Recommendation System Application Project Report

- LI HUANG

Part 1. Data description

https://grouplens.org/datasets/movielens/

with the dataset with size 100k. I am using this dataset as it is very detailed both in user and item info for analysis. There are three main dataset that are used in this project from this movie lens data collection, 'u.data','u.item','u.user','ua.test'. The size (rows, cols) of u.data, u.item and u.user datasets are (100000, 4), (1682, 24), (943, 5) respectively. The ua.test is the test dataset that set side for accuracy calculating in Part3.

The variables in the three datasets are:

```
dataCol = ['user id','movie id','rating','timestamp']
itemCol = ['movie id','movie title','release date','video release date','IMDb URL','unknown','Action',
'Adventure','Animation','Childrens','Comedy','Crime','Documentary','Drama','Fantasy','Film-Noir','Horror',
'Musical','Mystery','Romance ','Sci-Fi','Thriller','War' ,'Western']
userCol = ['user id','age','gender','occupation','zip code']
```

And here are how the three datasets look like:

u.data

```
   user id  movie id  rating  timestamp
0      196       242       3  881250949
1      186       302       3  891717742
2       22       377       1  878887116
3      244        51       2  880606923
4      166       346       1  886397596
5      298       474       4  884182806
6      115       265       2  881171488
7      253       465       5  891628467
8      305       451       3  886324817
9        6        86       3  883603013
(100000, 4)
```

u.item

	movie id	movie title	release date	video release date	IMDb URL	unknown	Action	Adventure	Animation	Childrens	...	Fantasy	Film-Noir	Horror	Musical	Mystery
0	1	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	0	0	0	1	1	...	0	0	0	0	0
1	2	GoldenEye (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?GoldenEye%20(...	0	1	1	0	0	...	0	0	0	0	0
2	3	Four Rooms (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Four%20Rooms%...	0	0	0	0	0	...	0	0	0	0	0
3	4	Get Shorty (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Get%20Shorty%...	0	1	0	0	0	...	0	0	0	0	0
4	5	Copycat (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Copycat%20(1995)	0	0	0	0	0	...	0	0	0	0	0

u.user

	user id	age	gender	occupation	zip code
0	1	24	M	technician	85711
1	2	53	F	other	94043
2	3	23	M	writer	32067
3	4	24	M	technician	43537
4	5	33	F	other	15213
5	6	42	M	executive	98101
6	7	57	M	administrator	91344
7	8	36	M	administrator	05201
8	9	29	M	student	01002

There is no missing values in the u.data and u.user dataset, and u.item have some missing values in three variables as below, but they are reference info and will not affect the implementation of our project, so we do not need to fulfill the missing values for this project.

```
item.isnull().any()
```

```
movie id               False
movie title            False
release date            True
video release date      True
IMDb URL                True
unknown                False
```

If we merge all the data together, we get a full dataset with rating, item info and user info as following:

```
all_data= pd.merge(pd.merge(user, data),item)
all_data[:10]
```

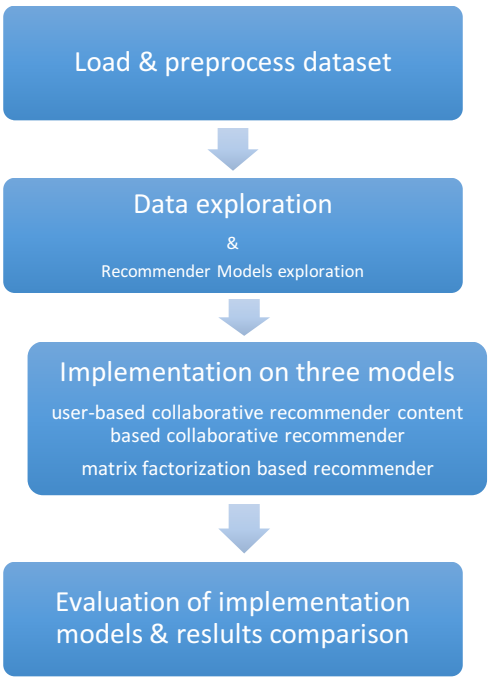
	user id	age	gender	occupation	zip code	movie id	rating	timestamp	movie title	release date	...	Fantasy	Film-Noir	Horror	Musical	Mystery	Romance	Sci-Fi	Thriller	War
0	1	24	M	technician	85711	61	4	878542420	Three Colors: White (1994)	01-Jan-1994	...	0	0	0	0	0	0	0	0	0
1	13	47	M	educator	29206	61	4	882140552	Three Colors: White (1994)	01-Jan-1994	...	0	0	0	0	0	0	0	0	0
						61	4	880130803	Three Colors: White (1994)	01-Jan-1994	...	0	0	0	0	0	0	0	0	0
3	58	27	M	programmer	52246	61	5	884305271	Three Colors: White (1994)	01-Jan-1994	...	0	0	0	0	0	0	0	0	0
4	59	49	M	educator	08403	61	4	888204597	Three Colors: White (1994)	01-Jan-1994	...	0	0	0	0	0	0	0	0	0
5	60	50	M	healthcare	06472	61	4	883326652	Three Colors: White (1994)	01-Jan-1994	...	0	0	0	0	0	0	0	0	0

