times of cross validation on both tries, results showing that Pearson correlation had a better performance since it had a lower value on the average RMSE score.

4.2.2 Item-Based Collaborative Filtering

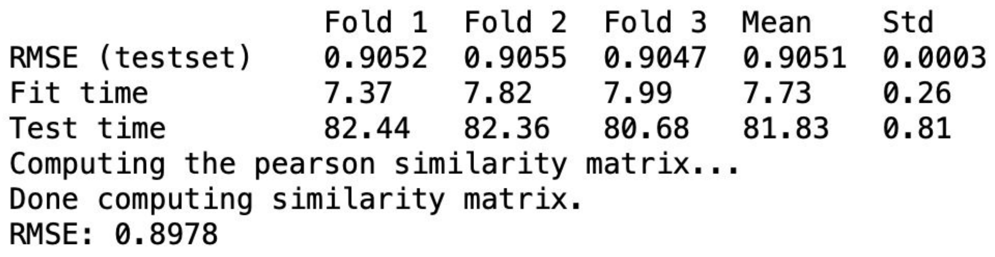
The basic idea of item-based collaborative filtering is similar to the user-based, except that it calculates the similarity between items based on the historical preference data of all users, and recommends similar items to the user.

The principle of item-based is pretty much the same as user-based. In our implementation, we also used the Surprise KNNBasic library function by changing the sim_options to realize and test the performance of the item-based collaborative filtering algorithm. Comparing the results from using both cosine similarity (Figure 4.2.3) and Pearson correlation (Figure 4.2.4):



```
                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.9470  0.9469  0.9448  0.9463  0.0010
Fit time          8.55    6.17    5.33    6.68    1.37
Test time         88.08   82.63   82.59   84.43   2.58
Computing the cosine similarity matrix...
Done computing similarity matrix.
RMSE: 0.9440
```

Figure 4.2.3. Item-Based Cosine Similarity RMSE



```
                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.9052  0.9055  0.9047  0.9051  0.0003
Fit time          7.37    7.82    7.99    7.73    0.26
Test time         82.44   82.36   80.68   81.83   0.81
Computing the pearson similarity matrix...
Done computing similarity matrix.
RMSE: 0.8978
```

Figure 4.2.4. Item-Based Pearson Correlation RMSE

Results from the item-based methods generated the same pattern as the used-based approach, in that Pearson correlation showed a better performance. Add on to that, as a whole, 1) the prediction quality of item-based algorithm was a little higher than that of user-based; 2) user-based was much more costly than item-based on fitting and testing processes, since in our dataset there were much more users than the number of movies; 3) item-based technique had a less standard deviation on RMSE score, which means that its overall performance was more stable.

4.3 Collaborative Filtering Model-Based

The core idea of this approach is matrix factorization, that is, dimensionality reduction using unsupervised learning technique (Girase & Mukhopadhyay, 2015). Specifically, this goal of this approach is to decompose the large and sparse user-movie rating matrix into the product of two long yet thin matrices (i.e., a user matrix and a movie matrix) that minimized the prediction error (i.e., RMSE). With these two matrices, all missing values in the user-movie rating matrix can be computed. In the user matrix, the rows represent users and the columns indicate a small number of latent factors, such as preference feature factors. While in the movie matrix, there is the same number of latent factor rows as the columns in the user matrix. These are movie characteristics factors. On the other hand, the rows in the movie matrix represent the movies. Notably, the semantic meanings of latent factors are equivalent between the two matrices. In this project, we used the SVD algorithm (Funk, 2006) for the model-based collaborative filtering approach.

In the algorithm, the predicted rating of user *u* to movie *i* $\widehat{r}_{ui}$ can be computed (Equation 4.3.1):

$$\widehat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u \quad \text{(Equation 4.3.1)}$$

Where μ denotes the overall average rating, and b_u and b_i represent the user biases and movie biases, correspondingly. Moreover, q_i indicates the item factors and p_u denotes the user factors.

To compute the missing value of the user-movie rating matrix, the objective function (Equation 4.3.2) is to minimize the regularized squared error:

$$\sum_{r_{ui} \in R_{train}} (r_{ui} - \widehat{r}_{ui})^2 + \lambda (b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2) \quad \text{(Equation 4.3.2)}$$

This minimization task is computed with stochastic gradient descent (Equation 4.3.3 - Equation 4.3.6):

$$b_u \leftarrow b_u + \gamma (r_{ui} - \widehat{r}_{ui} - \lambda b_u) \quad \text{(Equation 4.3.3)}$$

