ID2221 Data-Intensive Computing Project Report

Shaochen Bai and Yuxin Sun

October 30, 2022

1 Introduction

In the movie industry, online rating websites have become a popular way of receiving audience feedback and gathering opinions. Millions of users post their scores regarding the movie's quality on a scale of 1 to 5, and the collected scores can further be used for analysis and recommendation. With access to each user's history rating data, it is possible to develop a personalized recommendation system aiming to suit the need of each individual.

In this project, we build a prediction model that generates user-specified movie recommendations. The model is based on collaborative filtering method trained on the rating history of users. In addition to the traditional settings, we make further improvements to expand the recommendation system on genre metadata as well. Also, the training data is preprocessed to exclude biased entries from aggressive users. Our contribution could be summarized as:

- We implement a movie recommendation system based on collaborative filtering.
- Further effort is made to expand the recommendation system on genre metadata.
- We adopt filtering on the biased data to improve the performance.

2 Methods and Setup

2.1 Recommendation Model

Alternate Least Squares (ASL)[1] is a recommendation system for decomposing a sparse matrix and evaluating the values of missing items to obtain a basic training model. ASL is developed on the basis of the least squares method. Suppose we have a batch of movie rating user data, which contains m Users and n Items (Movies), then we define a Rating matrix, the elements of which represent the u_{th} User's rating for the i_{th} Item (Movie). Moreover, it is impossible for a user to rate all items, so the Rating matrix is destined to be a sparse matrix. The missing ratings in the matrix are also called missing items.

Spark uses ALS to solve matrix factorization problems in two cases: the dataset is explicit feedback and the dataset is implicit feedback. Recommendation systems rely on different types of input data, the most convenient being high-quality explicit feedback data, which contain explicit user evaluations of items of interest. For example, the review data for movies in movie rating websites, so we take the case of explicit feedback data.

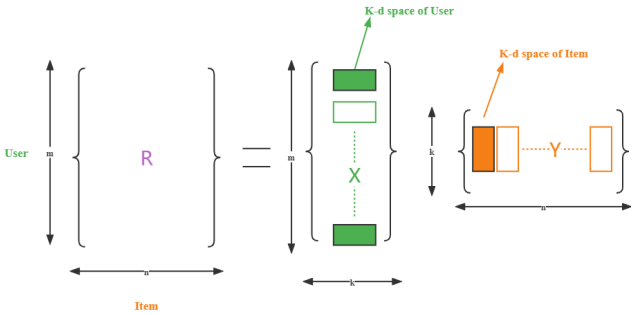

Figure 1: FunkSVD of the ALS algorithm process

In view of such characteristics, we can assume that there are several associated dimensions (such as movie genres, keywords, etc.) between User and Item (Movie), and we only need to project the Rating matrix onto these k dimensions. The mathematical representation of this project is shown as: $R_{m\times n} \approx X_{m\times k} Y^T_{n\times k}$ in Fig. 1. In general, the value of k is much smaller than the values of n and m, thus achieving the purpose of data dimensionality reduction. This method is called the algorithm probabilistic matrix factorization (PMF) algorithm. The ALS algorithm is the application of PMF in numerical calculation.

2.2 Evaluation Metrics

We divide both the dataset into training and test sets to measure the model performance. We utilize root mean squared error (RMSE), one of the most commonly used metrics for regression evaluations, to compare the model predictions with the actual ratings of users in the test set.

2.3 Data

The dataset is available on Kaggle called *The Movie Dataset*[2], which consists of the metadata with movie titles, keywords, and audience ratings from *GroupLens* and *TMDB*. We will learn the movie interest of users based on this dataset, which includes the 26 million ratings from 270,000 users for all 45,000 movies. The total dataset size is more than $500MB$, so concurrent processing on large data is necessary. From the entire dataset, we mainly choose a subset of the files to build the project, which includes:

1. *ratings.csv* is the whole rating dataset for the recommendation system.

2. *ratings_small.csv* is a smaller subset of *ratings.csv* which we use for the test set.

3. *movies_metadata.csv* is a metadata table containing information for the movies appeared in the rating files. In this project, we mainly focus on its key attributes like $genres, titles, etc.$

4. *links.csv* is a intermediate table that connects the movieId in *ratings.csv* with the tmdbId in *movies_metadata.csv*.

5. *links_small.csv* is the corresponding intermediate table over *ratings_small.csv*.

