Building Recommendation Engine – Collaborative Filtering

1. Collaborative Filtering Introduction

Recommendation systems are used widely nowadays in the e-commerce and content-streaming industries to boost up their sales. It provides a way to retain the customers by recommending them products such as movies, apparel, etc. Collaborative filtering is a technique used to build recommendation system. It uses the knowledge of users' preferences from the past to predict his/her preferences for the future. Collaborative filtering uses the technique of collaborating the ratings of different users and then using that knowledge to filter out the products which a particular user might like.

2. Dataset

The dataset: We have used the dataset for Amazon Movies. The dataset contains the User Id, Item Id, Ratings and Timestamp as tuples. The metadata or reviews data has not been included.

Data sampling: The dataset contains 2088620 unique users and 200940 unique items. The dataset has ratings for 4607046 user-item combination, which shows 99.9% of the user-item combination has no rating at all. We have explicit ratings with scale of 1-5. The sparsity of data is very high so we have taken a small subset of the dataset to build the recommendation engine.
In order to overcome the problem of sparsity, we have first taken out 90 of the items which have been rated most frequently. This gives us 90 unique items and 281592 unique users. This decreases our sparsity to 98.67%. Next thing which we do to reduce sparsity is to take out the first 10000 users who have rated the largest number of items from the previous subset of the data. This will give us 10000 unique users with 90 unique items. This give us the sparsity of 95.58%. Now we have good enough data to train and evaluate our model and hence predict unknown ratings for a user-item combination.

Before we go on with our predictions, since we have to check how good our model performs, we have created a test data from the train dataset. The test data contains few user-item ratings. To check the accuracy of the model, we set all those user-item ratings as null in train dataset. The ratings for those then have to be predicted and compared with the original ratings.

3. Objective

The objective is to predict the user item ratings for all the users in our dataset. We have implemented two brute force collaborative filtering algorithms – user-user neighborhood-based and model-based. The idea is to get the results from these models and then scale the data sample to see how the sparsity may affect the accuracy of the model.

4. Collaborative Filtering Algorithms

a) Neighborhood-based

In this algorithm, the ratings for a user is predicted either by similar users or similar items. Suppose we want to predict the ratings of an item "i" for a user "u". To do this, we can find the users who are similar to user "u" in rating the same items- ex. suppose user "u" rated 5, 3, 2 for items "1", "2", "3" respectively and the users similar to user "u" has also rated the same items with more or less same ratings. Then user "a" is most likely to rate item "i" like other similar users who have already rated the item "i". This is called user-user neighborhood-based model.

There is one more approach for neighborhood-based algorithm which is by item-item similarity. We have implemented the user-user neighborhood-based model for our dataset. For this we have

pre-computed the similarity matrix for all user-user combination. We have used Pearson similarity measure, given as:

$$Pearson(u,v) = \frac{\sum_{k \in I_u \cap I_v}(r_{uk} - \mu_u) * (r_{vk} - \mu_v)}{\sqrt{\sum_{k \in I_u \cap I_v}(r_{uk} - \mu_u)^2}\sqrt{\sum_{k \in I_u \cap I_v}(r_{vk} - \mu_v)^2}}$$

Before doing this, we normalize all the ratings by subtracting it from mean. Now to predict the ratings for all the items for a particular user, we should know the similar users. For this we have used KMeans algorithm, which will give us the nearest k-neighbors to a user. We then take the weighted average of the ratings of all the neighbors of a user for that particular item. To get the raw rating, we add the mean for that user to it. This method is applied to all the users for all its items.

$$\widehat{r_{uj}} = \mu_u + \frac{\sum_{v \in P_u(j)} Pearson(u,v).(r_{vj} - \mu_v)}{\sum_{v \in P_u(j)} |Pearson(u,v)|}$$

Major concerns with this model:

- As the sparsity of the matrix increases, the neighborhood-based model will not be able to find neighbors for the user (i.e. the similar users), hence it won't predict anything and the accuracy of the model will be very low.
- As the dataset size will increase, the running time of the algorithm will also increase. With more users and more items, the algorithm will take up more time to predict each user-item rating. Hence scalability is a major concern in neighborhood-based model.

