Movie Recommendation System

*EE695:Applied Machine Learning, Team:10

1st Fan Yang   2nd Zeliang Liu   3nd Liwang Zhou

***Abstract*—This work implements a movie recommendation system based on Movie-Lens database using three methods.**

***Index Terms*—Recommendation System, ItemCF, UserCF, MF**

I. INTRODUCTION(AUTHOR: FAN YANG)

With the continuous updating of computer technology in the current Internet environment, more functions and applications are provided to users, bringing a lot of convenience and entertainment to people's lives. Meanwhile, the data of the Internet has also grown exponentially, which has also brought problems. How to select the desired results from the huge Internet data? More specifically, When people want to watch movies, how to choose the movie that best matches users' demand?

The recommendation system is one of the most effective ways to solve this current problem. The recommendation system literally means recommending items to the user, which can be a product, news or movie. Objectively, it uses to help users filter some irrelevant information, and only display information in the user's interest.

In this work, we propose a movie recommendation system to give a movies recommendation list which meets users' interest. We specifically analyze three algorithms based on the MovieLen dataset, respectively. At last, we do implementation, improvement, and comparison.

II. RELATED WORK(AUTHOR: FAN YANG)

Recommendation algorithms can be roughly divided into three categories: content-based recommendation algorithms, collaborative filtering recommendation algorithms, and knowledge-based recommendation algorithms.

One approach to the design of recommender systems that has wide use is collaborative filtering. John S. Breese, David Heckerman, Carl Kadie described several collaborative filtering algorithms designed for predict additional topics or products a new user might like, including techniques based on correlation coefficients, vector-based similarity calculations, and statistical Bayesian methods in 1998.[1] Also, many algorithms have been used in measuring user similarity or item similarity in recommender systems. For example, the k-nearest neighbor (k-NN) approach. Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl investigated the use of dimensionality reduction to improve performance for a new class of data analysis software called "recommender systems" in 2000.[2]

Another common approach when designing recommender systems is content-based filtering. Content-based filtering methods are based on a description of the item and a profile of the user's preferences. George Lekakos, Petros Caravelas proposed a hybrid approach based on content-based and collaborative filtering, implemented in MoRe, a movie recommendation system.[3]

The final approach is knowledge-based recommendation algorithms. Brendon Towle and Clark Quinn perceived an opportunity for knowledge-based recommender systems to gain leverage on recommendation tasks by using explicit models of both the user of the system and the products being recommended.[4]

III. ITEMCF AND TOP-K ITEMCF (AUTHOR: FAN YANG)

Collaborative Filtering is the process of filtering or evaluating items using the opinions of other people.[5] It can be used in many tasks. Recommend items, show a list of items to a user, in order of how useful they might be. Or predict for a given item, given a particular item, calculate its predicted rating, and prediction can be more demanding than a recommendation.[6]

In this work, I use an item-based collaborative filtering algorithm based on the MovieLen database to recommend a movie list for given users.

A. Dataset Preprocess and Description

I use MovieLens database (ml-20m version) in this work. It describes 5-star rating and free-text tagging activity from-MovieLens, a movie recommendation service. It contains 20000263 ratings and 465564 tag applications across 27278 movies. These data were created by 138493 users between January 09, 1995 and March 31, 2015. This dataset was generated on October 17, 2016.

This dataset containes six files: genome-scores.csv, genome-tags.csv, links.csv, movies.csv, ratings.csv and tags.csv. In my work part, I only use ratings.csv and movies.csv. Movies.csv is used to find the corresponding characteristics of the movie. (moveis Id, titles, genres)

```
movies_title = ['movieId', 'title', 'genres']
movies = pd.read_csv('/Users/Lethe/Desktop/EE695-MachineLearning/final_research/ml-20m/movies.csv', encoding='latin-1',usecols=movies_title)
```

Algorithm 1: Python-extract movies.csv

1 *Department of Electrical and Computer Engineering, Stevens Institute of Technology, Hoboken, United States*

I extract partial data in ratings.csv, including 1000 users, 10,000 movies, 133334 ratings.

```
ratings_df = pd.read_csv(rating_file)
ratings_df = ratings_df[ratings_df.userId < 1001]
ratings_df = ratings_df[ratings_df.movieId < 10001]
ratings_df.head()
```

Algorithm 2: Python-extract ratings.csv

B. Introduction to ItemCF and Top-K itemCF algorithms

At the starting stage, I create a User-Item similarity matrix, and divide the Preprocessed database into training part and testing part. After this, I use a collaborative filtering algorithm to recommend movie list for the given users.

