Movielens: Several Approaches to Rating Prediction

Yifang Peng
A53216518
Email: yip032@ucsd.edu

Yuqing Duan
A53203965
Email: yud087@ucsd.edu

Zhaowen Zou
A53212285
Email: zwzou@ucsd.edu

***Abstract*—In this project, our group analyse the data of movies, users and ratings taken from MovieLens website to accomplish the movie rating prediction task. We train models with multiple methods, namely Latent Factor, Item-based K nearest neighbours (KNN), User-based K nearest neighbours (KNN) and Binary Decision Tree Regression (BDTR). Then we calculate the prediction error of each model. By comparing the prediction errors, we can evaluate the performance of the proposed models.**

I. Introduction

MovieLens was created by the GroupLens [1] Research, a research lab in the Department of Computer Science and Engineering at the University of Minnesota to collect large-scale data for personalized recommendation research.

MovieLens is a web-based recommender system that recommends movies to users. The data is generated from the reviews and ratings according to a one-to-five-star rating system submitted by users, together with the customized tags added to movies and the movie classification.

Our goal is to build our own recommender system using the data taken from MovieLens, which recommends movies to a user to meet his or her preference as well as possible. Therefore, we want to predict how a user will rate movies he or she has not watched, and recommend those with higher potential ratings. In this project, we focus on the rating prediction. We try to build several prediction models using multiple methods, and evaluate the performance of these models according to the prediction error (MSE and MAE).

II. Dataset

A. Dataset Description

The dataset is from the Movielens website hosted by Grouplens Research. (https://grouplens.org/datasets/movielens/) Subject to our equipment and time limitations, we cannot deal with a super large-scale dataset. Therefore, we choose the small latest datasets.

1) Dataset Format
   - movies.csv: movieId, title, genres
   - tags.csv: userId, movieId, tag
   - ratings.csv: userId, movieId, rating
2) Statistics
   The dataset we chose contains 100,000 ratings and 1,300 tag applications applied to 9,000 movies by 700 users.

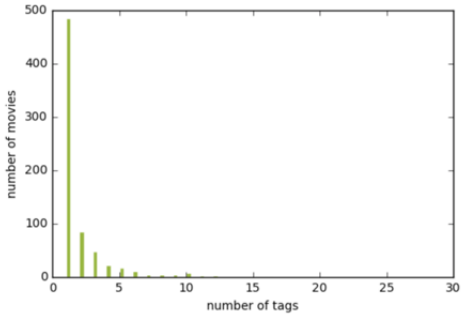

Fig. 1: Movie tag destribution

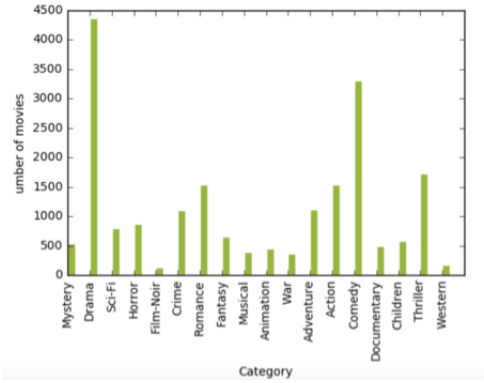

Fig. 2: Movie category distribution

B. Perform Exploratory Analysis

Because the features of users and movies provided by the original dataset are not enough or clear, we want to pre-process the data to generate more useful user and movie features by ourselves.

The number of categories and tags given in the dataset is quite huge (18 movie categories and 521 tags). If we generate a uniform movie feature vector with each single tag and category for each movie, the matrix containing the feature vectors of all the 9125 movies is too large, and too sparse as well. (Figure 1)

Since a movie can belong to multiple categories and tags, we can solve this problem by clustering similar categories and tags. In this project, we generate 3 clusters for both the tags and the categories using K-Means. (Figure 2)

C. Dataset Limitations

It is a pity that the information involved in the dataset from MovieLens is too little. If we can get more information about the users and movies, for instance, the gender, age, education level of users, as well as the actors, directors, and length of movies, the prediction may become more overall and accurate.