Hyper-parameters:

We used clustering algorithms to create the peer set for each user. Out of K-means, Agglomerative filtering, DBSCAN, Mean-Shift and Birch clustering, Birch Clustering gave the best results. Silhouette score was used to test performance of each model. Silhouette score is a measure of how similar a data point is to its own cluster as compared to other clusters. Cross-validation with 3-folds and grid search was used to systematically tune the following hyper-parameters.

n_clusters: This parameter is for dividing the dataset into n clusters. The GridSearchCV function in sklearn library is used to systematically test which value of n_clusters maximizes silhouette score.

branching_factor: This parameter is used for maximum number of sub-clusters in each node. The GridSearchCV function in sklearn library is used to systematically test which value of branching_factor maximizes silhouette score.

We got n_clusters = 100 and branching_factor=20 from the grid search.

b) Model-based:

In Model-based algorithm, machine learning techniques are employed to predict the ratings of items for a user. We have used Matrix-factorization algorithm – SVD to reduce the latent space of the dataset. In this method, we create embeddings for users and items based on the important features of items. For users, all those features are embedded which are relevant for a user. Like in movies recommendation engine, users embedding might have genre of the movie, popularity of the movie, etc.

We have implemented SVD using the scipy package because it lets us decide the number of latent factors we want to use for predictions. SVD factors m x n matrix X into three matrices: user-feature matrix - (U) of size m x r, diagonal matrix containing singular values of X - (S) of size r x r. transpose of product feature matrix – (V) of size n x r.

Major concerns with this model:

- Since the whole dataset is not used in matrix factorization as the dimensionality of the latent space is reduced, we cannot guarantee the quality of the recommendation. In our case, this is not the situation but when the scalability is increased, the quality of predictions might become poor.

Hyper-parameters:
Number of latent factors(k): svds from scipy allows to give our own latent factors which is used to approximate the original ratings matrix.

We used the Elbow method by manually trying out different values of number of latent factors and checking which gave best performance on the dataset i.e. Minimum root mean square error in predicting user-item ratings. We got the best result for k=25.

5. Evaluation

a) Accuracy

Accuracy is a good method to check how our model is performing on the test data. It tests our predicted values against the original observed ratings and the according to that give a score how good our model is. Here, we have used two accuracy metrics:

Root Mean Square Error (RMSE):
Let r_{uj} be the predicted rating for user 'u' and item 'j'
Let o_{uj} be the observed rating for user 'u' and item 'j'

$$RMSE = \sqrt{\frac{\sum_{j=1}^{n}(r_{uj} - o_{uj})^2}{n}}$$

where, n = number of ratings in test data.
RMSE can be used to measure the spread of the observed values of ratings about the predicted value of ratings. Lesser the value of RMSE, better the quality of predicted ratings.

Mean Absolute Error (MAE):
Using the same ratings notation:

$$MAE = \frac{\sum_{j=1}^{n} |r_{uj} - o_{uj}|}{n}$$

b) Coverage Ratio

Coverage ratio in recommender system is the percentage of items or users which were successfully recommended by the system. We have used here two types of coverage: Item coverage and Catalogue Coverage.

Item Coverage:

Item Coverage is the ratio of items included in the recommendation list over the number of potential items.

$$Coverage_{item} = \frac{n}{N}$$

Catalog Coverage:
Catalog Coverage is the ratio of recommended user-item pairs over the total number of potential pairs. Let L be the number of recommended user-item pairs.

$$Coverage_{catalog} = \frac{L}{N * U}$$

6. Results

We have 6 different cases of sample size, with which each model has been tested. The results for RMSE, MAE, Catalog Convergence Ratio and Fit time of the model has been formulated in a table and even represented graphically, to compare different models with different parameters.