1) Item-based collaborative filtering algorithm

Item-based collaborative filtering algorithm (ItemCF) recommends to users items that are similar to their favorite items. However, the ItemCF algorithm does not use the content attributes of items to calculate the similarity between items. It mainly calculates the similarity between items by analyzing user behavior records. The algorithm believes that the similarity between item A and item B depends on most users who like item A also like item B. The collaborative filtering algorithm based on items can use the user's historical behavior to provide recommendation explanations for the recommendation results.[7]
The item-based collaborative filtering algorithm is mainly divided into two steps:

- Calculate the similarity between items.

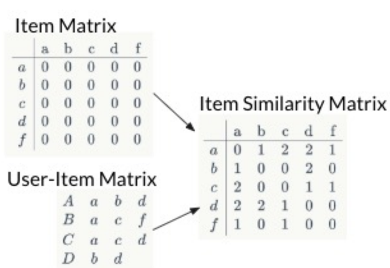


Fig. 1: Example for calculating item similarity

- Generate a recommendation list for users based on the items' similarity and the user's historical behavior.
  I calculate the items' similarity by cosine similarity:

$$sim(i,i') = \frac{r_i r'_i}{|r_i||r'_i|} = \sum_i \frac{r_{ui} r'_{ui}}{\sqrt{\sum_i r_{ui}^2}\sqrt{\sum_i {r'_{ui}}^2}} \tag{1}$$

2) Item-based collaborative filtering algorithm

In order to reduce RMSE, I introduce the Top-k Collaborative Filtering algorithm. This method calculate only the first K users that are most similar to the given user.

3) Evaluation section

The prediction accuracy of the score prediction is generally calculated by the Root Mean Squared Error. For an item in the testing set, there are the user's actual ratings of the item and the predicted ratings given by the recommendation algorithm, thus, RMSE can be defined as:

$$RMSE = \sqrt{\frac{\sum_{u,i\epsilon T}(r'_{ui} - r_{ui})^2}{|T|}} \tag{2}$$

r'_{ui}:Predict rating, r_{ui}:Actual rating
For the Top-k Collaborative Filtering algorithm, I add precision, recall, and coverage for evaluation metrics, due to the choice of k can affect the results. In this work, I test the values of these evaluation metrics based on different k values, in order to find the appropriate k value.

C. Implementation process

1) Compare itemCF and Top-k itemCF:

Create a User-Item similarity matrix:

```
max_user_id = ratings_df.userId.max()
max_movie_id = ratings_df.movieId.max()
ratings = np.zeros((max_user_id, max_movie_id))
ratings[ratings_df.userId.as_matrix()-1, ratings_df.
    movieId.as_matrix()-1] = ratings_df.rating.
    as_matrix()
ratings
```

Algorithm 3: Python-User-Item matrix

Divide the Preprocessed database into training part and testing part:

```
def split_data(ratings):
    test = np.zeros(ratings.shape)
    train = ratings.copy()
    for user in range(ratings.shape[0]):
        if len(ratings[user,:].nonzero()[0]) > 20:
            test_ratings = np.random.choice(ratings[
    user,:].nonzero()[0],size=10,replace=False)
            train[user,test_ratings] = 0
            test[user, test_ratings] = ratings[user,
     test_ratings]
    assert(np.all((train * test) == 0))
    return train, test
train, test = split_data(ratings)
```