In fact, the MovieLens website provided much larger dataset. (The larger version covers 20 million ratings, 27,000 movies and 138,000 users.) However, the amount of computation and demand of memory will be multiple times of the current one. Due to our equipment and time limitations, we cannot use it in order of more accurate prediction.

III. Predictive task

In order to build a recommender system that recommends movies to meet users preference as well as possible, we want to predict the ratings a user will give to movies he or she has not watched, and recommend those with higher potential ratings to the user.

Therefore, the predictive task this project focuses on is to do the rating prediction given a user and a movie, taking features of the user and movie, as well as the previous rating data we have (the train dataset) into consideration.

IV. Models

To put it briefly, we build several models using three approaches: content-based filtering, collaborative filtering as well as a hybrid approach.

The content-based filtering approach is based on the movies features and the users preference. The rating is predicted by matching the user interests with the movie features. This approach is related with machine learning techniques such as Logistic Regression. In this project, the Binary Decision Tree Regression (BDTR) method is used to implement the content-based filtering approach.

The collaborative filtering approach analyses the previous rating data. To predict how a user will rate a movie, we can refer to the ratings given to this movie by other similar users, or the ratings this user gave to other similar movies. The K-Nearest Neighbour (KNN) models can be used to implement this approach.

We try to combine the above two approaches to train hybrid models as well.

In the following parts, we will discuss the design and implementation details of the models we used.

A. Content-based Filtering

In order to create user and movie profiles, we used raw numerical data from selected features in the user and movie tables, as well as derived features. The feature we generated for content-based filtering is average rating of user, average rating of movies in each category of user, category of movie (uniform), average rating of movie. There are 18 categories of movies and there would be too many features if we user each category in our feature. Thus, we applied K-means to 18 categories and cluster them into 3 groups and created the above mentioned features based on the new categorization.

The features we generated are in the following form:

$$[avg\_rating\_user[u],$$
$$user\_category\_avg\_rating[u][0],$$
$$user\_category\_avg\_rating[u][1],$$
$$user\_category\_avg\_rating[u][2],$$
$$movie\_category\_cluster[m][0],$$
$$movie\_category\_cluster[m][1],$$
$$movie\_category\_cluster[m][2],$$
$$avg\_rating\_movie[m]]$$

We used Binary Decision Tree Regression to train on the generated features.

1) Binary Decision Tree Regression: Decision tree learning [3] (a.k.a CART which stands for Classification and Regression Trees) uses a decision tree as a predictive model which maps observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). It is one of the predictive modelling approaches used in statistics, data mining and machine learning. Tree models where the target variable can take a finite set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.

Decision tree has following advantages:

- Simple to understand and to interpret. Trees can be visualised.
- Requires little data preparation.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic.

Decision tree also has disadvantages:

- Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.

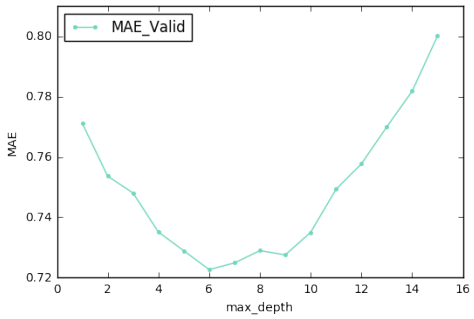

Fig. 3: BDTR MAE vs tree depth

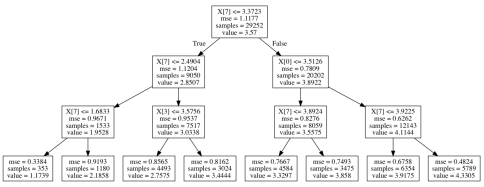

Fig. 4: BDTR tree (max_depth = 3)

- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

Python class DecisionTreeRegressor was applied to the training set to build the decision tree. Max_depth was set from 1 to 15. Refer to Figure 3 to see the relationship between max_depth and MAE on validation set. Figure 4 shows the binary decision tree when max depth equals to 3.