pand output; double click to hide output

Part 2. Methodology and Implementation Process Exploration

In this project, I have processed the dataset with analysis and visualization tools from python library numpy, panda, matplotlib, scikit learn; performed classification, to be specific, K nearest neighbor classification; implemented user-based collaborative filtering recommender, item-based collaborative filtering recommender and matrix factorization; and evaluation the results.

Following is the flow chart of methodology and structure of this project:



- **Data exploration and rating pivot**

  data descriptions for the three datasets:

  dataset: data

  From the summary, we can see the min rating is 1, and all the 0 values in the pivot means not rated/missing.

	user id	movie id	rating	timestamp
count	100000.00000	100000.000000	100000.000000	1.000000e+05
mean	462.48475	425.530130	3.529860	8.835289e+08
std	266.61442	330.798356	1.125674	5.343856e+06
min	1.00000	1.000000	1.000000	8.747247e+08
25%	254.00000	175.000000	3.000000	8.794487e+08
50%	447.00000	322.000000	4.000000	8.828269e+08
75%	682.00000	631.000000	4.000000	8.882600e+08
max	943.00000	1682.000000	5.000000	8.932866e+08

  dataset: item

	movie id	video release date	unknown	Action	Adventure	Animation	Childrens	Comedy	Crime	Documentary	...	Fantasy
count	1682.000000	0.0	1682.000000	1682.000000	1682.000000	1682.000000	1682.000000	1682.000000	1682.000000	1682.000000	...	1682.00000
mean	841.500000	NaN	0.001189	0.149227	0.080262	0.024970	0.072533	0.300238	0.064804	0.029727	...	0.01308
std	485.695893	NaN	0.034473	0.356418	0.271779	0.156081	0.259445	0.458498	0.246253	0.169882	...	0.11365
min	1.000000	NaN	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.00000
25%	421.250000	NaN	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.00000
50%	841.500000	NaN	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.00000
75%	1261.750000	NaN	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	...	0.00000
max	1682.000000	NaN	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1.00000

  dataset: user

	user id	age
count	943.000000	943.000000
mean	472.000000	34.051962
std	272.364951	12.192740
min	1.000000	7.000000
25%	236.500000	25.000000
50%	472.000000	31.000000
75%	707.500000	43.000000
max	943.000000	73.000000

rating pivot:

```
rating_pivot=data.pivot(index='user id',columns='movie id',values='rating').fillna(0)
```

```
rating_pivot[:10]
```

movie id	1	2	3	4	5	6	7	8	9	10	...	1673	1674	1675	1676	1677	1678	1679	1680	1681	1682
user id																					
1	5.0	3.0	4.0	3.0	3.0	5.0	4.0	1.0	5.0	3.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	4.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	4.0	0.0	0.0	0.0	0.0	0.0	2.0	4.0	4.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	5.0	0.0	0.0	5.0	5.0	5.0	4.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	5.0	4.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	4.0	0.0	0.0	4.0	0.0	0.0	4.0	0.0	4.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

- **recommender models exploration**

Direction 1: given a movie query with id 1547(randomly generated), recommend the movie title and the titles of the most similar 5 movies.

we are going to use item based knn to explore this direction. First, find the transpose of the rating pivot, and find k=6 nearest neighbors of the query item, and then return the neighbors, excluding itself. We are using k=6 here as the results also includes the movie itself and other 5 movies as required.