Algorithm 4: Python-Split dataset

Calculate the items' similarity:

```
def cosine_similarity(ratings, kind='item', epsilon
    =1e-9):
    sim = np.dot(ratings.T, ratings)
    sim += epsilon
    norms = np.array([np.sqrt(np.diagonal(sim))])
    return (sim / norms / norms.T)
item_similarity = cosine_similarity(train, kind='
    item')
```

Algorithm 5: Python-item similarity

Calculate prediction ratings based on itemCF:

```
def predict(ratings, similarity, kind='item'):
    norm = np.array([np.abs(similarity).sum(axis=1)
    ])
    return np.dot(ratings, similarity) / norm
```

Algorithm 6: Python-itemCF

Calculate prediction ratings based on Top-k itemCF:

```python
def predict_topk(ratings, similarity, kind='item',k=30):
    pred = np.zeros(ratings.shape)
    for j in range(ratings.shape[1]):
        top_k_items = [np.argsort(similarity[:,j])[:-k-1:-1]]
        for i in range(ratings.shape[0]):
            pred[i,j] = similarity[j,:][top_k_items].dot(ratings[i,:][top_k_items].T)
            pred[i,j] /= np.sum(np.abs(similarity[j,:][top_k_items]))
    return pred
```

Algorithm 7: Python-Top-k itemCF

2) Test Top-k Collaborative Filtering algorithm with different k values:

```python
k_array = [10, 20, 40, 60, 120, 160]
item_test_mse = []
item_train_mse = []
for a in k_array:
    item_pred = predict_topk(train, item_similarity, kind='item', k=a)
    item_train_mse.append(get_rmse(item_pred, train))
    item_test_mse.append(get_rmse(item_pred, test))
```

Algorithm 8: Python-Top-k-itemCF(different K values)

D. Comparison and Improvement

1) ItemCF vs Top-k itemCF

Comparing ItemCF and Top-k itemCF, it is obvious that RMSE decreases when using Top-k itemCF (k = 30).

```
Item-based CF RMSE: 3.618168592626016

Top-k Item-based CF RMSE: 3.264726105051021
```

Fig. 2: Evaluating parameters based on K values.

2) Different values for Top-k Item-based Collaborative Filtering algorithm

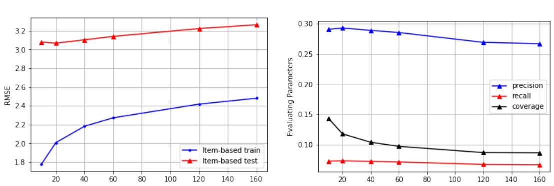


Fig. 3: Evaluating parameters based on K values.

In the first figure, the precision and recall are not linearly corrlated to the parameter k, However, selecting the appropriate k is important for obtaining high precison of this recommendation system.
In the second figure, the larger k, the smaller the coverage will be.
Obviously, when k = 20, precision and recall are optimal. So 20 is the best selection for k value.

3) Final movie recommendation list for corresponding users.

For a given user, a number of movies can be recommend for him, and the recommendation movie list is sorted by rating.
I list the movie recommendation list for user 33 (movies number = 10) and user 789 (movies number = 20). In addition, based on movies.csv, I extract the specific information of each movie and show it in a table.

```
The recommended list for user 789 is:
('1198', 12.103900468164218)
('2797', 10.572191997814306)
('4306', 10.426319868014739)
('1210', 10.038586336654133)
('4963', 8.894413286564674)
('1265', 8.784430112609233)
('1196', 8.783630577769204)
('2716', 8.29948952203735)
('3527', 8.295396410555604)
('1036', 8.133200328619754)
```

	movieId	title	genres
0	1198	Raiders of the Lost Ark (Indiana Jones and the...	Action|Adventure
1	2797	Big (1988)	Comedy|Drama|Fantasy|Romance
2	4306	Shrek (2001)	Adventure|Animation|Children|Comedy|Fantasy|Ro...
3	1210	Star Wars: Episode VI - Return of the Jedi (1983)	Action|Adventure|Sci-Fi
4	4963	Ocean's Eleven (2001)	Crime|Thriller
5	1265	Groundhog Day (1993)	Comedy|Fantasy|Romance
6	1196	Star Wars: Episode V - The Empire Strikes Back...	Action|Adventure|Sci-Fi
7	2716	Ghostbusters (a.k.a. Ghost Busters) (1984)	Action|Comedy|Sci-Fi
8	3527	Predator (1987)	Action|Sci-Fi|Thriller
9	1036	Die Hard (1988)	Action|Crime|Thriller

