Springboard Capstone Project Final Report

Recommender Systems

Data Science Career Track

Changmin Cheng

March 2022

Table of Content

1. Problem Description

Recommender system plays a critical rule in many hi-tech businesses in this time of e-commerce. When you buy one item at Amazon, most likely you will be recommended with something you might be interested to purchase. When you browse a website, most of your browsing clicks will be recorded by the website, such as items you purchased or saved in your favorite folder, or you have viewed. Based on this information from you as a user, the built-in recommender system will recommend items you might be interested. By doing this, the company is trying to encourage customers to be more active and to minimize customer churning. This way the company's profit is maximized.

Therefore, a smart recommender system is very critical to the company's business strategy. A successful recommender system should be able to accurately predict a user's interest on an item he or she has never used, so that the recommended items will most likely be selected by the user. That could be quite challenging, because the user's interest might change. For example, when a female user is a teen ager, she might love romantic movies. But when she grows older, she might switch to family comedy movies. So from the past data one may not always predict the future successfully.

2. Dataset Used in This Study

A typical dataset used to build a recommender system has at least three columns (features): user id, item id, and the rating to the item by the user. Many algorithms require the dataset to be in this format.

The dataset for this study is from the GroupLens. GroupLens Research is a social computing research lab at the University of Minnesota. In the year of 1997, they released the MovieLens project and started to collect ratings on movies from various users all over the world. They provide movie rating files with different data size. For example, the smallest dataset is the MovieLens 100k, which includes 100,000 movie ratings from 1000 users on 1700 movies. They also have datasets of MovieLens 1M, MovieLens 10M, MovieLens 20M, and MovieLens 25M. In each dataset, there are at least three columns, which are user id, movie id, and the rating value ranging from 0 to 5 (actually the minimum rating you can give is 0.5). There are also different features in different dataset. For example, in the 100k and 1M datasets, you could find demographic information about users such as age, gender, occupation, and zip code, while in other datasets there is no such information.

The MovieLens datasets have been widely studied and used to build different recommender systems. In this study, we will use the MovieLens 20M dataset. It contains 20,000,263 ratings and 465,564 tag applications across 26,744 movies. These data were created by 138,493 users between January 09, 1995 and March 31, 2015. This dataset was generated on March 31, 2015, and updated on October 17, 2016.

There are six data files in the MovieLens 20M dataset, which are “ratings.csv”, “tags.csv”, “movies.csv”, “links.csv”, “genome-scores.csv”, and “genome-tags.csv”. For details of those files and the format of each file, please visit: https://files.grouplens.org/datasets/movielens/ml-20m-README.html.

3. Brief Introduction of Algorithms for Recommender Systems

For the recommender system, two filtering algorithms are commonly used. One is the collaborative filtering, and the other is the content based filtering.

3.1 Collaborative Filtering

The insight in the collaborative filtering is that people’s preference is correlated. In reality you always see some groups of people who share common interests. Some love SciFi movies, others love family movies. Basically there are two types collaborative filtering: user based and item based. In the user based collaborative filtering, it is believed that people who gave similar ratings in the past would give similar ratings in the future. So you can use similar people’s rating to predict; For item based collaborative filtering, you will compare items the specific user has rated on, and choose the item which is similar to the one you need to predict. The advantage for item based collaborative filtering is that usually the amount of users is much larger than the amount of items, so it is more time consuming to compare users; And ratings on item is often very stable, while users are more “waving”.

In collaborative filtering, we don’t use any information on users (age, gender, occupation, location, etc) or items (producer, type, genres, author, etc). Typically we only need three columns on the data: user id, item id, and rating.

3.2 Content Based Filtering

In content based filtering, only items similar to the user’s preference are recommended. If the record shows that the user watched a lot of romantic movies, the system will recommend bunch

of other romantic movies to him/her. Items recommended are always in the “comfort zone”. For content based filtering, the data should include many features on the item.

3.3 Comparison on Two Filtering Algorithms

In collaborative filtering, you don’t need to have much features on items. That can be a big advantage, because in many cases features of the item might be missing. For example, grocery items often do not provide much information on nutrition in them.

Content based filtering is more successful when most users have pure favorite. But in real world, most people have mixed interests. And someone likes something in the past doesn’t always mean he will like them in the future.

The collaborative filtering is powerful to find patterns between different types of items. For example, it is common to see that people who love SciFi movies will also love fantasy movies, although there is no close connection between these two movie types.

For new movies, there are no much information of rating history to use. Therefore it is hard to include them in the collaborative filtering. But if we know many of their features, we could easily add them to a content based filtering algorithm.