look for movie title in the item dataset and return in the query result below:

```
distances, indices = model_knn.kneighbors(rating_pivot_movie.iloc[query_index, :].reshape(1, -1), n_neighbors = 6)
# look for movie title in the item dataset and return in the query result below:
for i in range(0, len(distances.flatten())):
    if i == 0:
        print 'Recommendations for {0}:\n'.format(item.loc[item['movie id'] ==rating_pivot_movie.index[query_index],'mo
    else:
        print '{0}: {1}, with distance of {2}:'.format(i, item.loc[item['movie id'] ==rating_pivot_movie.index[indices.
```

```
Recommendations for The Courtyard (1995):

1: Quartier Mozart (1992), with distance of 0.0:
2: I, Worst of All (Yo, la peor de todas) (1990), with distance of 0.0:
3: Touki Bouki (Journey of the Hyena) (1973), with distance of 0.0:
4: The Courtyard (1995), with distance of 0.0:
5: Pharaoh's Army (1995), with distance of 0.0:
```

```
/anaconda/lib/python2.7/site-packages/ipykernel_launcher.py:1: FutureWarning: reshape is deprecated and will raise i
n a subsequent release. Please use .values.reshape(...) instead
  """Entry point for launching an IPython kernel.
```

```
distances # we are looking at 6 neighbors, and the first one is the distance to itself, that is why it is 0.
```

```
array([[ 0.        ,  0.5119975 ,  0.51421624,  0.52473047,  0.5777278 ,
         0.58980777]])
```

Direction 2: given a user query id 902(randomly generated), return 5 users with most similar behavior.

we are going to use user based knn to explore this direction with k=6, and when we get the 6 results, exclude the most similar one as it would be the user self id, the results would be 5 other user's id as below:

```
query_index2=np.random.choice(vote.shape[0])
```

```
print query_index2
```

```
902
```

```
distances2, indices2 = model_knn2.kneighbors(vote[query_index2].reshape(1, -1), n_neighbors = 6)
# look for movie title in the item dataset and return in the query result below:
for i in range(0, len(distances2.flatten())):
    if i == 0:
        print 'Recommendations for user id {0}:\n'.format(query_index2)
    else:
        print '{0}: user id {1}, with distance of {2}:'.format(i, indices2.flatten()[i], distances2.flatten()[i])
```

```
Recommendations for user id 902:

1: user id 193, with distance of 0.550999338662:
2: user id 176, with distance of 0.55512852004:
3: user id 473, with distance of 0.557581364879:
4: user id 915, with distance of 0.557891291468:
5: user id 665, with distance of 0.560886530711:
```

```
indices2
```

```
array([[902, 193, 176, 473, 915, 665]])
```

```
distances2
```

```
array([[ 0.        ,  0.55099934,  0.55512852,  0.55758136,  0.55789129,
         0.56088653]])
```

Direction 3: given a user id 298 and movie query id 474(randomly generated), return the rating for this movie from the user.

User based method would be: calculate nearest k users using pivot with all users who have rated this movie and without column 474(as we assume this info is unknown for user 298); as column movie 474 would be saved as target and average the nearest k neighbor's rating on this movie

```
distances3, indices3 = model_knn3.kneighbors(record.reshape(1, -1), n_neighbors =6)
```

```
rate=0
counter=0
unique_index=rating_i.index.values
user_ids=[]
n_neighbors=6
original_rating= data.ix[(data['user id'] ==j) & (data['movie id']==i)].rating.item()
# look for movie title in the item dataset and return in the query result below:
for k in range(0, len(distances3.flatten())):
    user_id=unique_index[indices3[0][k]]
    user_ids.append(user_id)
    if i == 0:
        print 'Recommendations for user id {0}:\n'.format(j)
    else:
        newdf=data.ix[(data['user id'] ==user_id) & (data['movie id']==i)]
        if newdf.empty:
            pass
        else:
            rating=newdf.rating.item()
            counter=counter+1
            rate=rate+rating
average_rate=rate/counter
print 'Predicted rating of movie {0} for user id {1} is: {2}'.format(i,j,average_rate)
print 'The original rating of movie {0} for user id {1} is: {2} '.format(i,j,original_rating)
print "The nearest {0} neighbors are(including itself):".format(n_neighbors)
print user_ids
```

```
Predicted rating of movie 474 for user id 298 is: 4
The original rating of movie 474 for user id 298 is: 4
The nearest 6 neighbors are(including itself):
[298, 716, 313, 194, 533, 271]
```

Direction 4: given a user id, recommend some movies the user I have watched before.