Fig. 4: Movie Recommendation list for user 789

```
The recommended list for user 33 is:
('3527', 9.129648614069154)
('3793', 8.570574358354628)
('2001', 7.485862447022449)
('1580', 7.018807224690521)
('1036', 7.016754097529721)
('2916', 7.009641808556909)
('2797', 6.872599953556426)
('1196', 5.799693515864043)
('1210', 5.739257544212128)
('1240', 5.6621076570927436)
('1374', 5.3425000346779115)
('2628', 5.233192640226764)
('2987', 5.197516577105603)
('3114', 4.983417861075084)
('1291', 4.731077449394513)
('2918', 4.597165275336584)
('1200', 4.596121756785891)
('1198', 4.485044105881644)
('3175', 4.290128696861554)
('4636', 4.154964693347801)
```

	movieId	title	genres
0	3527	Predator (1987)	Action|Sci-Fi|Thriller
1	3793	X-Men (2000)	Action|Adventure|Sci-Fi
2	2001	Lethal Weapon 2 (1989)	Action|Comedy|Crime|Drama
3	1580	Men in Black (a.k.a. MIB) (1997)	Action|Comedy|Sci-Fi
4	1036	Die Hard (1988)	Action|Crime|Thriller
5	2916	Total Recall (1990)	Action|Adventure|Sci-Fi|Thriller
6	2797	Big (1988)	Comedy|Drama|Fantasy|Romance
7	1196	Star Wars: Episode V - The Empire Strikes Back...	Action|Adventure|Sci-Fi
8	1210	Star Wars: Episode VI - Return of the Jedi (1983)	Action|Adventure|Sci-Fi
9	1240	Terminator, The (1984)	Action|Sci-Fi|Thriller
10	1374	Star Trek II: The Wrath of Khan (1982)	Action|Adventure|Sci-Fi|Thriller
11	2628	Star Wars: Episode I - The Phantom Menace (1999)	Action|Adventure|Sci-Fi
12	2987	Who Framed Roger Rabbit? (1988)	Adventure|Animation|Children|Comedy|Crime|Fant...
13	3114	Toy Story 2 (1999)	Adventure|Animation|Children|Comedy|Fantasy
14	1291	Indiana Jones and the Last Crusade (1989)	Action|Adventure
15	2918	Ferris Bueller's Day Off (1986)	Comedy
16	1200	Aliens (1986)	Action|Adventure|Horror|Sci-Fi
17	1198	Raiders of the Lost Ark (Indiana Jones and the...	Action|Adventure
18	3175	Galaxy Quest (1999)	Adventure|Comedy|Sci-Fi
19	4636	Punisher, The (1989)	Action

Fig. 5: Movie Recommendation list for user 33

E. Future Work

In future work, I will consider some special cases to improve the presicion of recommendation.
Actually, some users always give higher ratings while others give lower ratings to a same movie. For example, for the same bad movie, someone may give 1 point, while Others may give 3 points for their efforts. This caused inaccurate rating predictions. Hence, this cosine similarity can be modified to eliminate this effect.[7]

$$r'_{ui} = \frac{\bar{r_u} + \sum_{u'} sim(u,u')(r_{u'i} - \bar{r_{u'}})}{\sum_{u'} |sim(u,u')|} \tag{3}$$

IV. UserCF and top-k UserCF (author: Liwan Zhou)

A. Dataset Preprocess and Description

This chapter introduces and evaluates various algorithms using the MovieLens dataset provided by GroupLens. This dataset (ml-20m) describes 5-star rating and free-text tagging activity from MovieLens, a movie recommendation service. It contains 20000263 ratings and 465564 tag applications across 27278 movies. These data were created by 138493 users between January 09, 1995 and March 31, 2015.
UserCF algorithm only use the rating.csv.