It is also possible to create a hybrid model by using both the collaborative filtering and the content based filtering.

4. Brief Introduction of the “Surprise” Module Package

Surprise is a Python scikit for building and analyzing recommender systems that deal with explicit rating data. The name "SurPRISE" is an abbreviation for the Simple Python RecommendatIon System Engine. It provides various ready-to-use prediction algorithms such as baseline algorithms, neighborhood methods, matrix factorization-based (SVD, PMF, SVD++, NMF), and many others. Also, various similarity measures (cosine, MSD, pearson…) are built-in. There are also built-in datasets such as Movielens and Jester. Users can also use their own custom datasets easily. If your data is in the format of Pandas data frame, Surprise provides easy-to-use tools to convert data frame to the format Surprise accepts.

Developers of Surprise also did a good job in documentation. They have tried to make as clear and precise as possible by pointing out every detail of the algorithms. You can find full description of Surprise at Surprise · A Python scikit for recommender systems. (surpriselib.com).

A few models will be tested and compared, and the model which gives the highest accuracy will be selected as the final model.

5. Main Steps in Building the Recommender System

In my understanding, the two main tasks for a recommender system to do is 1) to predict the rating value, and 2) to recommend items to users. In this study we will build a recommender system to complete these two tasks. Below are brief introductions of steps of what has been completed in this analysis.

5.1 Data Wrangling and Exploratory Data Analysis (EDA)

5.1.1 Facts about the rating data

By exploring the data, we found a few facts about it:

- The rating matrix sparsity is **5.4%;**
- The average rating for all movies from all users is **3.5**;
- On average, each user gives **144** ratings;
- On average, each movie has **748** ratings;
- The most rated movie is **Pulp Fiction (1994)**. It has **67,310** ratings.

5.1.2 Reduce Data Size and Export the Data

The rating data has 20,000,263 ratings, which is too many for the analysis in a personal computer. Therefore we need to reduce the data size. The ideal size is about 500,000 rows, which is about 2.5% of the original size.

Steps to select small part of the ratings:

- Step 1: Select ratings of movies in the top 20 genres. Size 200M → 9.2M;
- Step 2: Select ratings from users who give at least 100 ratings. Size 9.2M → 5.5M;
- Step 3: Select ratings from movies which has at least 100 ratings. Size 5.5 → 5.3M;
- Step 4: Randomly select 10% of all users and select only ratings from those random users. Size 5.3M → 0.5M.

So the final rating file has about 0.5 million ratings. We save these ratings in a new file for further study.

5.2 Preprocessing

The main purpose in this part is to find an algorithm which gives the best performance in the rating prediction. To do this, we would like to try on a few algorithms and use Root Mean Square Error (RMSE) as the metric to evaluate the prediction accuracy.

5.2.1 Prediction Models from Existing Module

Most algorithms are from the module package Surprise, except for the Mean Combination which was created by me. The Mean Combination is just a combinations of the movie rating average and user rating average. In equation it is:

Rating prediction on a (user, movie) pair = average rating to the movie * ratio + average rating by the user * (1-ratio).

By doing a 1-D grid search on the ratio value, it is found that the RMSE is smallest when ratio = 0.55. So we use ratio = 0.55 in the rating prediction.

Fig.1 shows the RMSEs from different algorithms. It is pretty clear that the SVD++ (Singular Value Decomposition plus plus) algorithm gives a best performance (lowest RMSE). Therefore we will use the SVD++ as our final algorithm for the modeling process.