User We have tried to predict the rating by user and by movie in the last two models, now we could try to recommend new movies to user to watch using matrix factorization. Here is part of the algorithm:

```
from scipy.sparse.linalg import svds
U, sigma, Vt = svds(data_demeaned, k = 50)
sigma=np.diag(sigma)
```

```
all_user_predicted_ratings = np.dot(np.dot(U, sigma), Vt) + user_rating_mean.reshape(-1, 1)
preds_df = pd.DataFrame(all_user_predicted_ratings, columns = rating_pivot.columns)
```

The recommended 5 movies:

```
predictions
```

	movie id	movie title	release date	video release date	IMDb URL
568	742	Ransom (1996)	08-Nov-1996	NaN	http://us.imdb.com/M/title-exact?Ransom%20(1996)
97	156	Reservoir Dogs (1992)	01-Jan-1992	NaN	http://us.imdb.com/M/title-exact?Reservoir%20D...
412	558	Heavenly Creatures (1994)	01-Jan-1994	NaN	http://us.imdb.com/M/title-exact?Heavenly%20Cr...
295	421	William Shakespeare's Romeo and Juliet (1996)	25-Oct-1996	NaN	http://us.imdb.com/Title?Romeo+%2B+Juliet+(1996)
340	474	Dr. Strangelove or: How I Learned to Stop Worr...	01-Jan-1963	NaN	http://us.imdb.com/M/title-exact?Dr.%20Strange...

- **implementation on three models**

  Firstly, load the test data from the m-lens data source: The data set ua.test is a test set from splitting the u data into a training set and a test set, with exactly 10 ratings per user in the test set. We are going to use the rating pivot as training dataset and the test set for testing in the following algorithms

```
test=pd.read_csv('ml-100k/ua.test', sep='\t', names=dataCol, encoding='latin-1')
```

```
test.shape
```

```
(9430, 4)
```

```
test.head
```

```
<bound method DataFrame.head of       user id  movie id  rating  timestamp
0           1        20       4  887431883
1           1        33       4  878542699
2           1        61       4  878542420
3           1       117       3  874965739
4           1       155       2  878542201
5           1       160       4  875072547
6           1       171       5  889751711
7           1       189       3  888732928
8           1       202       5  875072442
9           1       265       4  878542441
```

1. item-based collaborative recommender

For each instance (user_id, movie_id, rating) in the test, find all the movies the user have rated, selecte the closest k movies with the movie_id in the instance; the k movies would include the moive_id itself as the closest one, we would need to

exclude it when calculating rating, then get a average rating for this moive_id from this k-1 movies as the rating. save the rounded calculated rating into a list named labels; save the original rating from test as classes, so we could calculate the accuracy later.
here is the main function:

```
from time import time
t0=time()
rawRating=[] # for raw averaged rating from k nearest neighbors
ratingFortest=[] # fro rounded average rating from k nearest neighbors
for id in array_test:
    #print id
    query_index=id[1] #movie
    user_id=id[0] # user
    neighbors=[]
    rating=0.0
    counter=0
    rating_i=rating_pivot_movie.loc[rating_pivot_movie[user_id] != 0]
    rating_matrix_movie=csr_matrix(rating_i.values)
    #print rating_i[:5]
    model_knn=NearestNeighbors(metric='cosine',algorithm='brute')
    model_knn.fit(rating_matrix_movie)
    unique_index=rating_i.index.values
    #print unique_index
    if query_index in unique_index:
        distances, indices = model_knn.kneighbors(rating_i.loc[[query_index]].values, n_neighbors = 6)
        #print indices
        for i in range(0, len(indices.flatten())):
            if i == 0:
                pass
            else:
                movie_id=unique_index[indices[0][i]]
                rate=data.loc[((data['user id'] == user_id) & (data['movie id'] == movie_id))].rating.item()
                rating=rating+rate
                counter=counter+1
        avg_rating=rating/counter
    else:
        avg_rating=rating_i[user_id].mean()
    rawRating.append(avg_rating)
    ratingFortest.append(round(avg_rating))
print('done in %0.3fs.'%(time()-t0))
```