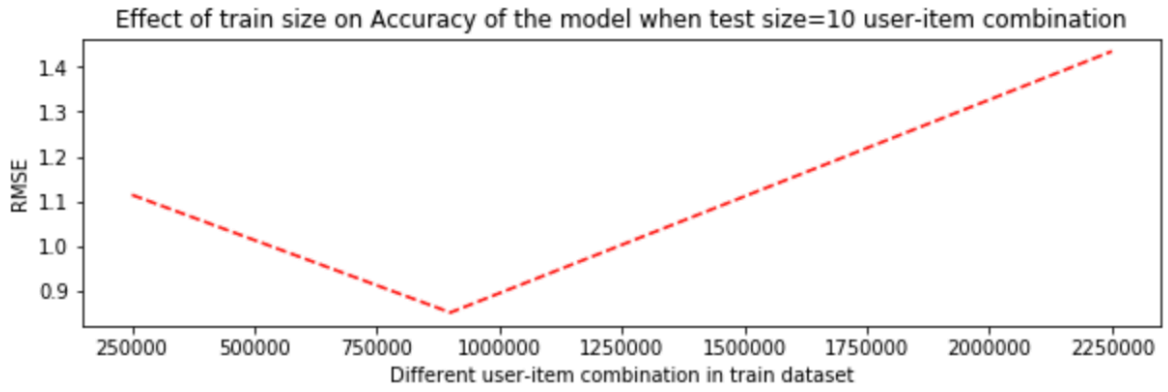
Neighborhood-based CF: Table 1

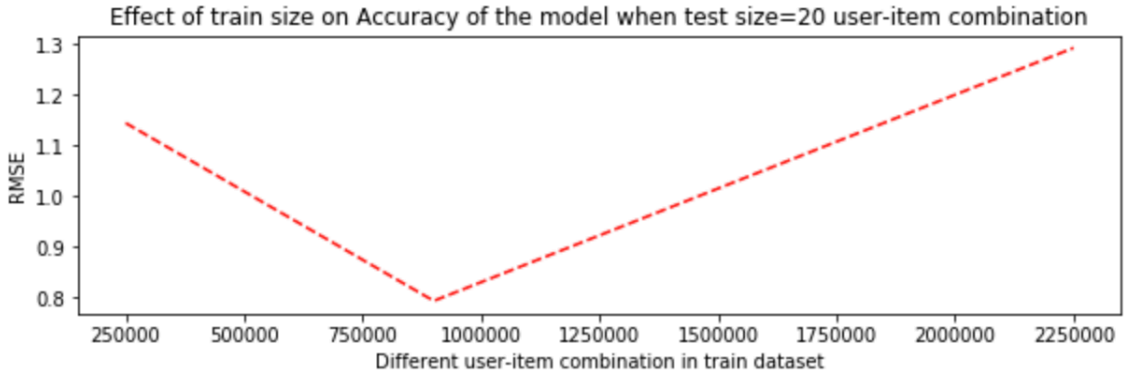
Total user-item pairs (train size)	**User-item pairs tested for accuracy (test size)**	**Root Mean Square Error (RMSE)**	**Mean Absolute Error (MAE)**	**Catalog Convergence Ratio**	**Fit time of the model**
250000 (50 items x 5000 users)	10	1.1140	0.79317	0.0555	11160.0759
250000 (50 items x 5000 users)	20	1.1437	0.7649	0.0555	11789.3045
900000 (90 items x 10000 users)	10	0.85089	0.59076	0.1	1144.3768
900000 (90 items x 10000 users)	20	0.79282	0.51794	0.1	1125.2785
2250000 (150 items x 15000 users)	10	1.43495	0.9984	0.0333	23713.3305
2250000 (150 items x 15000 users)	20	1.2919	0.80763	0.0333	23908.097