B. Introduction to UserCF

The user-based collaborative filtering algorithm consists of two steps.
(1) find a set of users with similar interests to the target users.
(2) find the movies that users like in this set and recommend them to the target users.

C. Implementation process

The key to first step is to calculate the similarity of interest between the two users. Here, the collaborative filtering algorithm mainly USES the similarity of behavior to calculate the similarity of interest. Given user u and user v, let N(u) represent the set of movies that user u has had positive feedback on, and let N(v) represent the set of movies that user v has had positive feedback on. Then, we can simply calculate the interest similarity of u and v through the following cosine similarity formula:

$$w_{1vv} = \frac{|N(u) \cap N(v)|}{\sqrt{N(u)||N(v)|}} \tag{4}$$

In fact, many users don't rate the same movie, which is $|N(u) \bigcap N(v)| = 0$. We can first calculate the user pair (u,v) of $|N(u) \bigcap N(v)| \neq 0$, and then divide that by the denominator of $\sqrt{|N(u)||N(v)|}$. The code is as follow:

```
def userSimilarityBest(self, train = None):
    train = train or self.traindata
    self.userSimBest = dict()
    #build inverse table for item_users
    item_users = dict()
    for u,item in train.items():
        for i in item.keys():
            item_users.setdefault(i,set())
            item_users[i].add(u)
    #calculate co-rated items between users
    user_item_count = dict()
    count = dict()
    for item, users in item_users.items():
        for u in users:
            user_item_count.setdefault(u,0)
            user_item_count[u] += 1
            for v in users:
                if u == v:continue
                count.setdefault(u,{})
                count[u].setdefault(v,0)
                count[u][v] += 1
    #calculate finial similarity matrix
    for u ,related_users in count.items():
        self.userSimBest.setdefault(u,dict())
        for v, cuv in related_users.items():
            self.userSimBest[u][v] = cuv / math.sqrt
    (user_item_count[u] * user_item_count[v] * 1.0)
```

Algorithm 9: Python-UserCF

After obtaining the user similarity, the UserCF algorithm will recommend to the user the movies preferred by K users with the most similar interests, which will be achieved by the following formula:

$$p(u,i) = \sum_{v \in S(u,K) \cap N(i)} w_{1v} r_{ij} \tag{5}$$

The code is as follow:

```
def recommend(self,user,train = None,k = 8,nitem =
    10):
    train = train or self.traindata
    rank = dict()
    interacted_items = train.get(user,{})
    for v ,wuv in sorted(self.userSimBest[user].
    items(),key = lambda x : x[1],reverse = True)[0:
    k]:
        for i , rvi in train[v].items():
            if i in interacted_items:
                continue
            rank.setdefault(i,0)
            rank[i] += wuv
    return dict(sorted(rank.items(),key = lambda x :
    x[1],reverse = True)[0:nitem])
```

Algorithm 10: Python-Top-k-userCF

Result:

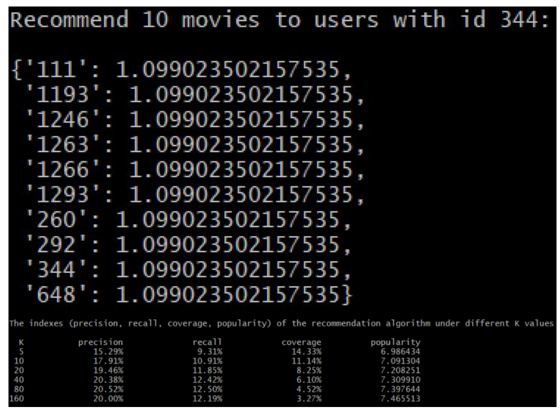


Fig. 6: Result

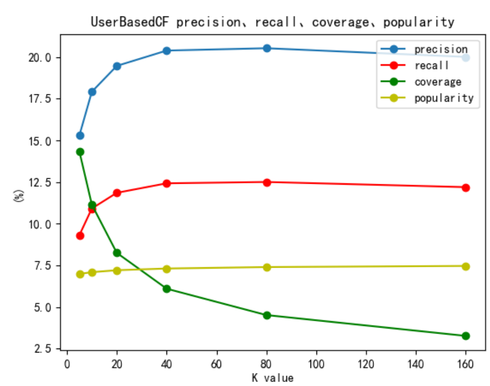


Fig. 7: Evaluation metrics-2

D. Comparison and Improvement

precision and recall: It can be seen that the evaluation indicators (precision and recall) of the recommendation system do not have a linear relationship with the parameter K. In the MovieLens dataset, choosing K=80 yields higher accuracy and recall. Therefore, it is important to select the right K to obtain high precision of the recommendation system.

Popularity: As we can see, the larger the K is, the more popular the UserCF recommendation results will be. This is because K determines how many other users' interests are similar to yours that UserCF will take into consideration when making recommendations to you. If K is large enough, the result will be globally popular movies.
Coverage: the larger the K, the lower the coverage will be.The decrease in coverage is due to the increase in popularity. With the increase in popularity, UserCF algorithm is more and more inclined to recommend popular movies, thus recommending less and less long-tail items, resulting in the decrease in coverage.

E. Future Work

The simplest formula for calculating user similarity matrix (cosine similarity formula) is presented above, but this formula is too rough and can be modified to improve the recommended performance of UserCF.
The formula is changed to:

$$w_{uv} = \frac{\sum_{i \in N(u) \cap N(v)} \frac{1}{\log 1 + |N(i)|}}{\sqrt{|N(u)||N(v)|}} \tag{6}$$

The formula punishes the influence of popular movies on their similarity in the list of common interests of user u and user v by $\frac{1}{\log 1 + |N(i)|}$.

V. Matrix Factorization (Author: Zeliang Liu)

A. Dataset Preprocess and Description

The dataset contains two files movies.csv and ratings.csv. movies.csv stores movie id and movie name and movie genres. The ratings.csv le stores movie id, user id, the rating and timestamp. I decided to split the data into training and test set. 70 percent of dataset is training set and 30 percent is test set.

B. Introduction to Matrix Factorization

Traditionally speaking, user-based and item-based collaborative filtering techniques are computationally and theoretically simple to implement and understand, but matrix factorization is more effective. Because matrix factorization allows us to discover the latent features, underlying the interactions between users and items. Given that each user would rate the items that they have not yet rated, such that we can make recommendations to users.
For the matrix factorization, we have to predict user ratings of movies based on the ratings that we have.
Rating set forms a matrix (users x movies) which we factorize into V(users x rank) and W (movies x rank) where V is represented of user-data and W is represented of movie-data. And we try to find the product of these two matrices: VW^T is as close to X as possible.[8]
There are several optimization algorithms can be used to optimize the cost function, and I use the stochastic gradient descent algorithm.

$$E = \sum_{i,j} ((R_{i,j} - (VW^T)_{i,j})^2 + \lambda(||v||^2 + ||W||^2)$$

Fig. 8: cost function

First, the cost function looks like below:
The first term is the squared error per (user, movie) pair. And the second term is a regularization parameter that we are trying to learn values in V and W matrices don't overfit to the training data. This particular is the L2 regularization as it adds the euclidean norm for matrices to the cost function. One can also use L1 norm as well to achieve L1-regularization.[9]
Stochastic Gradient Descent (SGD) is also know as incremental gradient descent. It is a stochastic approximation of the gradient descent optimization and iterative method for minimizing an objective function. Briefly explaining the SGD process, this optimization algorithm tries to look for the direction in which we should move out parameter values to observe a reduction in the cost function.[10] However, by calculating the gradient at a point and subtracting the product of gradient and learning rate from the parameter values, we might end up at a local minimization. To overcome this, random initialization of the parameter values often helps. The learning rate is a very important parameter, since high learning rate means that we are taking wider steps and might miss the optimum value, and low learning rate means that we are taking small steps and it might take a long time.
Subtracting the above gradients from V[i] and W[j] respectively after multiplying the gradients with learning rate, we get the following update rule:

$$V_i = V_i + \alpha[(R_{i,j} - (VW^T)_{i,j})(W_j) - \lambda V_i]$$
$$W_i = W_i + \alpha[(R_{i,j} - (VW^T)_{i,j})(V_j) - \lambda W_i]$$

