Movie Rating Prediction using the MovieLens dataset

Kun Li

kxl162530@utdallas.edu

1. Introduction

The goal of this project is to predict the rating given a user and a movie, using 3 different methods – base collaborative filtering, K-Nearest Neighbors method and latent factor model [22, 23] on the MovieLens 100k data set.

2. Data and Statistics

The initial data set, obtained from the MovieLens, a movie recommendation service, had 100,000 ratings (1-5) from 943 users on 1682 movies. Ratings are integers on a 5-star scale. Each user and each movie is identified by a unique id. Each user has rated at least 20 movies which results in a reasonable sparsity of 6.3%. This means that 6.3% of the user-item ratings have a value.

3. Movie Rating Prediction

Given a user u and a movie m, we would like to predict the rating for u will give to m. Although this problem is closely related to the problem of recommending the best possible items to a user based on the user's previous reviews or purchases, in this work we focus on accurately predicting the rating itself. Accordingly, we evaluate the performance of our models using the widely-acknowledged mean square error (MSE) defined as:

$$mse = \frac{1}{|S_{test}|} \sum_{(u,l) \in S_{test}} (r_{ul} - p_{ul})^2$$

4. Related Work

The MovieLens data set is a data set collected and made available by the GroupLens Research group. MovieLens is a website for personalized movie recommendations. The data set contains data from users who joined MovieLens in the year 2000.

Given a set of users, items and ratings given by some of the users for some of the items, the task of predicting the rating for a given user for a given movie has been studied in the literature in the form of *recommender systems*, which additionally perform the task of recommending items to users. The state-of-the-art in recommender systems is based on 2 main approaches - *neighborhood approach* and *latent factor models.* Both these methods rely only on past user behavior, without using any features about the users or items. Both these methods rely only on past user behavior, without using any features about the users or items for the best algorithm to predict user ratings for films, given only the previous ratings, without any other information about the users or films. An alternative to these methods is *content filtering* [2], which uses user features (such as age, gender, etc.) and item features (such as movie genre, cast etc.) instead of relying on past user behavior.

Neighborhood approaches and latent factor models are generally more accurate than content-based techniques [2], since they are domain-free and capture subtle information (such as a user's preferences) which is hard to extract using content filtering. On the other and, neighborhood approaches and latent factor models suffer from the *cold start* problem i.e. they are not useful on new users or new items.

In this work, we compare 3 models - the first one is user-based and item-based collaborative filtering and is

described in section 5, the second one is KNN algorithm and is described in section 6. The last one is a latent-factor model and is described in section 7.

5. Used-based and Item-based collaborative filtering

In this section, we focus on collaborative filtering models which can be generally split into two classes: user and item-based collaborative filtering. In either scenario, we build a similarity matrix. For user-based collaborative filtering, the user-similarity matrix will consist of some distance metric that measures the similarity between any two pairs of users. Likewise, the item-similarity matrix will measure the similarity between any two pairs of items. Figure 1 shows the collaborative filtering process.

5.1 Item similarity computation

The basic idea of similarity computation between two items i and j is to firstly isolate the users who have rated both items and then to apply a similarity computation technique to determine the similarity $s_{i,j}$. Figure 2 shows the isolation of the co-rated items and similarity computation.

A common distance metric is cosine similarity. The metric can be thought of geometrically if one treats a given item's column of the ratings matrix as a vector. For item-based collaborative filtering, two items' similarity is measured as the cosine of the angle between the two items' vectors. Similarity between item i and item j, denoted by $sim(i,j)$ is given by

$$sim(i,j) = \cos(\theta) = \sum_u \frac{r_{ui} r_{uj}}{\sqrt{\sum_i r_{ui}^2} \sqrt{\sum_j r_{uj}^2}}$$

Here, r_{ui} and r_{uj} are the ratings given by user u to movie i and movie j respectively.

After getting similarity between two different items, we then compute the prediction on an item i for a user u by computing the sum of the ratings given by the user on the items similar to i. The corresponding similarity weights each rating $s_{i,j}$. And final weighted sum is divided by the sum of similarity to get normalized prediction value. The prediction $p_{u,i}$is given by

$$p_{u,i} = \frac{\sum_N (S_{i,N} * R_{u,N})}{\sum_N |S_{i,u}|}$$

This approach tries to capture how the active user rates the similar items. The weighted sum is then scaled by the sum of the similarity terms to make sure the prediction is in the specific range.

5.2 User-based CF

Figure 3 shows the isolation of the co-rated users and similarity computation. The user-based CF is quite like item-based CF. Instead of computing the similarity between two items, we focus on the similarity between two customers. We use cosine similarity between two customers u, v, which is the same as item-based method. We denote that similarity as $S_{u,v}$. The prediction on an item for a user u is calculated by computing weighted sum of different users' ratings on item i. The prediction $p_{u,i}$ is given by

$$P_{u,i} = \frac{\sum_u (r_{u,i} * S_{u,v})}{\sum_v S_{u,v}}$$

Where $r_{u,i}$ is the ratings of user u on item i.

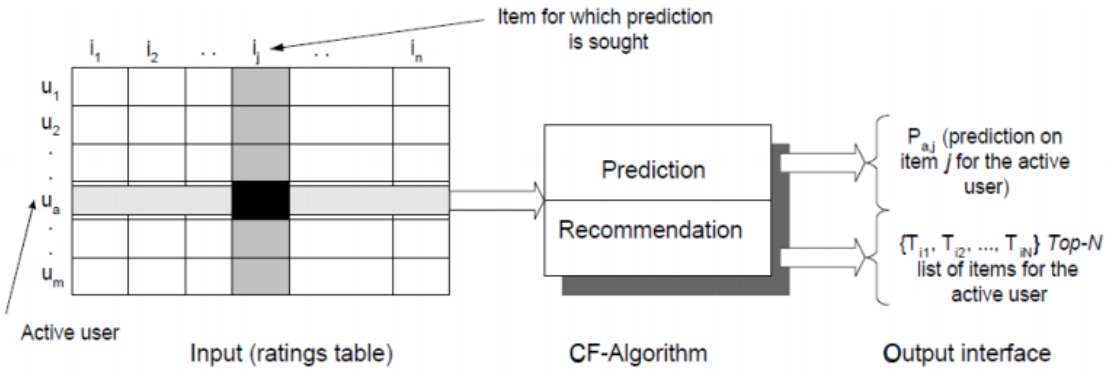

Figure 1: The collaborative filtering process.

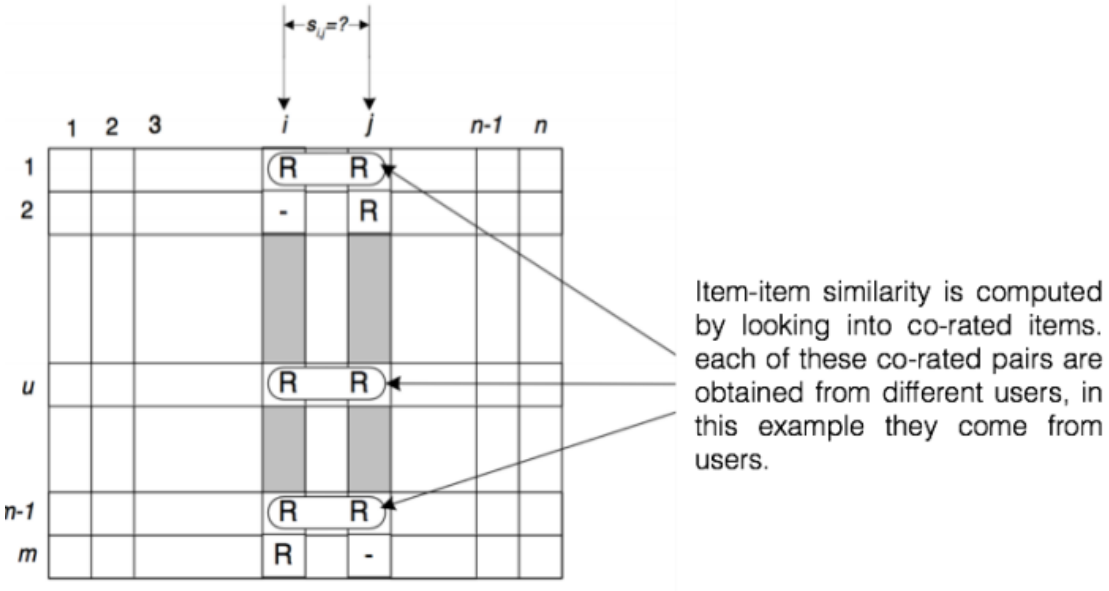

Figure 2: Isolation of the co-rated items and similarity computation.