And calculate completeness score and homogeneity score, mean absolute error.

```
ratingFortest[:10]
```

```
[5.0, 3.0, 5.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0]
```

```
from sklearn.metrics import completeness_score, homogeneity_score
```

```
print completeness_score(array_test[:,2],ratingFortest)
```

```
0.104681965465
```

```
print homogeneity_score(array_test[:,2],ratingFortest)
```

```
0.0790519777136
```

```
err=0.0
for i in range(9430):
    error=abs(classes[i]-ratingFortest[i])
    err=err+error
print 'The overall mean absolute error(MAE) of prediction is: '
print err/9430
```

```
The overall mean absolute error(MAE) of prediction is:
0.314492753623
```

2. user based collaborative recommender

For each instance(user_id, movie_id, rating) in the test, find the subset of ratings of all the ratings for this movie, find the k nearest neighbors of user_id, exclude itself, and calculate the average rating as the rating calculated.

main function:

```
from time import time
t0=time()
rawRating=[]
ratingFortest=[]
for id in array_test:
    #print id
    query_index=id[1] #movie
    user_id=id[0] # user
    neighbors=[]
    rating=0.0
    counter=0
    rating_i=rating_pivot.loc[rating_pivot[query_index] != 0]
    rating_matrix_movie=csr_matrix(rating_i.values)
    #print rating_i[:5]
    model_knn=NearestNeighbors(metric='cosine',algorithm='brute')
    model_knn.fit(rating_matrix_movie)
    unique_index=rating_i.index.values
    #print unique_index
    if rating_i.shape[0]<4:
        avg_rating=rating_i[query_index].mean()
    else:
        distances, indices = model_knn.kneighbors(rating_i.loc[[user_id]].values, n_neighbors = 4)
        #print indices
        for i in range(0, len(indices.flatten())):
            if i == 0:
                pass
            else:
                k_user_id=unique_index[indices[0][i]]
                rate=data.loc[((data['user id'] == k_user_id) & (data['movie id'] == query_index))].rating.item()
                rating=rating+rate
                counter=counter+1
        avg_rating=rating/counter
    rawRating.append(avg_rating)
    ratingFortest.append(round(avg_rating))
print('done in %0.3fs.'%(time()-t0))
```

And calculate completeness score and homogeneity score, mean absolute error, as in item-based recommender.

```
ratingFortest[:20]
labels=[int(i) for i in ratingFortest]
labels[:20]
```

```
[4, 3, 4, 3, 2, 4, 4, 4, 4, 4, 3, 4, 4, 3, 3, 3, 4, 4, 4, 1]
```

```
classes[:20]
```

```
[4, 4, 4, 3, 2, 4, 5, 3, 5, 4, 4, 5, 5, 3, 3, 3, 4, 4, 3, 1]
```

```
print completeness_score(classes,labels)
```

```
0.0547050495129
```

```
print homogeneity_score(classes,ratingFortest)
```

```
0.0450807342562
```

```
err=0.0
for i in range(9430):
    error=abs(classes[i]-ratingFortest[i])
    err=err+error
print 'The overall mean absolute error(MAE) of prediction is: '
print err/9430
```

```
The overall mean absolute error(MAE) of prediction is:
0.330851891128
The overall accracy of prediction is:
0.669148108872
```

3. matrix factorization based recommender

For each instance(user_id, movie_id, rating) in the test, find the rating of all the this user for this movie from the predictions of the matrix factorization results.

main function:

```
def matrix_factorization(R,P,Q,K,steps=5000,alpha=0.0002,beta=0.02):
    Q=Q.T
    for step in xrange(steps):
        for i in xrange(len(R)):
            for j in xrange(len(R[i])):
                if R[i][j]>0:
                    eij=R[i][j]-np.dot(P[i,:],Q[:,j])
                    for k in xrange(K):
                        P[i][k]=P[i][k]+alpha*(2*eij*Q[k][j]-beta*P[i][k])
                        Q[k][j]=Q[k][j]+alpha*(2*eij*P[i][k]-beta*Q[k][j])
        eR=np.dot(P,Q)
        e=0
        for i in xrange(len(R)):
            for j in xrange(len(R[i])):
                if R[i][j]>0:
                    e=e+pow(R[i][j]-np.dot(P[i,:],Q[:,j]),2)
                    for k in xrange(K):
                        e=e+(beta/2)*(pow(P[i][k],2)+pow(Q[k][j],2))
        if e<0.001:
            break
        print "step %d of %d; error: %0.5f; time:%0.2f"%(step+1,steps,e,time())
    return P,Q.T
```