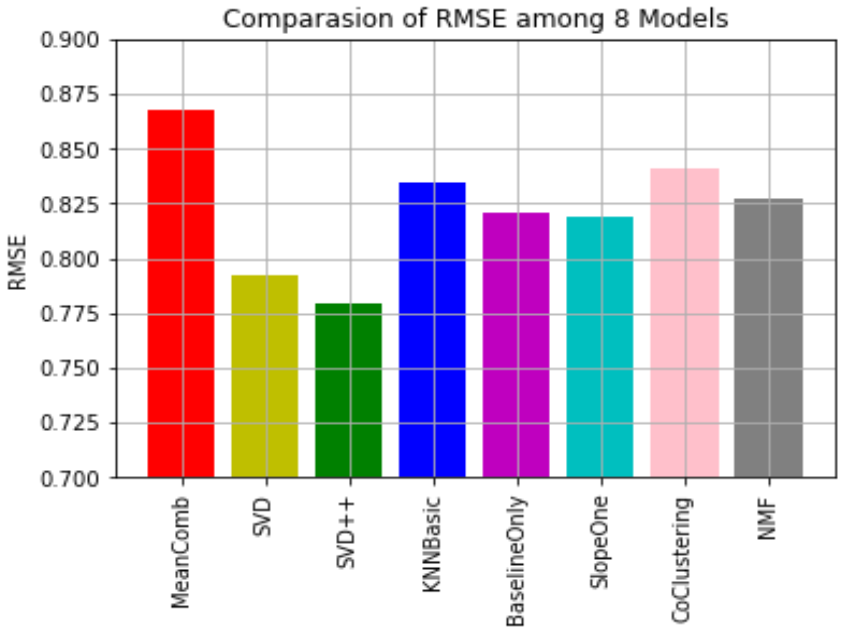

Fig.1 Comparison of RMSEs among 8 Models

5.2.2 Prediction Models from Collaborative Filtering and Content Based Filtering

In part 3 we introduced the collaborative filtering (user based and item based) and content based filtering. And we would like to develop our own codes to see what are the performances of these filtering algorithms.

Fig.2 shows the RMSEs of filtering algorithms, compared with the RMSE from the MeanCombination algorithm. Actually they are not pure filtering algorithm. We only use the prediction from the algorithm if the similarity is big enough (>=0.85). Otherwise we just used the MeanCombination as predictions. In the two collaborative filtering, the Pearson correction coefficients are used as similarity index, and in the content based filtering, the cosine similarity is used as the index.

In Fig.2 we can see that the performances of filtering algorithms are not as good as the existing models in Surprise. So we will not use them in our final model.

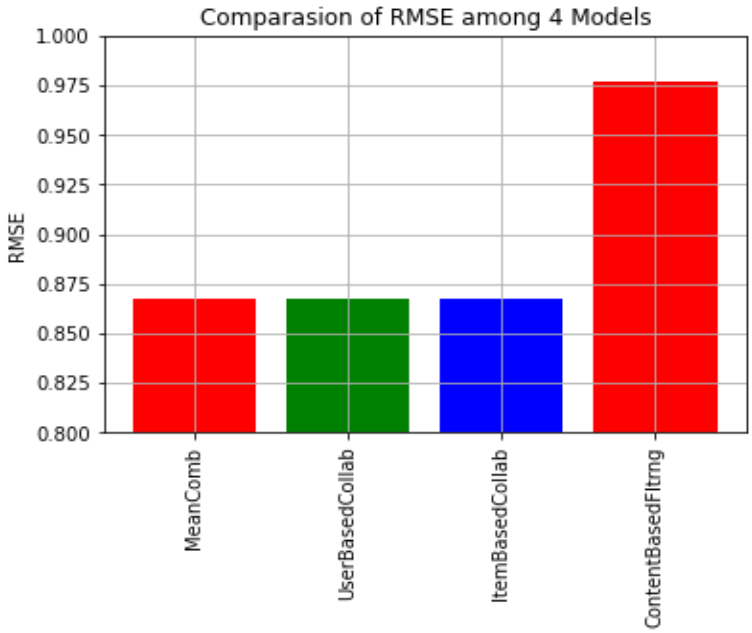

Fig.2 Comparison of RMSEs among Filtering Models

5.3 Final Modeling

The main tasks in the final modeling is to predict users rating and to recommend meaningful items to our users. So let's complete them one by one.

5.3.1 Predicting Users' Rating

To minimize the predicting error, we could remove those low value ratings. So only ratings with value > = 3.5 are kept. Rating data was split into train set and test set. The final SVD++ model gives the RMSE as 0.43.

5.3.2 Recommend Items to Users

How should we recommend items to users? There are different ways to do it. Let's discuss them one by one.

5.3.2.1 Collaborative Filtering Recommender

This recommender just recommends items with high predicted rating. For every (user, movie) pair in the test set, the SVD++ model could predict the rating. This recommender will then recommend certain amount of items (in this case it is 10) with highest predicted rating. One should be noted that these predicted ratings are only for pairs with actual ratings in the test set.

5.3.2.2 Popularity Recommender

This recommender recommends movies with the highest ratings to all users in the test set. It does not consider the user's specific favorites.

5.3.2.3 Random Recommender

This recommender randomly selects items from the rating file and recommend them to each user in the test set. It also does not consider the user's specific favorites.

