Exploring Collaborative Filtering Algorithms using MovieLens data

Shiven Mian* IIIT Delhi
Shantanu Jain* IIIT Delhi
Kushagra Singh* IIIT Delhi
{shiven15094,shantanu14097,kushagra14056}@iiitd.ac.in

Abstract

Recommendation Systems are widely used for predicting ratings of items and their users, and thus for recommending new items that the user is likely to be interested in - making them central to e-commerce today. Movie recommendations are one such popular application of recommendation systems, used for predicting ratings of new movies for users. In this paper, we study, implement and evaluate various approaches for movie recommendation using the MovieLens dataset.

1 Introduction

In this project, we build a movie recommendation system and compare the performance of various memory and model-based approaches. The recommendation problem can be formally described as this:

"Given a user **u** and item **i**, predict how **u** will rate **i**. This rating is denoted by $R_{u,i}$"

We tackle this problem using two kinds of approaches:

1. **Memory-based Approaches:** We use available data to study similarities between users and movie ratings. We study both item-item and user-user similarity models - we use similarity-based metrics for each model and compare our results using cross validation

2. **Model-based Approaches:** We use Machine Learning models for making predictions. We frame the problem as a classification task:

   Task: Predicting rating of movie **m** for user **u**. Possible classes - $1, 2, 3, 4, 5$

   Training labels: Ratings of movies rated by **u**

   Feature Vector: For each label, ratings of that movie by all the users

   Predicted Vector (Test): Ratings of movie **m** by all the users

2 Dataset and Evaluation

We use the MovieLens 100K dataset[1] released by GroupLens Research, to train our models. The dataset consists of 100,000 ratings (1-5) from 943 users on 1682 movies, where each user has rated atleast 20 movies.

Moreover, the MovieLens data is already split into 5 mutually exclusive train-test folds. We shall be using the standard test sets for evaluation. This shall also allow us to compare our results with the state of the art.

We use two metrics, Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) for evaluation of our algorithms. We compare the performance of our models with three baselines, which are

- Global average, which is the average rating across all user-item pairs
- User average. When making a prediction for user u, our prediction is the average of all the ratings made by the user,
- Item average. While making a prediction for an item i, our prediction is the average of all the ratings we have for the item.

Furthermore, we also compare our results with LibRec benchmarks, and find that the performance of our models are comparable to these.

*Denotes equal contribution

[1] https://grouplens.org/datasets/movielens/

Algorithm	RMSE (5-fold)	MAE (5-fold)
LibRec Benchmark – RegSVD	0.939	0.741
LibRec Benchmark – BiasedMF	0.911	0.720
LibRec Benchmark – SVD++	0.908	0.713
Baseline – Global Average	1.2671682	0.9447258
Baseline – User Average	1.0894600	0.8361767
Baseline – Item Average	1.0481092	0.8169033
User - User + Pearson	1.0528357	0.8189285
User - User + Cosine	1.0529436	0.8189220
User - User + Pearson + Bias Removal	0.9331722	0.7619358
User - User + Cosine + Bias Removal	0.9333060	0.7620755
Item - Item + Pearson	1.1116231	0.8460190
Item - Item + Cosine	1.1147506	0.8478902
Item - Item + Pearson + Bias Removal	**0.9108643**	**0.7528672**
Item - Item + Cosine + Bias Removal	0.9232937	0.7558854
Multinomial Naive Bayes	–	0.884
K Nearest Neighbours (K = 10)	–	0.9409
K Nearest Neighbours (K = 15)	–	0.9408
K Nearest Neighbours (K = 20)	–	0.9415
Random Forest	–	0.7886
Logistic Regression	1.12732	0.87744

Table 1: Performance of various algorithms (Root Mean Squared Error and Mean Absolute Error) on the test fold. We report the average across the 5 test folds. Performance of baselines and LibRec benchmarks are also introduced for comparison. Further benchmarks for LibRec can be found at https://www.librec.net/release/v1.3/example.html

3 Similarity Based Models

As a baseline, we attempt to see what will happen in case of a non personalized recommendation system. To do so, We calculate the prediction of an item i for the user u ($R_{u,\,i}$) by simply taking a mean of all the ratings of the item i.

$$R_{u,i} = \frac{\sum_{a \in U} R_{a,i}}{|U|}$$

We extend this baseline by personalizing it for users by incorporating behaviour of other users. This allows us to pay more emphasis on ratings by users that are similar to the user u.