B. Collaborative Filtering

Collaborative filtering, also referred to as social filtering, filters information by using the recommendations of other people. Item-based collaborative filtering based on the idea that people tend to like items that are similar to what they liked in the past. In the context of movie, we can imagine that if a person tends to rate comedy generally higher than other categories of movies, he would rate a new comedy relatively higher. User-based collaborative filtering based on the idea that people who agreed in their evaluation of certain items in the past are likely to agree again in the future. A person who wants to see a movie for example, might ask for recommendations from friends. The recommendations of some friends who have similar interests are trusted more than recommendations from others. This information is used in the decision on which movie to see.

1) Linear latent factor model

We start with the latent factor model we discussed in class:

$$f(u,i) = \alpha + \beta_i + \beta_u$$

We train the data with lambda = 0.1, 1, 10,100 and we find it is optimal to use lambda = 1 on validation set. We apply the model to test set and we get:

$$Minimal\_test\_MAE = 0.702151309$$

$$Minimal\_test\_MSE = 0.829224171$$

2) Non-linear latent factor model

For the non-linear model[2], we learn that the optimization equation is not convex, so we use the alternating least squares procedure. In each step, we fix alternately gamma_i or gamma_u and then solve the convex sub-problems until it converges (the difference between the value of optimization equation is less then 1e-10 for two following iteration) . And we use both MAE and MSE to measure the performance. We calculate the derivatives and the updates is as following:

$$f(u,i) = \alpha + \beta_i + \beta_u + \gamma_i * \gamma_u$$

The 5 derivative equations:

$$\alpha(t+1) = \frac{\sum_{u,i \in train} R(u,i) - \beta_i - \beta_u - \gamma_u * \gamma_i}{N_{train}}$$

$$\beta_i(t+1) = \frac{\sum_{u \in U_i} R(u,i) - \alpha - \beta_u - \gamma_u * \gamma_i}{\lambda + |U_i|}$$

$$\beta_u(t+1) = \frac{\sum_{i \in I_u} R(u,i) - \alpha - \beta_i - \gamma_u * \gamma_i}{\lambda + |I_u|}$$

$$\gamma_i(t+1) = \frac{\sum_{u \in U_i} (R(u,i) - \alpha - \beta_u - \beta_i) * \gamma_u}{\lambda + \sum_{u \in U_i} \gamma_u^2}$$

$$\gamma_u(t+1) = \frac{\sum_{i \in I_u} (R(u,i) - \alpha - \beta_u - \beta_i) * \gamma_i}{\lambda + \sum_{i \in I_u} \gamma_i^2}$$

It takes time for the non-linear model to converge. However, since the size of our data is not that large, the running time is not too bad. We find it is still optimal to use lambda = 1.0 and we get $MAE = 0.702151304$ and $MSE = 0.829224175$.

3) Item-based K Nearest Neighbors (KNN)

In the item-based KNN model, we predict the rating of user u on item i by looking at the top k items that are similar to i, and produce a prediction by calculating the weighted average of ratings from user u on these items. We first calculate the cosine similarity of each pair of movie X and movie Y:

$$\text{siml}(x,y) = \frac{\sum_{u \in U_{xy}} avg\_r_{u,x} avg\_r_{u,y} + C_x C_y + T_x T_y}{\sqrt{\sum_{u \in U_x} avg\_r_{u,x}^2 + C_x^2 + T_x^2} \sqrt{\sum_{u \in U_y} avg\_r_{u,y}^2 + C_y^2 + T_y^2}}$$

We predict the value of ratings user u gives to item i as a weighted average of similar items user u has rated:

$$r_{u,i} = \frac{\sum_{n \in N_u^k(i)} siml\,(i,n) * r_{u,n}}{\sum_{n \in N_u^k(i)} siml\,(i,n)}$$

4) User-based K Nearest Neighbors (KNN)

This model is rather similar to item-based KNN. The only difference is that we calculate similarity between each pair of users and predict the rating of user u on item i by looking at the top k users that are similar to u, and produce a prediction by calculating the weighted average of ratings from these users on the item.