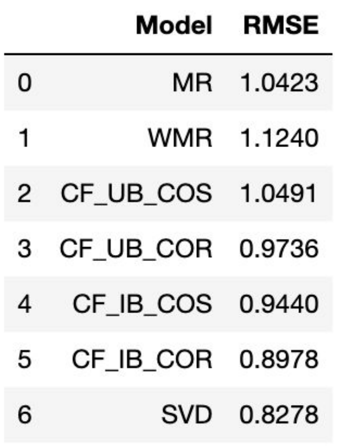
$$b_i \leftarrow b_i + \gamma\,(r_{ui} - \widehat{r}_{ui} - \lambda b_i) \quad \text{(Equation 4.3.4)}$$
$$p_u \leftarrow p_u + \gamma\,[(r_{ui} - \widehat{r}_{ui}) \cdot q_i - \lambda p_u] \quad \text{(Equation 4.3.5)}$$
$$q_i \leftarrow q_i + \gamma\,[(r_{ui} - \widehat{r}_{ui}) \cdot p_u - \lambda q_i] \quad \text{(Equation 4.3.6)}$$

Where γ denotes the learning rate and λ denotes the regularization term. These steps were computed with all training data and repeated for *n* times (i.e., epochs).

In this project, we used the aforementioned algorithm on the complete training dataset with 3-fold cross validations to tune the hyper-parameters that return the minimum RMSE. This search was performed with the grid search method. We first specified a series of epochs (i.e., 15, 20, 25), number of latent factors (i.e., 25, 30, 35, 40, 100), learning rates (i.e., 0.001, 0.003, 0.005, 0.008), and regularization rates (i.e., 0.08, 0.1, 0.15, 0.02), and used this grid search method to identify the best combination of the above hyper-parameters for the optimal model. We found that when the SVD model was set with 25 epoches, 25 latent factors, a learning rate of 0.005, and a regularization rate of 0.02, it resulted in the minimized RMSE. Therefore, we applied this set of hyper-parameters on the SVD model to train the prediction model and examined its accuracy on a test dataset. Our result suggested that this tuned SVD model had the optimal predictive performance, with a RMSE of 0.8278.

5. EVALUATION

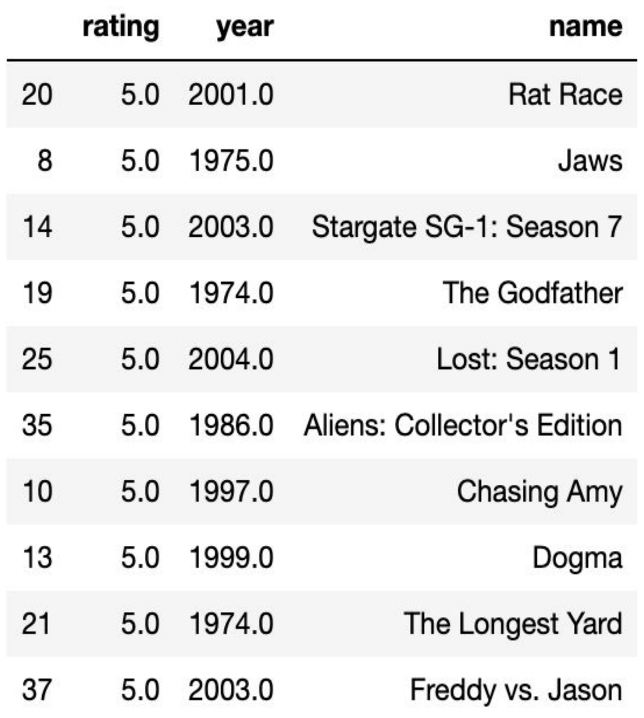
We used RMSE as the metric to evaluate the performance of all the proposed models. From the table 5.1 below, SVD outperformed other approaches with the lowest RMSE (0.8278), followed by Item-based CF using Pearson correlation with the RMSE of 0.8978. The result supported our hypothesis that Item-based CF worked better than User-based CF for our dataset. Last but not least, Collaborative Filtering approaches performed better than Popularity methods overall.



	Model	RMSE
0	MR	1.0423
1	WMR	1.1240
2	CF_UB_COS	1.0491
3	CF_UB_COR	0.9736
4	CF_IB_COS	0.9440
5	CF_IB_COR	0.8978
6	SVD	0.8278