Fig. 9: update rule

For testing the performance of our model, we split our dataset into training and test set and optimize our cost function by making updates while looking through the ratings in the training set. The test set is then used to evaluate our learnt model which in this case is V and W matrices. The evaluation metrices used are RMSE which stands for Average Root mean square error.[10]

C. Implementation process

In order to save times. I slice the ratings dataset. Since the ratings dataset is sorted according to the userId, for example, if I want to make recommendation to user 9, I can only choose the first 100 000 rows. I think that it is more informative to have more ratings on a smaller number of users than smaller numbers of ratings per user on a larger subset of users.
In order to minimize the Root-mean-square-error, wo should the best regularization parameter and rank k.
'Rank' is the number of latent features that we are trying to learn of the interaction between users and movies. 'lambd' is the regularization parameter which controls the degree

of regularization we perform during the training process by preventing the values in V and W matrices to obtain really high values, as the objective of the optimization process is to reduce the overall error and lambda multiplied with the sum of L2-norm of the V and W matrices is a positive term that we add to the squared error.[11] We are searching over the values of 'rank' and 'lambda' to find the pair of these values which gives the most optimal performance.
This part of the code is for calculating RMSE to find best rank and regularization paremeter.

```
lmbda_range = [0.01, 0.02, 0.05, 0.1, 0.2, 0.5]
k_range  = [5, 10, 20, 40, 80, 120]
m, n = train_data_matrix.shape
iterations = 100
alpha = 0.005
for lmbda in lmbda_range:
    for k in k_range:
        user_feature = np.random.rand(k, m)
        movie_feature = np.random.rand(k, n)
        train_errors = []
        test_errors = []
        users, items = train_data_matrix.nonzero()
        for iter in range(iterations):
            for u, i in zip(users, items):
                e = train_data_matrix[u, i] - predict(user_feature[:, u], movie_feature[:, i])
                user_feature[:, u] += alpha * (e * movie_feature[:, i] - lmbda * user_feature[:, u])
                movie_feature[:, i] += alpha * (e * user_feature[:, u] - lmbda * movie_feature[:, i])

            pred = predict(user_feature, movie_feature)
            train_rmse = rmse(pred, train_data_matrix)
            test_rmse = rmse(pred, test_data_matrix)
            train_errors.append(train_rmse)
            test_errors.append(test_rmse)

        pred = predict(user_feature, movie_feature)
        sgdBasedTrainRMSE = rmse(pred, train_data_matrix)
        sgdBasedTestRMSE = rmse(pred, test_data_matrix)
        trainRMSE.append(sgdBasedTrainRMSE)
        testRMSE.append(sgdBasedTestRMSE)
```

Algorithm 11: Python-Matrix Factorization

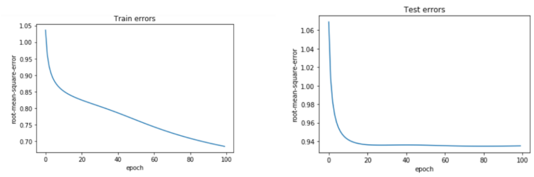
Result:

lmbda = 0.01			lmbda = 0.1		
K	train RMSE	test RMSE	K	train RMSE	test RMSE
5	0.675090912	0.989938609	5	0.738578	0.952103275
10	0.57014179	1.028065423	10	0.686099034	0.931224396
20	0.438293846	1.078425049	20	0.637355513	0.94126953
40	0.303768549	1.12213102	40	0.613572429	0.989866308
80	0.238805546	1.172678168	80	0.586284178	1.011922848
120	0.22231059	1.220482024	120	0.570171081	1.020465254
lmbda = 0.02			lmbda = 0.2		
k	train RMSE	test RMSE	k	train RMSE	test RMSE
5	0.683905056	0.983507994	5	0.820136393	0.96643696
10	0.585650216	1.005146673	10	0.807657267	0.944032449
20	0.45859736	1.041986299	20	0.800967953	0.954780754
40	0.353188986	1.066614324	40	0.8173092	1.018690543
80	0.285362377	1.117193374	80	0.807677635	1.052605873
120	0.284064295	1.139213542	120	0.794447817	1.052622116
lmbda = 0.05			lmbda = 0.5		
K	train RMSE	test RMSE	k	train RMSE	test RMSE
5	0.704329953	0.968153452	5	0.959581337	1.068964471
10	0.617413756	0.968355206	10	0.960119274	1.047263046
20	0.527127126	0.979647081	20	0.966324967	1.052784296
40	0.46001131	1.007002362	40	0.992737771	1.126237493
80	0.422063989	1.044691045	80	1.055638308	1.262830581
120	0.409928613	1.052020262	120	1.041753079	1.247619194

Fig. 10: Result

D. Comparison and Improvement

From the result, when lmbda is equal to 0.01 and the rank is equal to 100, the train RMSE is the smallest, but the test RMSE is very big. Also, when lmbda is equal to 0.2 and rank is equal to 20, the test RSME is the smallest but the train RMSE is big. Synthesizes all the result, when regularization parameter lmbda is equal to 0.1, and the rank is equal to 20, I can get the best train and test RMSE:



Train RMSE = 0.6860990340403824
Test RMSE = 0.9312243960781807

Fig. 11: Comparison Result

Taking user 9 as an example, I can make movie recommendation:

```
Movies recommended to user 9 are:
Sunshine Boys, The (1975)
Tron (1982)
Anger Management (2003)
Heathers (1989)
Animal Crackers (1930)
Eagle Eye (2008)
Defiant Ones, The (1958)
Ben X (2007)
Silver Linings Playbook (2012)
Neighbors (2014)
```

Fig. 12: Comparison Result

E. Future Work

In future work, I will consider the entire rating dataset. For example, I can set iteration is 2 and set larger range of rank k and regularization term lmbda to find the best parameters. Since the rating dataset just has positive samples (only what the user likes) and does not have negative samples (what the user is not interested in), I will implement a few methods to generate negative samples for each user. That will improve the accuracy of the recommended system.[12]

VI. Conclusion(Author: Fan Yang)

This work constructed a movie recommendation system based on the MovieLen dataset. We implemented three recommendation algorithms, itemCF, userCF, and matrix factorization. We did improvement of the current algorithms based on several evaluation metrics, respectively. Finally, we obtained a movie recommendation list based on users' interests. In future work, we will consider some special cases to improve the accuracy of the recommendation.

References

[1] ohn S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. 1998.

[2] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Application of dimensionality reduction in recommender system a case study. 2000.

[3] George L. and Petros C. A hybrid approach for movie recommendation. 2008.

[4] Brendon T. and Clark Q. Knowledge based recommender systems using explicit user model. 2000.

[5] Konstan J. A. Riedl J. Herlocker, J. L. Explaining collaborative filtering recommendations. 2000.

[6] Frankowski D. Herlocker J. Sen S. Schafer, J. B. Collaborative filtering recommender systems. 2007.

[7] Badrul S., George K., Joseph K., and John R. Item-based collaborative filtering recommendation algorithms. 2001.

[8] Sinno J. P. Qiang Y. Lili Z., Zhongqi L. Matrix factorization for movie recommendation. 2016.

[9] Albert A. Y. Matrix factorization: A simple tutorial and implementation in python. 2010.

[10] Vardhaman M. Movie recommendation using regularized matrix factorization, 2010. [Online; accessed 16-September-2010].

[11] Kilol G. Movie recommendation using regularized matrix factorization, 2010. [Online; accessed May-2018].

[12] Liang X. Recommended system practice, 2010. [Online; accessed October-2017].