2.4 Platform

The environment we use is Databricks Community Edition, with a specific cluster version: DBR 10.4 LTS ML, Spark 3.2.1, Scala 2.12. Databricks Community is a free version of Databricks' cloud-based big data platform, which not only provides a Notebook workspace, but also a cloud data storage system. In Notebook, it supports multiple programming languages, and the console automatically outputs useful information such as DataFrame headers. Most importantly, it supports Spark MLlib, which allows us to run machine learning data science analytics in the cloud.

3 Experiments and Results

3.1 Data Preprocessing

Filtering To improve the data quality for model training, we intend to filter biased rating data in the beginning. More specifically, biased ratings refer to unfair ratings against the actual movie quality which often appear along with aggressive users. We first visualize the average rating and its corresponding user number histogram in Fig. 2, and find that a small proportion of users tend to over-rate or under-rate the movies (with its average rating above 4.5 or below 1.5). We call these users aggressive users and further discard their ratings in the training procedure.

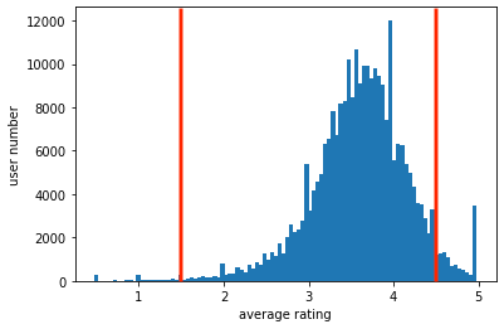

Figure 2: Average rating distribution per user

In the experiment, there are 270246 users in the training dataset, among which 15078 are categorized as aggressive users. After the filtering, the number of remaining users in the training dataset is 255168, which are considered valuable in the model training.

3.2 Parsing, Joining and Transforming

In the experiment, we first parse the *genres* column in the *movies_metadata.csv*. The procedures include regular expression, split, and explode operations. The extracted information from *movies_metadata.csv* is named *metadata_parse_df*. Then, since the *id* column in *movies_metadata.csv* and *ratings.csv* does not match, we need a further linking procedure to join *metadata_parse_df* with *links.csv* and adopt the correctly matched *id* column. The procedure is illustrated in Fig. 3.

Then to fit the genre data into collaborative filtering models, we need to connect the genre metadata with the rating data. We adopt a connecting technique that firstly joins *movie_id* in *movie_genre_df* with *movie_id* in *ratings_df*. Then we group the rating value based on $(userId, genreId)$ by averaging over the rating value. To this end, for every individual user, one average rating value will be assigned to a specific genre. The procedure is shown as Fig. 4

3.3 Model Training and Evaluation

We further split both the genre rating dataset and movie rating dataset into the train and test set. The split is based on the *ratings_small.csv*. Details of the model and evaluation metrics are described in Section 2.1 and 2.2. The results are shown in Table 1. The results show a clear advantage of applying data filtering in the recommendation system.

Table 1: RMSE loss for movie and genre recommendation models. w/o f indicates the model variant without data filtering.

	movie	genre
model w/o f	1.401	1.229
model	**1.379**	**1.144**

3.4 Prediction and Output Formatting

For prediction, we first generate the top 10 recommendations for both genre types and movies with the trained model. Then we merge the two recommendation tables by joining the two rating scores together. It is noteworthy that scaling on two recommendation scores separately is necessary since we want to assign equal weights to both systems. The final output of the model is shown in Fig. 5.

4 Discussion and Conclusion

In this project, we design and implement a move recommendation system based on collaborative filtering. We adopt both genre-based and movie-based recommendation mechanism and further utilizes data filtering to improve the performance. However, the project also has some limitations: **1)** The overall efficiency of the model is not enough, so it is difficult to deploy the model on streaming data. **2)** The metadata utilization is not enough, further improvement can be done by importing metadata information like $vote, vote\_average, popularity, etc.$