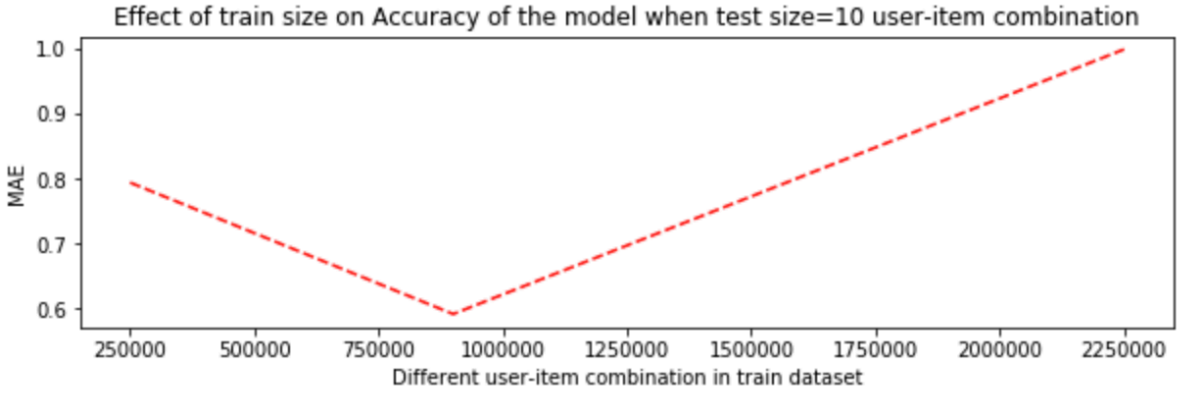
Model-based CF: Table 1

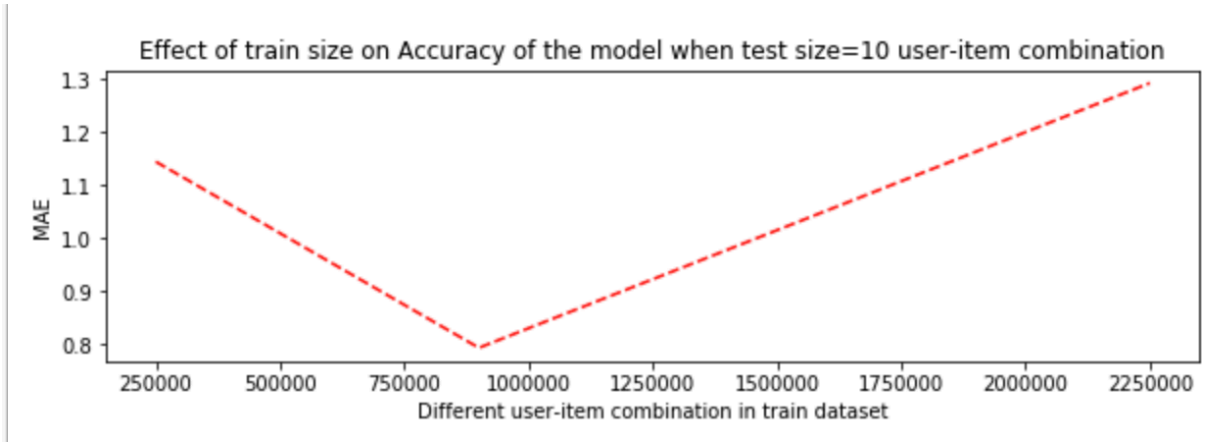
Total user-item pairs (train size)	**User-item pairs tested for accuracy (test size)**	**Root Mean Square Error (RMSE)**	**Mean Absolute Error (MAE)**	**Catalog Convergence Ratio**	**Fit time of the model**
250000 (50 items x 5000 users)	10	0.93247	0.69054	0.0555	338.0911
250000 (50 items x 5000 users)	20	0.77312	0.52085	0.0555	323.7847
900000 (90 items x 10000 users)	10	1.21461	1.02620	0.1	228.2901
900000 (90 items x 10000 users)	20	0.90574	0.38323	0.1	229.3215
2250000 (150 items x 15000 users)	10	0.76758	0.60394	0.0333	410.3723
2250000 (150 items x 15000 users)	20	0.95830	0.70778	0.0333	425.7028

If we plot these values for Neighborhood-based Collaborative Filtering and Model-based Collaborative Filtering, we observe that, the RMSE and MAE follows the same pattern for neighborhood-based model with the accuracy being the best for 90 items x 10000 users (both the test cases).