result:

```
from time import time
t0=time()
fP,fQ=matrix_factorization(dataMat,P,Q,K,steps=steps)
print('done in %0.3fs.'%(time()-t0))
```

```
step 2983 of 3000; error: 71301.89337; time:1510548784.94
step 2984 of 3000; error: 71301.47912; time:1510548789.18
step 2985 of 3000; error: 71301.06519; time:1510548793.56
step 2986 of 3000; error: 71300.65159; time:1510548797.82
step 2987 of 3000; error: 71300.23831; time:1510548802.03
step 2988 of 3000; error: 71299.82536; time:1510548806.28
step 2989 of 3000; error: 71299.41273; time:1510548810.53
step 2990 of 3000; error: 71299.00042; time:1510548814.81
step 2991 of 3000; error: 71298.58844; time:1510548819.06
step 2992 of 3000; error: 71298.17678; time:1510548823.33
step 2993 of 3000; error: 71297.76544; time:1510548827.55
step 2994 of 3000; error: 71297.35442; time:1510548831.79
step 2995 of 3000; error: 71296.94372; time:1510548836.03
step 2996 of 3000; error: 71296.53335; time:1510548840.30
step 2997 of 3000; error: 71296.12329; time:1510548844.54
step 2998 of 3000; error: 71295.71356; time:1510548848.75
step 2999 of 3000; error: 71295.30414; time:1510548852.99
step 3000 of 3000; error: 71294.89504; time:1510548857.28
done in 13122.586s.
```

```
Preds=np.dot(fP,fQ.T)
Preds.shape
```

```
(943, 1682)
```

And calculate completeness score and homogeneity score, mean absolute error:

```
labels[:10]
```

```
[4.0, 3.0, 4.0, 3.0, 2.0, 4.0, 5.0, 4.0, 4.0, 4.0]
```

```
print completeness_score(classes,labels)
```

```
0.243055407939
```

```
print homogeneity_score(classes,labels)
```

```
0.203494139909
```

```
err=0.0
for i in range(9430):
    error=abs(classes[i]-labels[i])
    err=err+error
print 'The overall mean absolute error(MAE) of prediction is: '
print err/9430
```

```
The overall mean absolute error(MAE) of prediction is:
0.217511488158
```

Part 3. Performance evaluation

Result:

Accuracy	Item_based recommender	User_based recommender	Matrix_factorization_based recommender
Completeness_Score	0.104	0.054	0.243
Homogeneity_Score	0.079	0.045	0.203
MAE	0.314	0.331	0.218
Efficiency	**Item_based recommender**	**User_based recommender**	**Matrix_factorization_based recommender**
Run Time	102.042s	159.563s	13122.586s

From the comparison of the result, we can see the Matrix_factorization based recommender is more accurate than the other two recommenders, but it also takes a much longer time to generate the prediction result. User_based recommender takes a little bit longer than the item_based recommender, and the Mean absolute error is also a little higher than item_based recommender.

Part 4. Future direction

Due to the capacity of my computer, I am using a 100,000 rating dataset, which include some one rating only movie, and make it hard to predict its rating. For future research reference, if possible, using a larger dataset to test on these three algorithms would generate better results.

Readme file:

The data used in this application is movie-lens 100k rating data, with mainly four features used, movie_id, user_id, rating and movie_title from 3 dataset u.data, u.item, u.user; and a set aside testing dataset ua.test, including mainly three features used for each instance, movie_id, user_id and rating.

From the models explored, this application could answer below queries:

1. Give a user_id, return k nearest neighbors with the most similar rating behavior;
2. Given a movie_id, return k nearest neighbors with the most similar rating;
3. Given a user_id, recommend k movies the user have not watched before;
4. Given a user_id, movie_id, return the predicted rating from three algorithms: user_based recommender, item_based recommender and matrix_factorization_based recommender.
5. Gvien a list of user_id, movie_id, return the predicted rating and MAE for the three algorithms: user_based recommender, item_based recommender and matrix_factorization_based recommender.