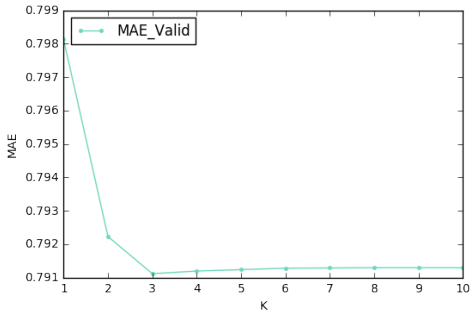

Fig. 5: MAE on validation set vs K

In this approach, we calculated similarity based on following method:

$$siml\,(x,y) = cost\,(\overrightarrow{x},\overrightarrow{y}) = \frac{\sum_{i\in I_{xy}} r_{x,i}r_{y,i}}{\sqrt{\sum_{i\in I_x} r_{i,x}^2}\sqrt{\sum_{i\in I_y} r_{i,y}^2}}$$

We used cross-validation to train the model with different k. (Figure 5)

One observation from the result is that as the value of k increases, we can get a lower MAE, i.e. a better prediction. However, the improvement plateaus at k = 3.

3) Non-linear latent factor model

C. Hybrid Model

We implemented a hybrid model to combine both content-based filter and collaborative filter. The purpose is to take the advantage of users and movies profile while also maintaining the advantage of a neighborhood model.

We trained KNN model first (item-based) and obtained predictions. The prediction was then added to Binary Decision Tree Regression model as a feature to recalculate the model.

V. Literature

The MovieLens dataset is collected and provided by the GroupLens Research lab at the University of Minnesota. Given users, movies and ratings submitted by some of the users for some of the movies previously, the predictive task of movie rating prediction the ratings has been studied in the literature in the form of recommender systems, which additionally perform the task of recommending movies to users.

According to related work, the two prevalent approaches to build recommender systems are Collaborative Filtering and Content-based recommending approaches. Collaborative Filtering methods is based on exploiting similarities among profiles of users. Content-based methods is determined by features of the movies and the movies meeting the interests of the given user.

Melville proposed a hybrid content boosted collaborative filtering system. In addition, content-based techniques are used to generate user features like pseudo-ratings. Then similarities among users are calculated as the Pearson correlation coefficient according to the original user features, and the rating prediction is computed using user-based collaborative filtering approach. [4] In this project, we also build several models implementing for Collaborative Filtering, Content-based and hybrid recommending approaches.

VI. Results

In order to evaluate the performance of the methods we implement, for each model we calculate the prediction errors using the test dataset. The most widely used evaluation metrics for recommender system are mean squared error (MSE) and mean absolute error (MAE). We used both to evaluate the performance of the above six methods we implement.

MSE is defined as:

$$MSE = \frac{\sum (\hat{r}_{u,i} - r_{u,i})^2}{n}$$

MAE is defined as:

$$MAE = \frac{\sum |\hat{r}_{u,i} - r_{u,i}|}{n}$$

where $\hat{r}_{u,i}$ is the predicted rating from user u on item i, $r_{u,i}$ is the actual rating and n is the size of test dataset.

As to the feature representations, as mentioned in the dataset part of this report, the information about each movie or user we can get from the MovieLens dataset is not much, so we decide to augment the movie and user profiles. We add average ratings related to a single user or movie to the features. The ratings submitted by each user to movies of different kind can also be a characteristic of the user. We also set categories and tags to be characteristics of movies. However, the number of tags and categories are so huge that the distribution seems sparse. Therefore, we use python K-Means class to merge them into more general features.

