Universitat de Barcelona, Faculty of Mathematics and Computer Science

Fundamentals of Data Science

Course: Recommenders System

Task #2 - Movie Recommender - report

Aleksandra Bednarczuk

1. Introduction

In this report I describe the methodology and results of the project aimed to create and compare two types of movie recommender system - item-based approach and latent factor approach. To access the Python code for this project, please follow this link to my github repository: https://github.com/abednarczuk/MasterUB_MovieRec_task2.

2. Methodology

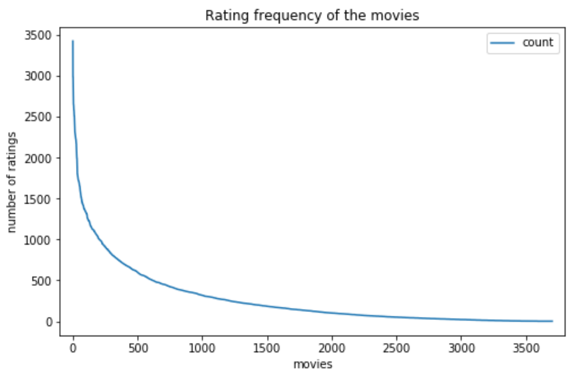
2.1. Data

The project is based on the MovieLens dataset (1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users who joined MovieLens in 2000).

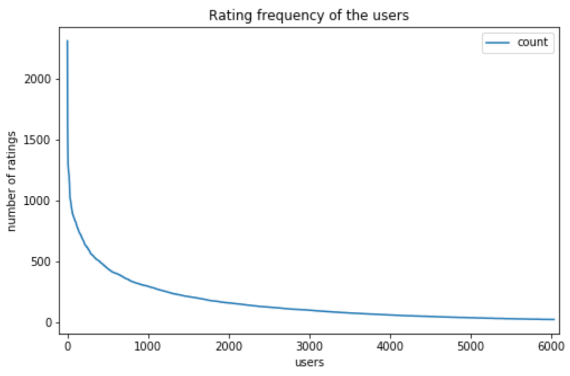
01_MovieRec_EDA.ipynb presents basic exploratory data analysis and addresses the long-tail phenomenon - when the big portion of dataset is of low level of user-interaction. Two aspects of this problem are investigated - movies with small number of ratings (Figure 1a) and users with low level of activity (small number of rated movies) (Figure 1b). To remove the bias introduced by less popular movies and inactive users, the dataset is limited to contain only the records for movies rated at least 100 times (~50% of all movies) and users who rated at least 50 movies (~70% of all users). After that, the dataset gives the information about 2019 movies rated by 4296 users. To lower the computation time of the calculations, the dataset is limited one more time to contain ~50% of the movies (movie_id < 1800) Final dataset is named *dataSmall*, is of shape: (324426, 4) and is divided into train and test datasets with test set size of 33%.

Figure 1. Long-tail problem.

a) Most of the movies has very small number of ratings.



b) Most of the users rate small number of movies.



2.2. Item-based recommender system

For the item-based recommender system, following the suggestions from scientific papers [2] and the course materials [1], *SimAdjCos* function is defined and used to calculate adjusted cosine similarity between items. By subtracting the corresponding user average from item rating, he differences in rating scales between users are taken into account [2], what makes the similarity measure more standard and more accurate.

$$sim(i,j) = \frac{\sum_{u \in U}(R_{u,i} - \bar{R_u})(R_{u,j} - \bar{R_u})}{\sqrt{\sum_{u \in U}(R_{u,i} - \bar{R_u})^2}\sqrt{\sum_{u \in U}(R_{u,j} - \bar{R_u})^2}}.$$

ItemCF class is designed to train the model to perform rating estimation based on item-item recommendation approach. The input dataset (dataSmall_train, shape: (258587, 4), #movies: 784, #users: 4294) is used to create user-item rating matrix. Based on the user-item rating matrix and *SimAdjCos* function, the similarity matrix is created by computing the similarity rate between pairs of movies in the dataset. Next, the estimation for users rating is calculated by computed weighted sum of all similar items.

$$P_{u,i} = \frac{\sum_{\text{all similar items, N}}(s_{i,N} * R_{u,N})}{\sum_{\text{all similar items, N}}(|s_{i,N}|)}$$

2.3. PureSVD

For the factorization model, the approach for PureSVD model from the paper by Cremonesi P. et al [3] is reconstructed. *PureSVD* class uses *randomized_svd* method from sklearn (sklearn.utils.extmath.randomized_svd) to find an approximate truncated singular value decomposition (number of components=150 - following the findings by Cremonesi P. et al [3]) using randomization to speed up the computations[1]. The input dataset is decomposed into **U**, **Σ** and **Q** matrices, which are further used to calculate rating matrix - R = $\mathbf{U}*\mathbf{\Sigma}*\mathbf{Q}^T$. Each row of **U** * **Σ** represents the user factor vectors and each row of **Q** represents the movie factors vector.