3.1 Exploiting User-User Similarity

The intuition is to give more weight to the ratings by users similar to the user u. We calculate this similarity based on the previous behaviour (ratings for other user-item pairs already available to us). Multiple different metrics can be used to calculate similarity, we use Pearson's coefficient for our work.

3.1.1 Similarity Metrics

Pearson's coefficient is probably the most famous similarity coefficient. It is computed in the following manner.

$$sim(a,b) = \frac{\sum_{p \in P}(r_{a,p} - \overline{r_a})(r_{b,p} - \overline{r_b})}{\sqrt{\sum_{p \in P}(r_{a,p} - \overline{r_a})^2}\sqrt{\sum_{p \in P}(r_{b,p} - \overline{r_b})^2}}$$

Here a and b are the users, P is the set of items which both a and b have rated.

We can also make use of cosine similarity.

$$sim_{cos}(a,b) = \frac{\sum_{i \in I_a \cap I_b} r_{a,i} \cdot r_{b,i}}{\sqrt{|\vec{r_a}|}\sqrt{|\vec{r_b}|}}$$

Here a and b are the users, I_a is the set of items which have been rated by a, I_b is the set of items which have been rated by b. The rating vector comprising of all the ratings by user u is denoted by $\vec{r_u}$.

3.1.2 Making Predictions

The predicted rating is calculated by the following function

$$R_{u,i} = \frac{\sum_{a \in U_{sim}} sim(u,a) * r_{a,i}}{\sum_{a \in U_{sim}} |sim(u,a)|}$$

Which can be interpreted as a weighted sum of other users' ratings of item i.The set U_{sim} is the set of similar users we consider. This can either by the entire set of users, or some top-K similar users.

3.1.3 Removing User Bias

The above method for rating prediction assumes that all users' rating methodology is objective, which might be not true. Each user has his/her own bias, which reflects in the ratings. We can remove this bias by making a small alteration to the prediction function.

$$R_{u,i} = \overline{r_u} + \frac{\sum_{a \in U_{sim}} sim(u,a) * (r_{a,i} - \overline{r_a})}{\sum_{a \in U_{sim}} |sim(u,a)|}$$

To remove this inconsistency in the user behaviour, we compute the mean rating of a user, and subtract this from all the ratings of the user. Of course, we add back the mean rating of the user for whom we are computing the rating.

3.2 Exploiting Item-Item Similarity

In the previous approach we made use of all the users' behaviours, and predicted a user's item for a rating by looking at other similar users. This approach suffers from two big flaws

- As the number of items increase, the set P of common items rated by a pair of users tends to decrease. This makes the correlation coefficient (similarity) unreliable.
- Since the attributes related to a user keep changing frequently, we need to constantly recomputing our model.1

In the item-item similarity approach for calculating $R_{u,i}$, we look at items most similar to i that a user has already rated. This item-item similarity is also based on the ratings of other users (and not physical attributes of items).

This is usually more stable than user-user similarity based techniques, since on average an item has more ratings, compared to the number of ratings by a user.

3.2.1 Calculating Similarity

For item similarity, we can once again make use of Pearson's coefficient or cosine similarity.

The Pearson's similarity coefficient for item-item similarity is defined by

$$sim(i,j) = \frac{\sum_{q \in Q}(r_{q,i} - \overline{r_i})(r_{q,j} - \overline{r_j})}{\sqrt{\sum_{q \in Q}(r_{q,i} - \overline{r_i})^2}\sqrt{\sum_{q \in Q}(r_{q,j} - \overline{r_j})^2}}$$

Here i and j are the items, Q is the set of users which have rated both i and j.

The cosine similarity metric for item-item similarity is defined by

$$sim_{cos}(i,j) = \frac{\sum_{u \in U_i \cap U_j} r_{u,i} \cdot r_{u,j}}{\sqrt{|\vec{r_i}|}\sqrt{|\vec{r_j}|}}$$

Here i and j are the items, U_i is the set of users who have rated i and U_j is the set of users who have rated j. The rating vector comprising of all the ratings for the item i is denoted by $\vec{r_i}$.

3.2.2 Making Predictions

The predicted rating for a user-teim pair is calculated by the following function

$$R_{u,i} = \frac{\sum_{j \in I_u} sim(i,j) * r_{u,j}}{\sum_{j \in I_u} |sim(i,j)|}$$

Which can be interpreted as a weighted sum of the user's ratings on other items similar to i. The set I_u is the set of similar items we consider. This can either by the entire set of items, or some top-K similar items.