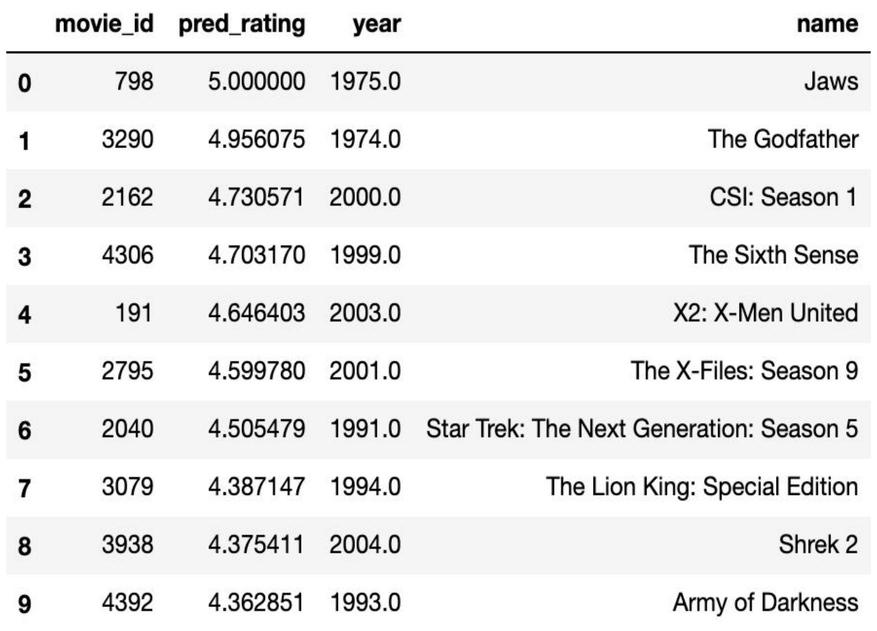
Table 5.1. Model RMSE Comparison

The following tables give a simple demonstration of SVD recommendations. Given user 828444, table 5.2 are 10 of the top movies he liked in the past. Table 5.3 are the top 10 movies our system recommends to the user 828444 based on estimated ratings by SVD.



	rating	year	name
20	5.0	2001.0	Rat Race
8	5.0	1975.0	Jaws
14	5.0	2003.0	Stargate SG-1: Season 7
19	5.0	1974.0	The Godfather
25	5.0	2004.0	Lost: Season 1
35	5.0	1986.0	Aliens: Collector's Edition
10	5.0	1997.0	Chasing Amy
13	5.0	1999.0	Dogma
21	5.0	1974.0	The Longest Yard
37	5.0	2003.0	Freddy vs. Jason

Table 5.2. Movies User 828444 Liked



	movie_id	pred_rating	year	name
0	798	5.000000	1975.0	Jaws
1	3290	4.956075	1974.0	The Godfather
2	2162	4.730571	2000.0	CSI: Season 1
3	4306	4.703170	1999.0	The Sixth Sense
4	191	4.646403	2003.0	X2: X-Men United
5	2795	4.599780	2001.0	The X-Files: Season 9
6	2040	4.505479	1991.0	Star Trek: The Next Generation: Season 5
7	3079	4.387147	1994.0	The Lion King: Special Edition
8	3938	4.375411	2004.0	Shrek 2
9	4392	4.362851	1993.0	Army of Darkness

Table 5.3. Top 10 Movie Recommendations for User 828444

Limited to the computing power, this research project only used a subset of the original data. With all data included, results should be different and that the RMSEs might be lower for all the approaches with an increased sample size. Nonetheless, with a substantial subset of the original data, our results highlighted a clear pattern in terms of predictive performance across the proposed models. Moreover, we believe SVD could make even more accurate predictions with a larger dataset due to its usage of Stochastic Gradient Descent.

However, because Collaborative Filtering depends on user-item interaction, cold-start could be an issue for our models if a movie or a user is new to the platform with no or little information. In order to make the system perform better in this scenario, we would use Popularity methods to create a baseline for the recommendation system. If we were able to acquire more metadata about users and movies, Content-based filtering and other methodologies could be applied to investigate their predictive performance.

6. CONTRIBUTIONS

We collaborated on all the works.

BIBLIOGRAPHY

Ajitsaria, A. (2019, July 22). Build a Recommendation Engine With Collaborative Filtering. Retrieved from https://realpython.com/build-recommendation-engine-collaborative-filtering/

Bokde, Dheeraj, Girase, Sheetal, Mukhopadhyay, & Debajyoti. (2015, March 25). Role of Matrix Factorization Model in Collaborative Filtering Algorithm: A Survey. Retrieved from https://arxiv.org/abs/1503.07475

Chandarana, N. (2019, August 30). SVD: Where Model Tuning Goes Wrong. Retrieved from https://towardsdatascience.com/svd-where-model-tuning-goes-wrong-61c269402919

Funk, S. (2006, December 11). Netflix Update: Try This at Home. Retrieved from https://sifter.org/~simon/journal/20061211.html

Grover, P. (2020, March 31). Various Implementations of Collaborative Filtering. Retrieved from https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0

Hug, N. (2015). Matrix Factorization-based algorithms. Retrieved from https://surprise.readthedocs.io/en/stable/matrix_factorization.html

Thorat, P. B., Goudar, R. M., & Barve, S. (2015). Survey on collaborative filtering, content-based filtering and hybrid recommendation system. *International Journal of Computer Applications*, 110(4), 31-36. https://doi.org/10.5120/19308-0760