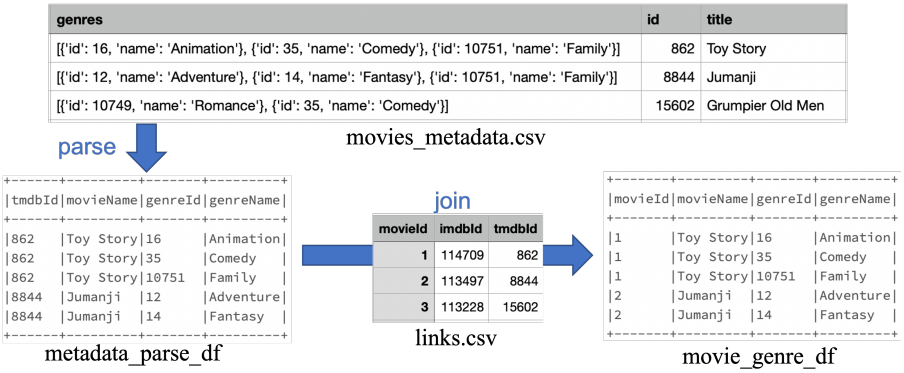

Figure 3: Parse, join procedure

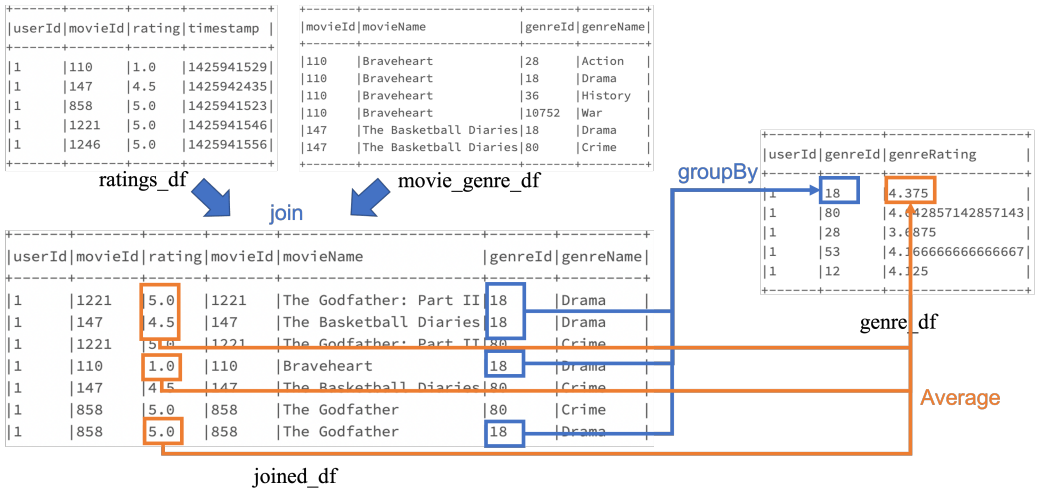

Figure 4: Join, aggregating procedure

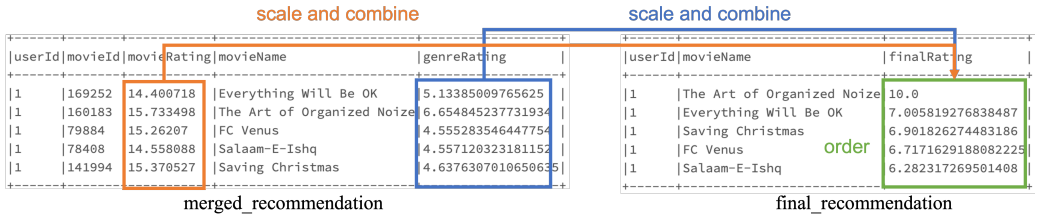

Figure 5: Final output

5 Running Guide

5.1 Preparing

1. Create or use an environment that supports Spark≥2.0, python=3.7, and MLlib API. Especially recommend Databricks;

2. Download the Movies Dataset from Kaggle. Extra processing should be done on $movies_metadata.csv$ to eliminate multiple lines for a single record.

5.2 Usage

The *ID2221_Project.html* is the print script of the notebook with command outputs to train an ALS-movie recommendation system for the desired dataset and compute RMSE for selected datasets. It can be imported into a Databricks workspace or viewed as HTML.

The same content file *ID2221_Project.dbc* is a Databricks archive, which is a binary format that includes metadata and notebook command outputs and it can be imported into a Databricks workspace directly.

We comment on some of the codes for visualization or analysis in commands 4, 5, and 6 of the notebook. This should not prevent you from running the entire pipeline. To reproduce the results for the ASL-movie recommendation system, you simply need to run the notebook code block one by one. If it is in need to save or read the model, modify and run the command 12.

References

[1] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, p. 30–37, aug 2009.

[2] https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset.