Movie Recommender System

Group #6, TERA: Cuisheng Yao, Qi Wu, Shuai Guo, Xiaoyuan Liu

College of Computer and Information Science, Northeastern University

benjiminhugh@gmail.com
wuqiank@gmail.com
guoshuai5156@gmail.com
xy.jason.liu@gmail.com

ABSTRACT

Recommender system is usually an essential part of online video or music systems, such as YouTube and Douban FM, defined as the technique to predict the rating of a new item given related information. For example, YouTube recommender system will list out the "similar" videos that you may want to watch next according to your watching history. Recommendation system saves a lot of time for a single user to cope with the problem of too much choice. Our team implements a movie recommender system that builds the model by training the data set from Yahoo! Movie Community, which basically takes a user and his or her list of movies watching history and outputs a number of recommended movies based on the rating it predicts. We accomplished three algorithms, KNN, Slop One and SVD, and we implemented the first two algorithms by ourselves and used a third party library for SVD. In this report, we will take an in-depth view of our movie recommender system, including the background, approaches, evaluations and algorithms comparison.

Keywords: *movie recommendation; KNN; Slop One; SVD*

BACKGROUND

Movie Recommender System

For movie lovers in today's world, it is more and more convenient to watch hit movies online. However, the explosion of internet and information also bring each customer too many choices. Which movie should they watch instead of others? To help customers cope with this problem, a movie recommender system is very necessary for guiding the customers. A typical movie recommendation system saves the customers watching history and then recommends new movies using the preference of other "similar" customers. The concept of the similarity is the key point of solving recommendation problem. In our system, we both considered the similar users and similar movies.

PROBLEM FORMALIZATION

Goal

Although we used three different methods, the final goal of this movie recommender system remains the same, which is to recommend a number of movies determined by user as an input parameter for each unseen customer according to the rating. Once we got the result, basically we can evaluate the precision by compare the predicted answer with the ground truth.

Data Set

The data set we use contains a small sample of Yahoo! Movies community's preference. We have got a "user-movie-rating" training set of 211231 data points, and a test set of 10136 data

points. Also, a 7642 points set of user info and a 106959 points set of movie descriptive info.

(1) "***ymovies-user-movie-ratings-train-v1_0.txt***"

anonymized user_id	movie_id	rating (1 to 13)	converted rating (1 to 5)

(2) "***ymovies-user-movie-ratings-test-v1_0.txt***" This data set is treated as the ground truth, which has the same format training data set.

Developing Environment & Tools

Our team prefers Python as developing language with third party libraries. We want to first implement our own algorithm then make a comparison with the library functions.

APPROACHES

Data Preprocessing

All the data sets are saved in txt format, separating each attribute by space and each data point by a new line, which is good. However, there are a lot of dirty points with empty attribute value. Some attributes also are totally useless, for example, "movie description" is too tedious to focus.As to the user-movie-rating set, we first scan the whole dataset and compute the average rating for every movie. We record the result in a list and this list of average rating for every movie will be used in judging the noisy user.

Classification

Lazy Leaner, K-Nearest Neighbor:

The idea of using this model is that we want to find the similar users to a specified user, and we can predict the rating of this specified user based on the similar users. In the process of building the KNN model, we found the most difficult thing is to decide what can present the similarity of two users. In the textbook it uses distance to present the similarity such as ***Euclidean distance***, ***Manhattan*** distance and ***Cosine Similarity***.

$$\boldsymbol{Euclidean\ Distance}: \boldsymbol{d(p,q)} = \sqrt{\sum_{i=1}^{n}(\boldsymbol{p_i} - \boldsymbol{q_i})^2}$$

$$\boldsymbol{Cosine}: \ \boldsymbol{d(p,q)} = \frac{\sum_{i=1}^{n} \boldsymbol{A_i} * \boldsymbol{B_i}}{\sqrt{\sum_{i=1}^{n}(\boldsymbol{A_i})^2} * \sqrt{\sum_{i=1}^{n}(\boldsymbol{B_i})^2}}$$

So we firstly use these distance measure to find out the users near a specified user. In a simple case of two dimensional spaces, suppose two customers have 2 ratings for 2 movies each, saying user x_1 has a rate pair (r_1, r_2) for movie m_1 and m_2, while user x_2 has (r_3, r_4), the similarity is just the Euclidian distance between this two vectors. However, this is just a very simple situation with nothing else we can use, like the inherent relationship between movies, specifically, the type, director, actors and so on.

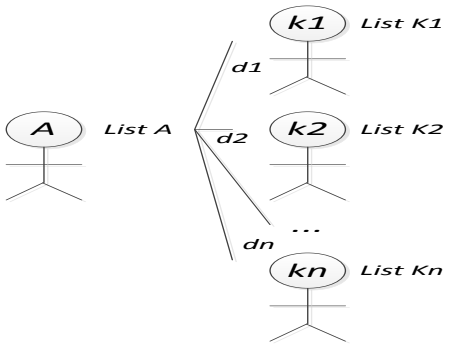

Figure 1

Figure 1 gives a brief demonstration of how we apply KNN algorithm into our system. Suppose now we want to recommend movies for user ***A***, and we have found n nearest neighbors ***{k1, k2 … kn}*** with n distances ***{d1, d2 … dn}*** and each user keeps a list of the id of movies that he watched, along with his rating of them. First we traverse each movie ***x*** in each neighbor ***k*** of ***A***, if ***x*** is not in the list of ***A***, then we give a weighted relative rating of ***x***, which is denoted as $\boldsymbol{R^*_{xk}}$:

$$\boldsymbol{R^*_{xk}} = \boldsymbol{R_{xk}} \cdot \frac{\boldsymbol{d_k}}{\sum_{i \in S(x)} \boldsymbol{d_i}}$$

Where $\boldsymbol{S(x)}$ is the set of neighbors who also rate movie x, and $\boldsymbol{R^*_{xA}}$ is final predicted rating, denoted as:

$$\boldsymbol{R^*_{xA}} = \sum_{i}^{n} \boldsymbol{R^*_{xk_i}}$$

Once we finished computing all the ratings of new movies of ***A***, we pick out 5 movies with the highest scores as recommendations. In the early period of algorithm implementation, the result is not very good and we found the reason, which is that our data set is too sparse. For example, say two customers ***x1*** and ***x2***, they have not watched even one same movie, so we don't know the similarity of the two. However the distance

between them will be zero, which indicates they are very similar. Given this situation, in our finial coding of KNN model, we used ***Cosine Similarity*** as our similarity measure, which performs much better than the ***Euclidean*** distance when dealing with the sparse data set. Also, due to the sparse matrix problem, sometimes we will get a scenario that all the distances from d_1 to d_n is 0, which will cause mathematic error when compute the weight division, so we will add a very tiny number to denominator if necessary. Our team also improved the precision rate by computing the average rating $\bar{R}$of each movie among all users, if the absolute difference between predicted R' and $\bar{R}$ is larger than 6, then we will dismiss the R' given by such user.

Slope One:

	M_1	M_2
U_1	5	6
U_2	4	5
U_3	3	?

Figure 2

Figure 2 is the basis of Slope One schemes. Let us consider a situation of three users U_1, U_2 and U_3, and they have watched 2 common movies M_1 and M_2. Now we want to predict the rating of M_2 given by U_3. We observed that M_2 is rated more than M_1 by $6 - 5 = 1$ for U_1, and $5 - 4 = 1$ for U_2. So the average difference is $\frac{1+1}{2} = 1$, thus we can predict that U_3 will give M_2 a rating of 3+1= 4. Formally, given a training dataset χ and two movies i and j with ratings u_i and u_j, $S_{i,j}(\chi)$ is the set of the users who have rated both movie i and movie j, then we consider the average deviation of item j respect to i as:

$$dev_{j,i} = \sum_{u \in S_{j,i}(\chi)} \frac{u_j - u_i}{NumOf(S_{j,i}(\chi))}$$

So $dev_{j,i} + u_i$ is the predicted rating for movie j, and for all such predictions:

$$P(u)_j = \frac{\sum_{i \in S(u)} (dev_{j,i} + u_i) * NumOf(S_{j,i}(\chi))}{\sum_{i \in S(u)} NumOf(S_{j,i}(\chi))}$$

$S(u)$ is the set of movies rated by user u.Here we give a more complicated example:

	M_1	M_2	M_3	M_4
U_1	5	6	7	8
U_2	4	5	6	7
U_3	3	?	5	6

Figure 3

$dev_{2,1} = ((6-5) + (5-4)) / 2 = 1$
$dev_{2,3} = ((6-7) + (6-5)) / 2 = -1$
$dev_{2,4} = ((6-8) + (6-7)) / 2 = -2$
$P(u)_2 = ((1 + 3)*2 + ((-1) + 5)*2 + ((-2) + 6)*2) / (2+2+2) = 4$

So the predicted rating of M_2 by U_3 is 4. Since in our data, the range of movie rating starts from 1 to 13, so if the final rating exceeds the upper bound or lower bound, we just make the rating score the maximum 13 or minimum 1. Instead of trying to find the similar user, Slope One algorithm, for each new user, it takes not only the information from other users who rated the same movie, but also the information from other items from itself.

Singular Value Decomposition:

SVD is an operation of factorization of a matrix, in order to reduce the high dimensionality (as mentioned in data preprocessing, after parsing the data set, each user has a list of movies watched as his attributes). Formally, it is used for an $m * n$ matrix M, which has

$$M = U\Sigma V^*$$

where U is an $m*m$ unitary matrix, Σ is an $m*n$ diagonal matrix with nonnegative in the diagonal, V^* is an $n*n$ unitary matrix, and the diagonal entries are ***singular values***. So in our system, the matrix M is 7642 * 106959, U is 7642 * 7642, Σ is 7642 * 106959, V^* is 106959 * 106959.

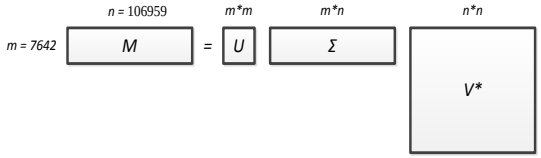

Figure 4

The diagonal entries of Σ are sorted in a decreasing order, and we take first k largest entries of Σ, which results in $\widetilde{\Sigma}$.

So $\widetilde{\Sigma}$ is a k*k matrix with non-empty value in its diagonal, formally, $\widetilde{\Sigma} = \begin{pmatrix} a_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & a_{kk} \end{pmatrix}$.

Then we delete the columns from column k+1 to m in U and get $\widetilde{U}$; delete the rows from row k+1 to n in V^* and get $\widetilde{V^*}$. Then we get the new matrix$\widetilde{M}$.

$$\widetilde{M} = \widetilde{U}\widetilde{\Sigma}\widetilde{V^*}$$

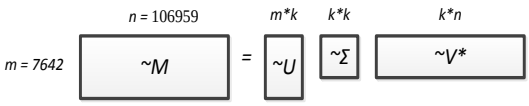

Figure 5

Then, if we want to predict how user A rate movie x, we can simply find the rate in the A-th row and x-th column of the $\widetilde{\mathbf{M}}$. When doing the recommendation for user ***A***, we just need to first find all entries in A-th row of $\widetilde{\mathbf{M}}$, then find the movies which are not rated before and pick out the top five as its recommendations.
The formula is:

$P(u)_i$ = M[u][i] if u-th user have already rated i-th movie
= $\widetilde{M}$[u][i] if u-th user have not rated i-th movie

After implementation of this approach, we did some modification.

First, we compute the new M, called M'
M'[i][j] = M[i][j] – average rate of j-th movie.

Then, we use SVD to get $\widetilde{M'}$, this step is the same as the original one.

Third, we can predict how user u rate movie i by the following formula:

1.If u-th user have already rated i-th movie:
$P(u)_i$ = M[u][i] + average rate of movie i
2.If u-th user have not rated i-th movie:
$P(u)_i$ = $\widetilde{M}$[u][i] + average rate of movie i

The result turns out to be better than the original one, and the RMSE is 0.2 less than the original one.

Since this algorithm is matrix related, we find it very difficult to be implemented by ourselves in a limited period of time, so we use some third party libraries including "scipy", "numpy" and "sparsesvd".

EXPEROMENT & EVALUATION

For every user in the data set, we will predict rating of the movies which this user has not seen before and we will compare these predicted ratings with the ground true ratings in the test set. Because some rating of the movie cannot be predict, if these movie exist in the test set we will ignore them. The evaluation criterions we use in this project are MAE and RMSE.

$$\boldsymbol{MAE} = \frac{\mathbf{1}}{\boldsymbol{n}} \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} |\boldsymbol{predicted_i} - \boldsymbol{groundTrue_i}|$$

$$\boldsymbol{RMSE} = \sqrt{\frac{\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} (\boldsymbol{predicted_i} - \boldsymbol{groundTrue_i})^2}{\boldsymbol{n}}}$$

N is the number of the movies in the whole cooperation, predicted_i is the predicted rating of a movie to a specified user; groundTrue_i is the rating of this movie to this user in the test set.

Experiment Design

For K-NN we need to figure out which k will lead to best performance, therefore we run the program multi-times with k from 5 to 205, interval is 10.

For weighted slop one, we simply run the program once and record the result.

For SVD, we are not sure about the number of feature we need to choose, therefore we run the program multi-times with k within (2~50, interval is 10) , (50~500, interval is 50) and (1000~5000, interval, is 1000).

Evaluation for accuracy

Results for k–NN:
Because we don't know which k is better, therefore we run k-NN multi-times the result is as follows:

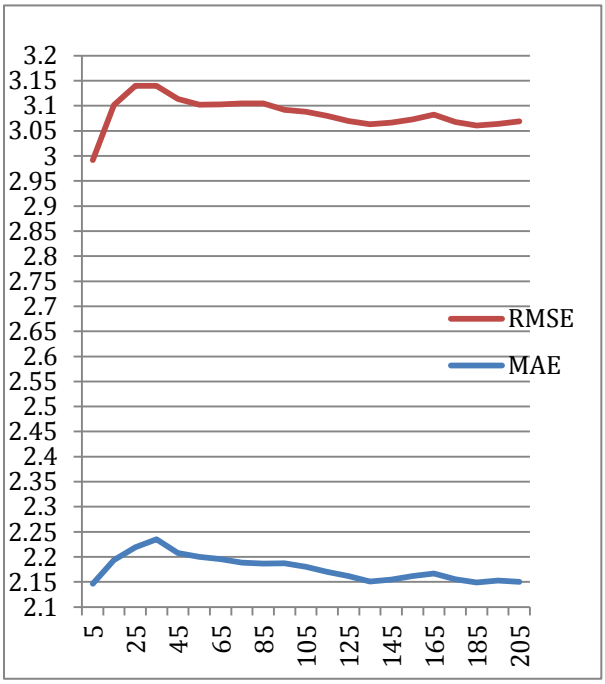

Figure 6

In figure 6, we see that when k is small (about 5), the accuracy is better, we think the reason is that the neighbors selected are very close to the specified user, so that the result is more accurate.
When k increasing from 5 to 35, the accuracy becomes worse, because the neighbors far away from the user were taken into consideration (but these neighbors still rate some movies which also rated by the user). After about 35, the accuracy becomes better slightly, because more and more far away users who don't rate any movie which also rated by the specified user were taken into consideration, therefore the accuracy will be slightly better and will tend to be flat.

Results for weighted slop one:

MAE = 2.0981000612
RMSE = 2.91225010734

Results for SVD:
Because we don't know the number of features we need to choose, we also run svd multi-times, the result is as follows:

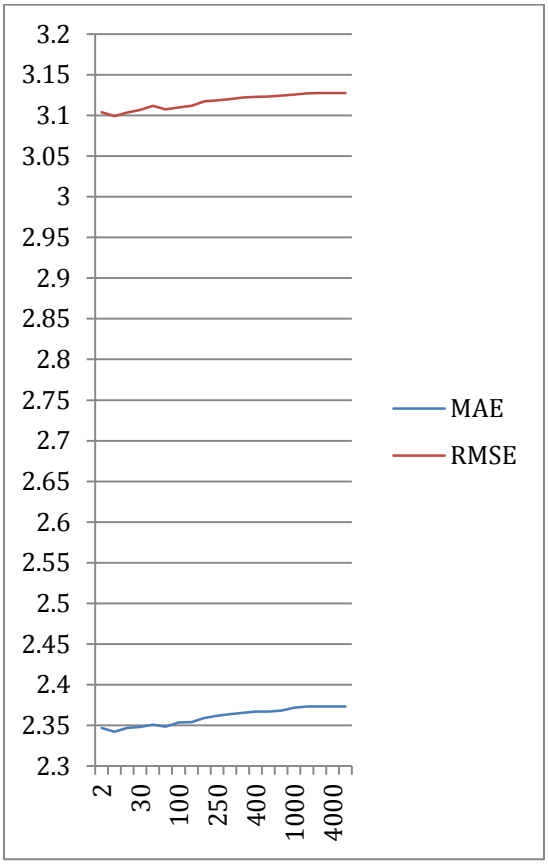

Figure 7

In figure 7, we see that when the feature number is small (about 10), the accuracy is better, which maybe because other features are not such importance as the 10 features, and this features which have some bad effect on the accuracy.

Here is the comparison of accuracy of the 3 approaches we use.

	K-NN	Slope one	SVD
MAE	2.147	2.098	2.342
RMSE	2.991	2.912	3.099

Figure 8

We can see that slope one is better than other approaches in our project.

Evaluation for numbers of movies can predicted

For K-NN:
Here is the chart of the relation between number of movies can be predicted and k.

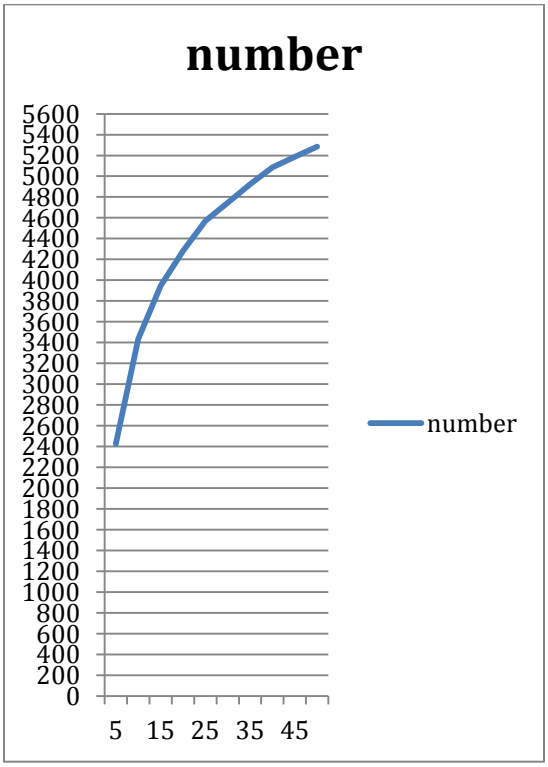

Figure 9

We can see that the number increases when k become bigger. The reason is that when k become bigger, more neighbors will be taken into consideration, and more movies that the specified

user doesn't rate will be found, therefore, more movies will be predicted.

For slope one, the number of movies can be predicted is 7849, which is much more than the number of K-NN.

