Personalization Project 1 Report

1. Introduction

This is the report of *IEOR E4571: Personalization: Theory & Application* Project 1 created by:

- Ying Du (yd2519)
- Chenlu Jia (cj2616)
- Chengyou Ju (cj2624)
- Jiaqi Tang (jt3169)

In this project, we aim to build a relatively efficient and accurate movie recommendation system. In general, we will recommend new movies to a user based on the movies he/she has already watched and rated.

The dataset for our project is the MovieLens 20M Dataset (http://grouplens.org/datasets/movielens/), which contains 20 million ratings and 465,000 tag applications applied to 27,000 movies by 138,000 users.

This report provides general ideas and conlcusions about our projects. More detailed explanations and step-by-step results and comparisons can be found in **Project_1_KNN_SVD.ipynb** and **Project_1_ALS.ipynb**.

---

2. Collaborative Filtering Algorithms

We utilize three collaborative filtering algorithms to recommend, including one neighborhood-based algorithm and two model-based algorithms.

After checking the dataset, we notice that only a few movies have a large number of ratings, and only a few users rate a large number of movies. Therefore, before trying different models, we make some tweaks to the dataset so that it can better serve our project. We focus on movies that are rated by multiple users and users who make multiple ratings. In other words, we only care about popular movies and active users.

We fit our model to four dataset with different sizes sampled from the original large dataset. The four sampled datset includes 1% popular movies& 1% active users, 1% popular movies& 2% active users, 1% popular movies& 3% active users and 2% popular movies& 3% active users accordingly.

2.1 Neighborhood-based

We choose the k-nearest neighbors algorithm, and specifically, we utilize **NearestNeighbors** from **sklearn**.

In this method, for a given user, we make recommendations based on each of the movies he/she has rated. We also determine the number of recommmended movies for each movie he/she has rated and pass it in as a parameter. As a result, we can get all the recommended movies represented in their IDs, as well as the distance between this movie and the original movie that the user has rated.

When uisng this method, we transform our dataframe into a matrix form with userIds as rows, movieIds as columns and ratings of corresponding user and movie as entries. We fill the unknown rating entry with 0's. When making predictions, we first fill out the entrie matrix based on the given entries. After getting all entries, we find the 10 movies with the smallest distances to the given movie.

2.2 Model-based

We utilize two model-based algorithms towards our dataset, which are **SVD** from **Surprise** and **ALS** from **Spark ML**. We decided to try these two algorithms since we wanted to figure which one performs better in our case.

The general idea is that for a given user, predict the ratings he/she will give to all the movies in the dataset that he/she has not watched yet, and recommend the top ten rated movies to the user.

The metrics we care about are **RMSE** (Root Mean Square Error) and **MAE** (Mean Average Error), and after testing using cross validation, we find that all of these models give us relatively small error scores.

---

3. Evaluation Methods

3.1 Evalution of k-nearest neighbors

For the testing of k-nearest neighbors, we refered to the paper posted on Piazza (https://piazza.com/class_profile/get_resource/jzeg1fosono5to/k0fp5mg47zt4qi).

We calculate the score using the following equation:

$$p_{u,i} = \frac{\sum_{j \in S} s(i,j) r_{u,j}}{\sum_{j \in S} s(i,j)}$$

We want to get the score of user u and each movie i that is recommended to him/her, which is represented by $p_{u,i}$. Here $s(i,j)$ represents the distance (similarity) between movie i and movie j, and $r_{u,j}$ represents user u's rating on movie j.

Also note that in order to get the best results, we only use the movies that are rated greater than or equal to 4.0 for the recommendation, since it is more likely for a user to like another movie related to a movie he/she likes.

We notice that k-nearest neighbors is not an efficient way for movie recommendation, as the size of the dataset increases, the run-time of this method increases fast.

The k-nearest neighbors also has another drawback as the data coverage when making the predictions is relatively low. We notice that the movies recommended are from the same time period as the given movie, which means this method can only make recommendations from a certain part of the dataset instead of the entrie dataset.

3.2 Evaluation of SVD

After generating the smaller datasets, we use cross-validation on the new datasets to get RMSE and MAE scores, which are the two metrics that we care about. Since both of these scores are small, we can use this model and make recommendations.

We also use the GridSearch function from surprise package to find the best-fit parameters to the SVD model. As there are a number of possible values for each parameter, we only tried four possible values for each of the three chosen parameters and found the best combination among them.

When fitting to datasets with different sizes, we notice that as the size of the datset increases, the run-time also increases linearly with respect to the dataset size.

Meanwhile, from the graphs in **Project_1_KNN_SVD.ipynb**, we can see that RMSE and MAE scores remain almost unchanged among our different datasets.

The data coverage for the SVD model is pretty good when making predictions. The recommended movies are from different time periods and different genres, covering a wide range of the dataset.

3.3 Evalution of ALS

We use GridSearch to find the best hyperparameters for our model. Detailed test results can be seen from **Project_1_ALS.ipynb**.

---

4. Obervations and Reflections

After testing the performance of these three models on the selected dataset from the MovieLens dataset, we believe that our models can make relatively good recommendations to users based on the movies they have already rated.

We believe that the SVD model is the best one among the three models that we have tested.

- When making recommendations, it chooses from a wider range of movies comapred to the k-nearest neighbors model.
- It has a relatively small run-time compared to the ALS model and the k-nearest neighbors model as the run-time only increases linearly as the dataset size increases.
- It's easier to train and predict comapred to the the ALS model and the k-nearest neighbors model as it's no need to make transformations of the dataframe beforehand as in the other two models.

There are several points we want to point out when making recommendations:

- When the dataset is very large, it's not a good idea to use the k-nearest neighbors model because of the long run-time.
- When fitting and testing on a smaller sampled dataset from the original dataset, it's very important to choose the sample wisely as the recommended results can be quite different for different samples.
- Before choosing and fitting the models, it's important to filter out the unnecessary information and keep the dataset for training clean.
- The Cold Start Problem. We saw there is 12% of user-movie interactions are zero, even after we filtered out less-known movies and inactive users. This shows that we don't even have sufficient information for the system to make reliable inferences for users or items. In case only a few interactions are available, although a collaborative algorithm will be able to recommend it, the quality of those recommendations will be poor.
- Popularity bias. Think about the previous long-tail skewed distribution of movie rating frequency plot. Some amount of items have an extremely high number of iteractions, while most of the items only very few interactions
- This list of movie recommendations using ALS is highly different from the list from KNN model recommender. Not only it recommends movies outside the similar years, but also recommends movies that are less known. So users won't get bored by getting the same popular movies all the time.