3. Results

3.1. Evaluation method

The results of both approaches are evaluated based on the defined *precision_recall* function. The function calculates precision and recall metrics following the instructions from Cremonesi P. et al [3]. To lower the computational intensity of the evaluation process, the metrics are calculated using 1,4% random sample from the training dataset (dataSmall_train). Next, the test set is limited to contain only the records with rating 5.0 (T is the number of rows in this dataset). For each item i rated 5.0 by user u in this dataset, 100 random movies unrated by u are selected. Then, ratings for those 100 movies together with movie i are calculated. Next, the items are ordered to create a top-N ranking (N=20 is used in the calculations). If movie i is included in this ranking, we have a hit. At the end, the overall precision and recall are defined as:

$$\text{recall}(N) = \frac{\#\text{hits}}{|T|}$$

$$\text{precision}(N) = \frac{\#\text{hits}}{N \cdot |T|} = \frac{\text{recall}(N)}{N}$$

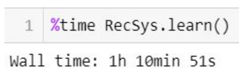
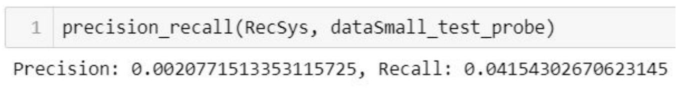
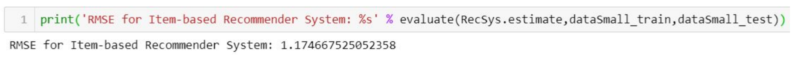
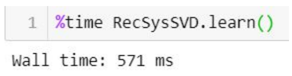
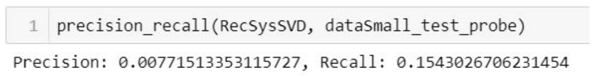
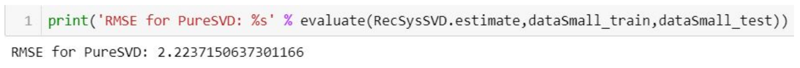
3.2. Results comparison

Table 1 presents the evaluation of two methods described above. First and probably the most important observation is that the *PureSVD* method is much faster than the *ItemCF*. Secondly, *PureSVD* achieves better *precision_recall* results. Lastly, *ItemCF* achieves better results in terms of RMSE. However, when the recommender system is aimed to provide the

[1] https://scikit-learn.org/stable/modules/generated/sklearn.utils.extmath.randomized_svd.html

best top-N recommendations, it is not necessary to accurately predict the ratings of the user - more important is to recommend a good set of new items for this user. That is why using classification rather than regression accuracy metrics gives better information about the accuracy of created recommendation system.

Table 1. Evaluation of ItemCF and PureSVD.

ItemCF	% time	 `%time RecSys.learn()` `Wall time: 1h 10min 51s`
	precision_recall	 `precision_recall(RecSys, dataSmall_test_probe)` `Precision: 0.0020771513353115725, Recall: 0.04154302670623145`
	RMSE	 `print('RMSE for Item-based Recommender System: %s' % evaluate(RecSys.estimate,dataSmall_train,dataSmall_test))` `RMSE for Item-based Recommender System: 1.174667525052358`
PureSVD	% time	 `%time RecSysSVD.learn()` `Wall time: 571 ms`
	precision_recall	 `precision_recall(RecSysSVD, dataSmall_test_probe)` `Precision: 0.00771513353115727, Recall: 0.1543026706231454`
	RMSE	 `print('RMSE for PureSVD: %s' % evaluate(RecSysSVD.estimate,dataSmall_train,dataSmall_test))` `RMSE for PureSVD: 2.2237150637301166`

4. Conclusions

As it was already presented and commented above, the latent factor method outperformed the item-based collaborative filtering in terms of time (and probably also memory) intensity as well as accuracy of estimations. For additional improvements, models could be trained on a bigger sample of the MovieLens dataset. Moreover, different approach for handling long-tail phenomenon and including less popular movies could be applied. To improve the results of the item-based method, ratings estimation could be performed using KNN method for finding the most similar items instead of using all the items already rated by the user. To improve the results of the latent factor method, optimization for randomized_svd could be performed.

5. References

1) Recommenders System course materials (https://github.com/ssegui/recsysMaster2020)
2) Badrul Sarwar, George Karypis, Joseph Konstan, John Riedl, *Item-based Collaborative Filtering Recommendation Algorithms*, 2001 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.167.7612)
3) Paolo Cremonesi, Yehuda Koren, Roberto Turrin, *Performance of Recommender Algorithms on Top-N Recommendation Tasks*, 2010 (https://dl.acm.org/doi/pdf/10.1145/1864708.1864721)
4) https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-1-knn-item-based-collaborative-filtering-637969614ea
5) https://github.com/KevinLiao159/MyDataSciencePortfolio/blob/master/movie_recommender/movie_recommendation_using_KNN.ipynb
6) https://github.com/mesuvash/pyrec/blob/master/pyrec/recommender/bilinear_models/pureSVD.py