For SVD, the number is 10136, which means it can predict every movies in the test set (the number of movies in test set is 10136)

Evaluation for performance of the 3 approaches

Here is the comparison of the performance of the 3 approaches in evaluation regarding 2308 users in test data set.

	K-NN	Slope one	SVD
Time(s)	1641	7823	877

Figure 10

Here is the comparison of the performance of the 3 approaches in recommendation, which is average time of doing recommendation for 1000 users.

	K-NN	Slope one	SVD
Time(ms)	753	2784	13

Figure 12

According to the 2 figures above, we can see that SVD is the fastest, because it will first predict all the movies, and store them in a matrix, when recommending movies for one user, it simply get the value in the matrix, the time complexity is O(1)

Slope one takes more time than others, because when recommending movies for one user, the time complexity is $O(n^2)$.

The time complexity of recommending movies for one user in K-NN is O(n), therefore it takes less time than slope one.

Here is the comparison of the time in preprocessing in 3 approaches. We repeatedly do the experiment for 50 times and get the average time.

	K-NN	Slope one	SVD
Time(ms)	1047	2355	137236

Figure 11

Because K-NN and slope one are both lazy algorithm, the preprocessing of the two are much faster than SVD.

CONCLUSION

In this course project, our team has learned the basic concept of a recommender system, and also implemented three approaches to build our models,

REFERENCES

[1] Ron Zacharski, http://guidetodatamining.com

[2] Eyrun A. Eyjolfsdottir, Gaurangi Tilak, Nan Li, ***"MovieGEN: A Movie Recommendation System"***

[3] Daniel Lemire, Anna Maclachlan, ***"Slope One Predictors for Online Rating-Based Collaborative Filtering"***

[4] Alan Kaylor Cline, Inderjit S. Dhillon , ***"Computation of the Singular ValueDecomposition"***

[5] JONATHAN L. HERLOCKER, JOSEPH A. KONSTAN, LOREN G. TERVEEN, JOHN T. RIEDL, ***"Evaluating Collaborative Filtering Recommender Systems"***

[6] Kaggle Wiki, ***Root Mean Squared Error (RMSE)***

TASK DISTRIBUTION FORM

Task	People
1. Collecting and preprocessing data	Shuai, Cuisheng
2. Implementing Algorithm 1	Xiaoyuan, Shuai
3. Implementing Algorithm 2	Xiaoyuan, Cuisheng, Qi
4. Implementing Algorithm 3	Shuai, Qi
5. Evaluating and comparing algorithms	Xiaoyuan, Cuisheng
6. Write Report	Xiaoyuan, Qi

ADDITIONAL POINTS

1. K-NN and SlopeOne Algorithms are realised by ourselves with improvements. We implemented SVD using third party tool in python, but the function package does not provide data set parser and converted matrix implementation. So we have also written several helper functions and predict function to make SVD work. We also compare the accuracy and efficiency of different input parameter (r, the size the middle matrix size) and pick the best r value.
2. After finishing the original baseline of two algorithms, K-NN and SVD, we also implemented SlopOne as our third baseline, the accuracy and predict coverage is pretty good.
3. After we finish the algorithm, we found there are several limitation about out training data set (the number of instances is not big enough; there is lots of noise in the data set). We write a crawler to get more movie and user information on the yahoo movie website. On the other side, we make full use of the two ratings of the user (one is from 1 to 5 and the other is from 1 to 13) to get rid of the some dirty data. What's more, we compare user's rating with the average rating of every movie. If the user's rating is far away from the average rating, we give less weight to this user. On the other hand, we build our own test data set by randomly selecting users and then we separate the movies of selected user into two parts. One half is regarded as example set and the other half is regarded as test.
4. We create our own KNN implementation which is quite suitable for recommendation system. Compared with the original KNN, the new implementation is more efficient and more accurate. Because it combine the accuracy of the predict score and value score. Instead of predict all the movie score for the user, the new algorithm only compute value for high accurate scores and only recommend movies with high score and high accurate score to user.