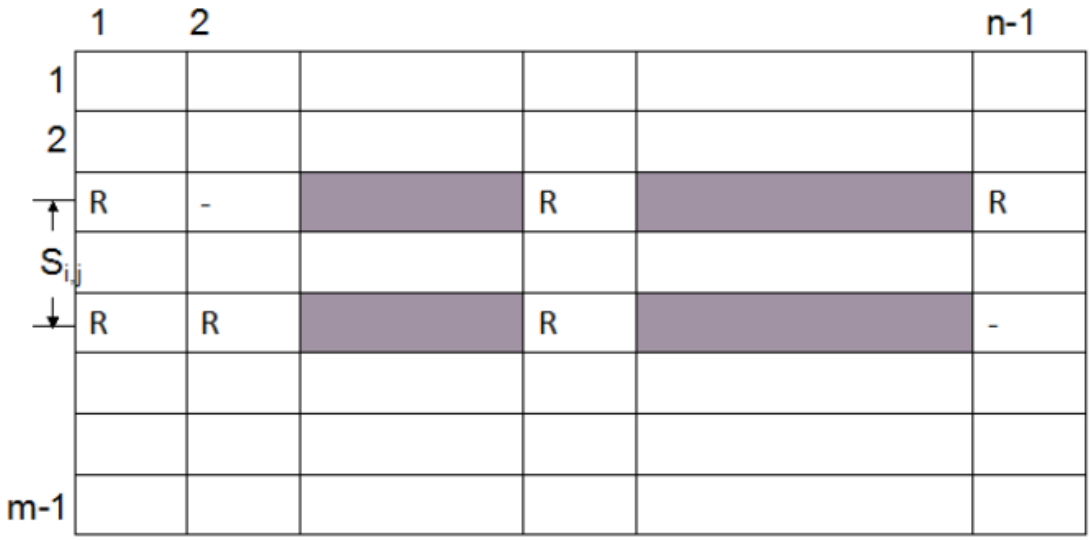

Figure 3: Isolation of the co-rated users and similarity computation.

6. KNN Algorithm for Collaborative Filtering

6.1 Item-based K Nearest Neighbor algorithm

The philosophy is the same as the above item-based collaborative filtering algorithm. In order to determine the rating of User u on Movie m, we can find other movies that are similar to Movie m, and based on User u's ratings on those similar movies we infer his rating on Movie m. In order to determine which movie are "similar", we need to define a similarity function, we use the Euclidean metric between Movie i and j:

$$sim(i,j) = \sqrt{\sum_{u}(r_{u,i} - r_{u,j})^2}$$

Where $r_{u,i}$ and $r_{u,j}$ are user u's rating on movie i and j.

The difference is that KNN finds the nearest K neighbors of each movie under the above defined similarity function and use the weighted means to predict the ratings. The prediction formula is as follows:

$$P_{u,m} = \frac{\sum_{j \in N_u^K(m)} sim(m,j) R_{j,u}}{\sum_{j \in N_u^K(m)} |sim(m,j)|}$$

where $N_u^K(m = \{j: j$ belongs to the K most similar movies to movie m and user u has rated j$\}$, and $R_{j,u}$ are the existent ratings (of user u on movie j) and $P_{u,m}$ is the prediction.

6.2 user-based K nearest neighbor algorithm

The user-based K nearest neighbor algorithm is the same as the item-based K nearest neighbor algorithm, except that we focus on the similarity between two customers. We use the Euclidean metric between user u and user u':

$$sim(u,u') = \sqrt{\sum_{i}(r_{u,i} - r_{u',i})^2}$$

Where $r_{u,i}$ and $r_{u',i}$ are ratings of user u and user u' to movie i.

And we choose top k most similar neighbors of each user under the above defined similarity function and use the weighted means to predict the ratings. The prediction formula is as follows:

$$P_{u,m} = \frac{\sum_{j \in N_u^K(m)} sim(m,j) R_{j,u}}{\sum_{j \in N_u^K(m)} |sim(m,j)|}$$

where $N_i^K(m = \{j: j$ belongs to the K most similar user to user m and user u has rated j$\}$, and $R_{j,u}$ are the existent ratings (of user u on movie j) and $P_{u,m}$ is the prediction.