3.2.3 Removing Item Bias

The above method for rating prediction assumes that all users' rating methodology is objective, which might be not true. Each user has his/her own bias, which reflects in the ratings. We can remove this bias by making a small alteration to the prediction function.

Just like we removed user bias in the user-user similarity approach, we should remove any item specific bias as in our current prediction model. We do this by making the following alteration

$$R_{u,i} = \overline{r_i} + \frac{\sum_{j \in I_u} sim(i,j) * (r_{u,j} - \overline{r_j})}{\sum_{j \in I_u} |sim(i,j)|}$$

To remove this inconsistency in the user behaviour, we compute the mean rating of a user, and subtract this from all the ratings of the user. Of course, we add back the mean rating of the user for whom we are computing the rating.

4 Recommendation Using Classification and Regression

We can also formulate the problem of predicting missing ratings as a classification problem as follows:

To predict the rating of item i by user u, the possible classes (for Movielens data) are $\{1, 2, 3, 4, 5\}$, the training feature vectors are the ratings of items $\{1 \dots i-1, i+1 \dots n\}$ by all the users where n is the total number of items and the corresponding labels are the ratings by user u for these items. The prediction vector is the ratings of item i by all the users.

Similarly, using the same feature vectors, we also tried framing this as a regression problem. This was more intuitive since our target variables were ordinal. It is better if our model predicts 3 for a true rating of 4, rather than predicting a 1. This information can be nicely capture if we use a loss like MSE or MAE and regressing. The results are summarised in the table.

5 Cold Start

The Cold Start problem in recommendation systems is the problem of predicting ratings by new users i.e users which are not present in the training data. Since no previous data about users is available, our previous memory based approaches and model based approaches will fail. In this part of our project, we try to tackle the cold start problem by using the meta data available for users and movies (also distributed as a part of the MovieLens data set).

We treat this as a regression problem, trying to predict a value between 1 and 5. We extract features from the meta data and train a multi-layer perceptron using them. The features are extracted in the following manner

5.1 Extracted Features

We use a mixture of user and item meta data to generate the feature vector. All features were one-hot encoded and then concatenated. The final feature vector had length 64, comprising of the following

- Item Genre (dimension = 19)
- Item Release Year (binned into 12 buckets)
- User Occupation (dimension = 21)

Algorithm	RMSE	MAE
Baseline – Global Average	1.124563	0.944230
Baseline – Item Average	1.124661	0.944121
MLP (64, 64) + ReLU	1.073124	0.87143
MLP (128, 128) + ReLU	1.032214	0.86312

Table 2: Performance of our MLP regressor (Root Mean Squared Error and Mean Absolute Error) on the test folds. We report the average across the 5 test folds. Performance of baselines are also present for comparison.

- User Gender (dimension = 1)
- User Age (binned into 11 buckets)

For the Item Release Year feature, we created 12 bins of 10 years each starting from 1900 till 2020. For the User Age, we created 11 bins of size 10, beginning from 0 till 110. This allowed us to represent all ages in the MovieLens data set.

We use these extracted features and regress over the training set. We sklearn Neural Network Regression module, training a two layer network with ReLU activation.

5.2 Evaluation

Our evaluation metrics remain the same as before, root mean squared error and mean absolute error. We once again perform 5-fold cross validation, however the folds are generated in a different manner than in the previous approaches. Rather than splitting user-movie pairs into 80-20, we split the users into 80-20 train-test ratio. We train our model on all the user-item rating pairs of the users in the train set, and test our model on the user-item rating pairs of the users in the test set. This ensures that during testing, our model has never seen a user before.

6 Results and Conclusion

In this project we explored multiple collaborative filtering algorithms using the MovieLens data set. We framed the problem in multiple ways and explored different approaches including similarity based models, classification models and regression models.

We notice that the best approach is the memory based item-item similarity model, with bias removal. This approach achieves a MAE of 0.75286 across the 5 test folds. It is significantly better than the three baselines, and within comparable range

of LibRec benchmarks (which use SVD based algorithms).

In the second part of our project, we tried to solve the cold start problem on the same data set. We extracted features from user and item metadata and used a simple NN regressor with ReLU activation to make predictions. Our best model is a two layer network with 128 neurons each, which achieves a MAE of 0.86312 across the 5 test folds. This is significantly better than the baslines.

7 Future Work

In the future, we would like to explore latent factor based approaches. Matrix factorization and SVD models are heavily used in recommendation systems, we didn't get sufficient time to explore these during the semester.