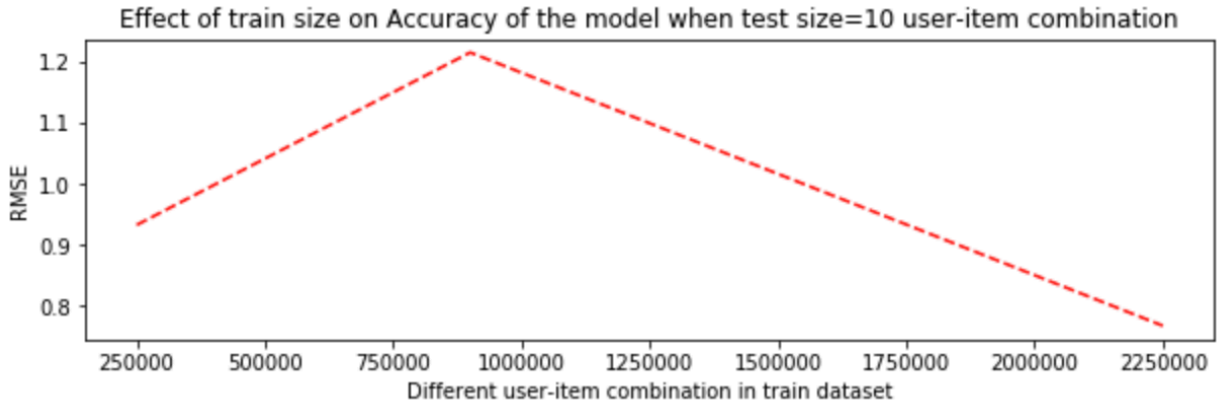


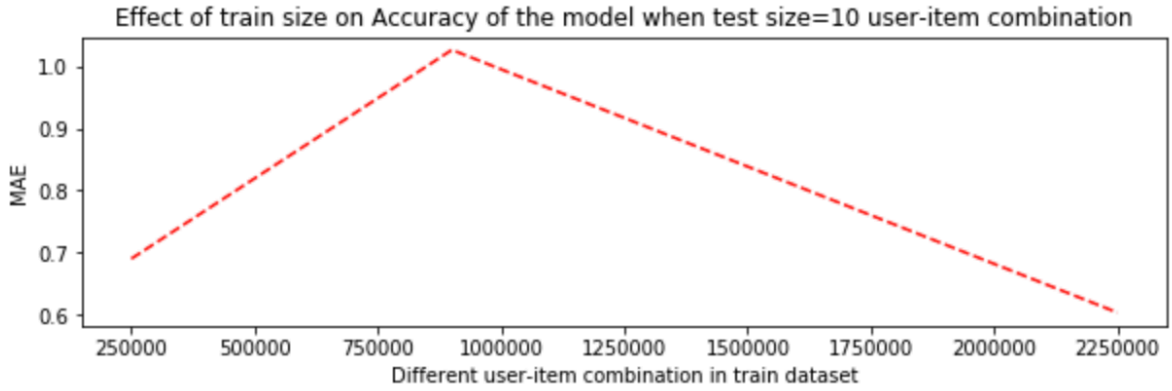




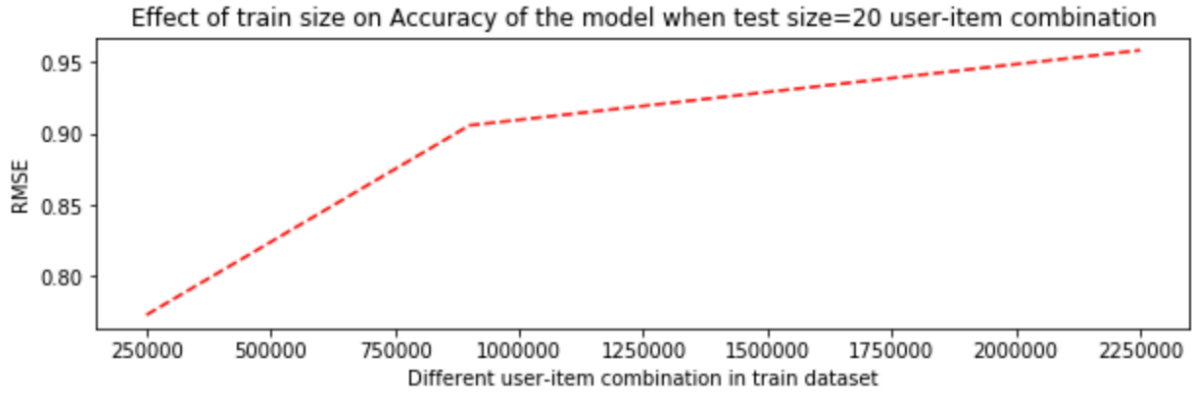
But for Model-based Collaborative filtering, RMSE and MAE both differ for test cases = 10 and 20 user-item combination. For test case = 10 user-item combination, we have accuracy best for 2250000 user-item combination (150 items x 15000 users). For test case = 20 user-item combination, we have accuracy best for 25000 user-item combination (50 items x 500 users).

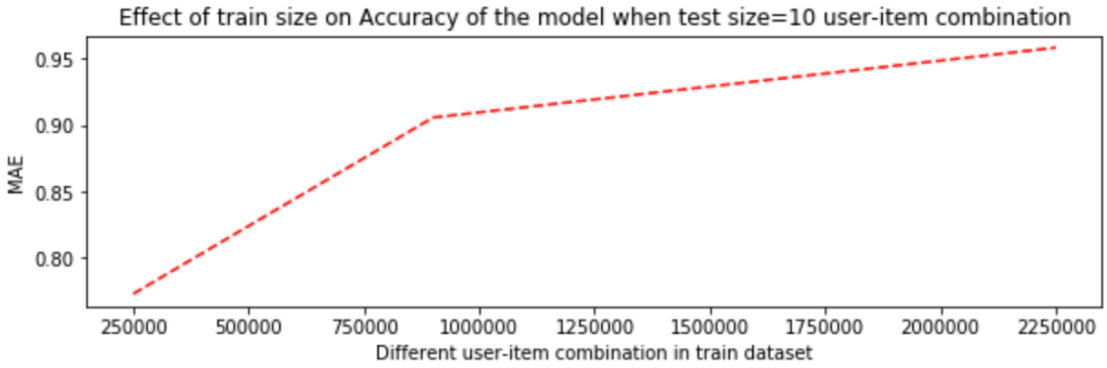
Test Sample = 10:





Test Sample = 20:





7. Conclusion

The pros and cons of the models discussed above are as follows:

a) Neighborhood-based:

Pros:

- We implemented different clustering algorithms (K-means, Birch, Mean-Shift, Agglomerative, DBSCAN) to find the best fit for our dataset.
- Grid Search is used to tune the hyper-parameters for each subset of data for Birch clustering algorithm
- Each user has at least one neighbor and hence we have predicted ratings for each user

Cons:

- As the subset size increases, the fit time of the model increases. Thus, the algorithm is not scalable for large size subsets.

b) Model-based:

Pros:

- The time to fit the model is considerably lesser than neighborhood based-approach and it scales better with increasing subset size as compared to neighborhood-based approach.
- SVD is used to perform dimensionality reduction of user-item matrix and thus is more space efficient.

Cons:

- The model is not flexible because every time we add new data the model has to be fitted again which is very time-consuming

8. Practical results:

- From the graphs above we can observe that we do not have a consistent pattern for accuracy as the subset size changes. Thus, we cannot comment on the accuracy of both Neighborhood-based and Model-based approach.
- We can also observe that fit-time of the model increases for both approaches as the subset size increases. While the model-based approach is more time-efficient than neighborhood-based approach it is still not efficient enough to be implemented in production.
- Both models are inflexible as we have to fit the model again as we add new data which requires a lot of resources.

Thus, neither of the models can be used in production but we can use the model-based approach as a baseline model since it is more time efficient.