5.3.2.4 Parallel Recommender

The cover picture in this report is about a shopping lady who read a sign of "customers who bought this item also bought…". This gives us hints that we could also build a recommender in this way. Here are steps to build the parallel recommender:

- Step 1: for each user in the test set, find 5 movies he/she gave highest ratings;
- Step 2: find the total rating on these 5 movies by each user (other than the user in step 1) in the rating file, and list top 10 users who give the highest total ratings on these 5 movies.
- Step 3: for each of the 10 users in Step 2, find one movie he/she rated highest. This movie should not be in the 5 movies in Step 1. So we get 10 movies from these 10 users.
- Step 4: recommend these 10 movies to the original user in Step 1.

5.3.2.5 Unrated Recommender

Typically a recommender should recommend items a user has not used. We can use the SVD++ algorithm to predict the ratings of movies a user has not watched, and recommend those with high predicted ratings to the user.

Here are steps for building an unrated recommender.

- Step 1: For each user in the test set, find all movies he/she has not rated.
- Step 2: Create a dataset for this user. There should be 3 columns in this dataset. The first column should be purely the user id, and the second column is the list of the unrated movies created in Step 1. The third column is the "actual" rating. Since all movies in this dataset are those the user has not rated, there is actually no "actual" rating. But if we don't have this column, the SVD++ algorithm is not able to proceed. So let's just set all "actual" rating to be 0.0.
- Step 3: Apply the dataset created in Step 2 to the SVD++, and we will get a prediction matrix which includes the three columns described in Step 2, as well as a new column with predicted ratings for each row. Sort the prediction matrix by the value of the predicted rating, and listed top 10 movies with highest predicted ratings.
- Step 4: Recommend these 10 movies to the original user in step 1.

I really like this recommender, because it has good usage of our efficient algorithm, and it also works the way a recommender should work: recommend items a user has not used but are likely interested to use.

6. Recommenders Evaluation and Conclusions

The "recmetrics" is a powerful module for the recommender evaluations. You can find details of this module in the author's Github (statisticianinstilettos/recmetrics: A library of metrics for evaluating recommender systems (github.com)). She also has a media article which has more readable description of the module (Evaluation Metrics for Recommender Systems | by Claire Longo | Towards Data Science). We will use a few functions of the module for this study.

6.1 Long Tail Plot

Let's make a plot to show the amount of movie ids vs the amount of ratings. Typically, in a rating dataset, only a small part of items have a high volume of interactions, and this is referred to as

the “head”. Most items are in the “long tail”. They only make up a small percentage of interactions. That’s why this plot is called “long tail plot”, which is shown in Fig.3.