Table I shows the MSE and MAE of each model we implement.

Of all the models, the non-linear latent factor model has the best performance whereas user-based KNN has the least performance.

It is interesting to note that the non-linear model, which is suppose to be more complex and accurate, does not outperform the simple linear model. It is because the data is sparse enough so it is hard for a pair of user and movie in validation data set together that appear in the training set.

Comparing KNN to latent factor model, KNNs performance is not good. The main reason is that movieLens rating data is highly sparse. Specifically, even we are aware of the k items most similar to the item i, the probability of user u having rated any of these items is relative small. So we are still not able to make good predictions. In statistics, a power law is a functional relationship between two quantities, where a relative change in one quantity results in a proportional relative change in the other quantity, independent of the initial size of those quantities: one quantity varies as a power of another. We examine whether movieLens graph follows a power law distribution. Figure 6 and 7 illustrate the degree distribution of each node type in a log-log scale(user-rating count and movie-rating count). We can see that that their

TABLE I: Results of various algorithms

Method	MSE	MAE
Baseline	0.829224171	0.702151309
Latent Factor Model (non-linear)	0.829224175	0.702151304
Binary Decision Tree Regression	0.955519724	0.724868757
Item-based KNN	1.082323384	0.788465577
User-based KNN	1.047869821	0.790273905
BDTR + Item-based KNN	0.949782511	0.722663867

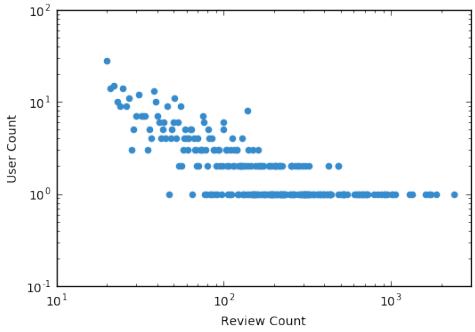

Fig. 6: Degree Distribution of User Rating Counts

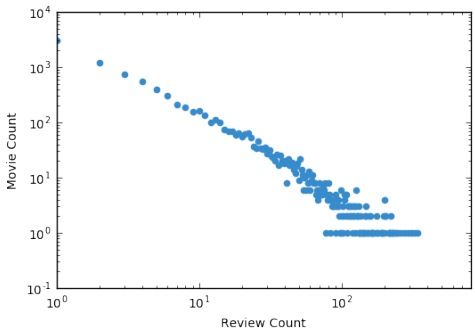

Fig. 7: Degree Distribution of Movie Rating Counts

distribution follows an exponential pattern. This confirms our assumption of cold start problem of traditional latent factor recommendation model, and proofs that similarity based model can result worse performance on the data.

In the user-based KNN model, we calculated similarity based on two methods. One is the users average ratings on different categories of movies and the other one is the users ratings on different movies. Result turns out that the second more complicated method has better performance.

In BDTR model, we first used the original categories as our content-based features, the result was not good enough. After using K-means to cluster categories, the perforamance was enhanced. The dimension reduction approach works pretty well under this model.

The hybrid model shows better performance than solely cotent-based model since it takes consideration of users and movies features and also takes advantage of a neighborhood model.

REFERENCES

[1] GroupLens MovieLens small Latest dataset, , http://files.grouplens.org/datasets/movielens/ml-latest-small-README.html.

[2] Latent Factor Models for Web Recommender Systems, http://www.ideal.ece.utexas.edu/seminar/ LatentFactorModels.pdf.

[3] Wei-Yin Loh, *Classification and regression trees*. John Wiley & Sons, 2011. 14-23.

[4] P. Melville, R. J. Mooney, and R. Nagarajan, *Content-boosted collaborative filtering for improved recommendations*. AAAI, 2002.