7. Latent Factor Model

Given a user u and a movie I, we predict the rating that the user will give to the movie as follows:

$$r_{u,i} = \mu + b_u + b_i + \gamma_u \cdot \gamma_i$$

Where μ is the global bias, and b_u (b_i) is the user (item) bias. γ_u and γ_i are latent factors for user u and movie

i respectively, which will be learned during the training process. γ_u and $\gamma_i$are K-dimensional vectors. The error function L is defined as follows:

$$L = \sum_{u,i}(r_{u,i} - (\mu + b_u + b_i + \gamma_u \cdot \gamma_i))^2 + \lambda_{ub}\sum_u \|b_u\|^2 + \lambda_{ib}\sum_i \|b_i\|^2 + \lambda_{u\gamma}\sum_u \|\gamma_u\|^2 + \lambda_{i\gamma}\sum_i \|\gamma_i\|^2$$

Where $\lambda_{ub}, \lambda_{ib}, \lambda_{u\gamma}, \lambda_{i\gamma}$ are used to control the trade-off between accuracy and complexity during training, and $\sum_u \|b_u\|^2$, $\sum_i \|b_i\|^2$, $\sum_u \|\gamma_u\|^2$, $\sum_i \|\gamma_i\|^2$ penalize model complexity and reduces over-fitting.

We have the following expressions for the gradient of the error function:

$$\frac{\partial L}{\partial b_u} = 2\left(r_{u,i} - (\mu + b_u + b_i + \gamma_u \cdot \gamma_i)\right)(-1) + 2\lambda_{ub} b_u$$

$$\frac{\partial L}{\partial b_u} = 2\big(error_{u,i}\big)(-1) + 2\lambda_{ub} b_u$$

$$\frac{\partial L}{\partial b_u} = -error_{u,i} + \lambda_{ub} b_u$$

$$\frac{\partial L}{\partial b_i} = -error_{u,i} + \lambda_{ib} b_i$$

$$\frac{\partial L}{\partial \gamma_u} = -error_{u,i}\gamma_i + \lambda_{u\gamma}\gamma_u$$

$$\frac{\partial L}{\partial \gamma_i} = -error_{u,i}\gamma_u + \lambda_{i\gamma}\gamma_i$$

There are two approaches to solving the above optimization problem to find all of our features – stochastic gradient descent(SGD) and alternating least squares(ALS). We use both SGD and ALS to update these parameters end up being:

$$b_u \leftarrow b_u + \eta(error_{u,i} - \lambda_{ub} b_u)$$

$$b_i \leftarrow b_i + \eta(error_{u,i} - \lambda_{ui} b_i)$$

$$\gamma_u \leftarrow \gamma_u + \eta(error_{u,i}\gamma_i - \lambda_{ub}\gamma_u)$$

$$\gamma_i \leftarrow \gamma_i + \eta(error_{u,i}\gamma_u - \lambda_{ub}\gamma_i)$$

We run SGD for 200 iterations (where each iteration is one pass through the entire training set). We use grid search to find out the best latent factor K = 80 and those four regularization term are all equal to 0.01.

8. Comparison of the Models

The collaborative filtering models discussed in section 5 and 6 are known as neighborhood methods in literature [3]. Latent factor models are more expressive and tend to provide more accurate results than neighborhood models. Because latent factor models have the ability to capture and learn the latent factors which encode users' preferences and items' characteristics. On the other hand, neighborhood models are relatively simpler and offer more intuitive explanations of reasoning behind recommendation. Both models suffer from the cold-start problem, but give good performance for user and items with enough data [4].

9. Results and Conclusions

This section discusses the results of the 3 models discussed in the previous sections. We will split our data into training and test sets by removing 10 ratings per user from the training set and placing them in the test set. We run each of the models on the same training and test set for a fair comparison. Table 1 lists the MSE values for each of the models.

	Training	test
User-based CF	2.77371130536	2.77152813784
Item-based CF	2.93319437285	3.34185232357
User-based KNN		2.3628
Item-based KNN		2.9213
Mixed user-item-based KNN		2.1421
Latent Factor Model(ALS)	3.70500202429	3.75176169294
Latent Factor Model(SGD)	0.950595604635	1.01422554245

The results in table 1 indicate that the latent factor model with SGD optimization method perform best among the 7 models we consider. This is expected, as mentioned in section 8.

Based on the results, we conclude that latent factor models tend to perform better than neighborhood approaches on the movie rating prediction tasks, provided there is enough data for each user and for each movie.

[1] GroupLens MovieLens 100k dataset. http://files.grouplens.org/datasets/movielens/ml-100k-README.txt

[2] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42:30{37, August 2009

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

[4] Y. Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4, January 2010.