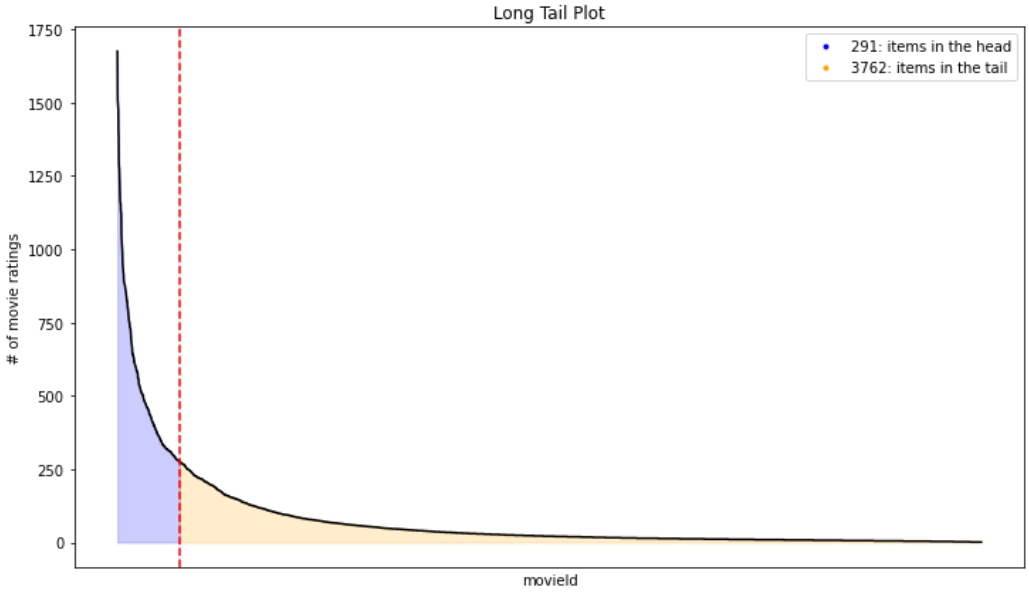

Fig.3 Long Tail Plot

6.2 The Mean Average Recall at K (MAR@K) Plot

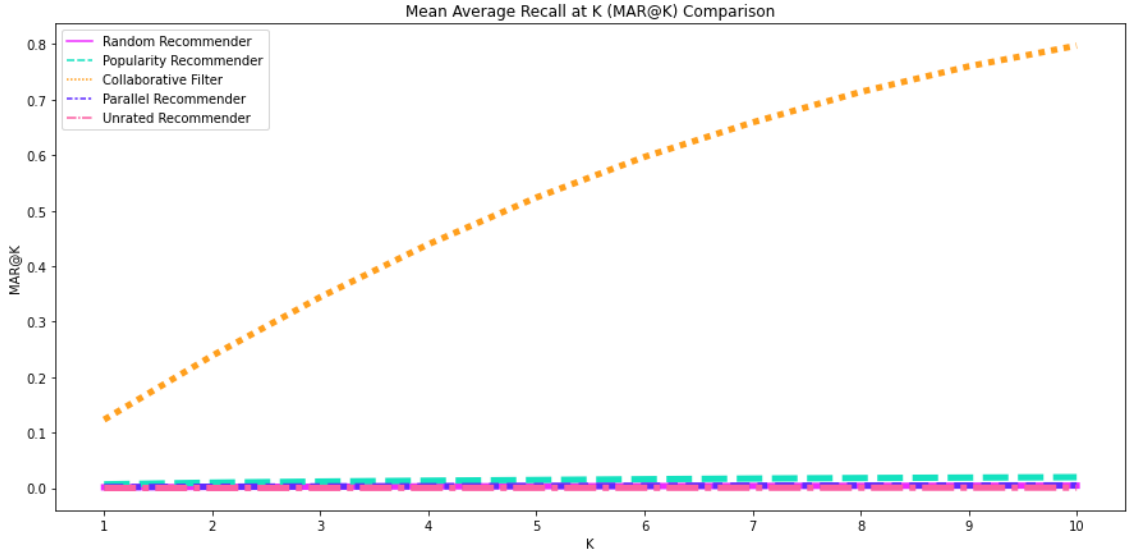

Fig.4 MAR@K plot

The plot of mean average recall at K (Fig.4) shows the amount of “relevant” items in the k recommended items. A relevant item is defined as an item which has a rating >=3.5 by the user. In Fig.4 we can see that the line for the collaborative filtering recommender has a much bigger slope than all other recommenders. That is because this recommender only searches in movies

rated by the user, and recommend those with high predicted ratings. Other recommenders search in the whole rating file, and recommend users with items selected with different criteria.

6.3 Conclusions

The recmetrics use the MAR@K plot as the metrics to evaluate the recommender performance. But I have a big question mark for this metrics. As described above, the mean average recall at K shows how many relevant items has been recommended to the user. We can see that the collaborative filtering recommender gives high recall value. This is because it only look at movies a user has already rated, and recommend movies with high predicted rating to the user. Since the algorithm has already been proved to be effective in predicting the rating (RMSE=0.43), we can imagine that the actual rating overlaps largely with the predicted rating. This means that the recommender basically just recommends movies with high rating by the user, to exactly the same user. This is not the function a successful recommender should do.

Actually I do prefer the unrated recommender. This is because the recommender will first use the prediction algorithm to predict ratings of movies a user has not rated, and recommend movies with high predicted ratings to this user. Since the algorithm has proved to be effective in predicting ratings, a movie with high predicted rating is very likely loved by the user.

The parallel recommender is my second favorite recommender, because it finds other users who give high ratings on movies a user rated high, and recommend this user with other movies those users rated high. This strategy is similar to the user based collaborative filtering.

6.4 Summaries

1. The SVD++ is the best model in predicting users' ratings in this study.

2. It is **doubtful** that the **MAR@K is an effective metrics** to evaluate the performance of a recommender system;

3. The **unrated recommender** is believed to be the **best** recommender in all;

4. List of remaining recommenders from my **best favorite** to **least favorite**: **parallel recommender, popularity recommender, collaborative filtering recommender, and random recommender**.

6.5 Further Studies to be Done

How can we know if our recommender system work well or not? It is always challenging. In reality we can see the percentage of items a user finally choose among all recommended items. For example, Amazon recommends 10 items to a user. If the user finally view or buy 5 of them, it is verified that the Amazon recommender works well. But in this study we only have the rating file. So it is not easy to verify our recommender. If given more time I could explore more to find an effective way to evaluate our system.

Our best performed algorithm gives an accuracy of RMSE = 0.42. Is there any better algorithm? We don't know. And I believe that worth some further algorithm searching.

THE END