item-based-recommender-system

December 15, 2017

1 How to run this program

First, please make sure that you have **python3** installed (preferably **Anaconda** package).
Then use **jupyter notebook** to run the **.ipynb** file.
If you have any missing python modules, please install them using **pip install**.

```
In [1]: import numpy as np
        import pandas as pd
        from sklearn.model_selection import train_test_split
        import matplotlib.pyplot as plt
```

2 Loading data

Load the MovieLens data (educational version)

```
In [2]: %%time
        data_path = "ml-latest-small/ratings.csv"
        data = pd.read_csv(data_path, sep=',', header=0)
        data = data[['userId', 'movieId', 'rating']]
```

```
Wall time: 56.2 ms
```

```
In [3]: user_ids = sorted(set(data['userId']))
        movie_ids = sorted(set(data['movieId']))
        n_users = len(user_ids)
        n_movies = len(movie_ids)

        print("Number of users: {}\nNumber of movies: {}".format(n_users, n_movies)
```

```
Number of users: 671
Number of movies: 9066
```

```
In [4]: # show how many users have rated a certain movie
        vector_sizes = data.groupby('movieId')['userId'].nunique().sort_values(asce
        print(vector_sizes)
        print('On average, each movie is rated {} times'.format(vector_sizes.mean()
```

```
movieId
356       341
296       324
318       311
593       304
260       291
480       274
2571      259
1         247
527       244
589       237
1196      234
110       228
1270      226
608       224
1198      220
2858      220
780       218
1210      217
588       215
457       213
2959      202
590       202
47        201
50        201
4993      200
858       200
364       200
150       200
380       198
32        196
         ...
26694       1
26695       1
26701       1
3003        1
26492       1
26346       1
3021        1
26349       1
26350       1
26371       1
26393       1
3031        1
26394       1
26400       1
3025        1
26404       1
```

```
26409       1
26413       1
26487       1
26414       1
26422       1
26430       1
26435       1
26462       1
26464       1
26467       1
26471       1
26480       1
26485       1
163949      1
Name: userId, dtype: int64
On average, each movie is rated 11.030664019413193 times
```

In [5]:
```
# show examples of the data
data
```

Out[5]:

	userId	movieId	rating
0	1	31	2.5
1	1	1029	3.0
2	1	1061	3.0
3	1	1129	2.0
4	1	1172	4.0
5	1	1263	2.0
6	1	1287	2.0
7	1	1293	2.0
8	1	1339	3.5
9	1	1343	2.0
10	1	1371	2.5
11	1	1405	1.0
12	1	1953	4.0
13	1	2105	4.0
14	1	2150	3.0
15	1	2193	2.0
16	1	2294	2.0
17	1	2455	2.5
18	1	2968	1.0
19	1	3671	3.0
20	2	10	4.0
21	2	17	5.0
22	2	39	5.0
23	2	47	4.0
24	2	50	4.0
25	2	52	3.0

```
26          2      62     3.0
27          2     110     4.0
28          2     144     3.0
29          2     150     5.0
...       ...     ...     ...
99974     671    4034     4.5
99975     671    4306     5.0
99976     671    4308     3.5
99977     671    4880     4.0
99978     671    4886     5.0
99979     671    4896     5.0
99980     671    4963     4.5
99981     671    4973     4.5
99982     671    4993     5.0
99983     671    4995     4.0
99984     671    5010     2.0
99985     671    5218     2.0
99986     671    5299     3.0
99987     671    5349     4.0
99988     671    5377     4.0
99989     671    5445     4.5
99990     671    5464     3.0
99991     671    5669     4.0
99992     671    5816     4.0
99993     671    5902     3.5
99994     671    5952     5.0
99995     671    5989     4.0
99996     671    5991     4.5
99997     671    5995     4.0
99998     671    6212     2.5
99999     671    6268     2.5
100000    671    6269     4.0
100001    671    6365     4.0
100002    671    6385     2.5
100003    671    6565     3.5

[100004 rows x 3 columns]
```

3 Mean centering

Subtract mean of rating for each user

In [6]:
```
%%time
# find the mean of each user
user_group = data.groupby(by='userId')
user_means = user_group['rating'].agg(['mean', 'count'])
```

```
Wall time: 9.02 ms
```

In [7]:
```
# create a new column named "meanCenteredRating"

# this function takes in ratings of one user and return mean_centered ratin
mean_centering = lambda ratings: ratings - ratings.mean()
data['meanCenteredRating'] = user_group['rating'].transform(mean_centering)
data
```

Out[7]:

	userId	movieId	rating	meanCenteredRating
0	1	31	2.5	-0.050000
1	1	1029	3.0	0.450000
2	1	1061	3.0	0.450000
3	1	1129	2.0	-0.550000
4	1	1172	4.0	1.450000
5	1	1263	2.0	-0.550000
6	1	1287	2.0	-0.550000
7	1	1293	2.0	-0.550000
8	1	1339	3.5	0.950000
9	1	1343	2.0	-0.550000
10	1	1371	2.5	-0.050000
11	1	1405	1.0	-1.550000
12	1	1953	4.0	1.450000
13	1	2105	4.0	1.450000
14	1	2150	3.0	0.450000
15	1	2193	2.0	-0.550000
16	1	2294	2.0	-0.550000
17	1	2455	2.5	-0.050000
18	1	2968	1.0	-1.550000
19	1	3671	3.0	0.450000
20	2	10	4.0	0.513158
21	2	17	5.0	1.513158
22	2	39	5.0	1.513158
23	2	47	4.0	0.513158
24	2	50	4.0	0.513158
25	2	52	3.0	-0.486842
26	2	62	3.0	-0.486842
27	2	110	4.0	0.513158
28	2	144	3.0	-0.486842
29	2	150	5.0	1.513158
...	...	...	...	...
99974	671	4034	4.5	0.582609
99975	671	4306	5.0	1.082609
99976	671	4308	3.5	-0.417391
99977	671	4880	4.0	0.082609
99978	671	4886	5.0	1.082609
99979	671	4896	5.0	1.082609
99980	671	4963	4.5	0.582609

```
99981       671    4973   4.5           0.582609
99982       671    4993   5.0           1.082609
99983       671    4995   4.0           0.082609
99984       671    5010   2.0          -1.917391
99985       671    5218   2.0          -1.917391
99986       671    5299   3.0          -0.917391
99987       671    5349   4.0           0.082609
99988       671    5377   4.0           0.082609
99989       671    5445   4.5           0.582609
99990       671    5464   3.0          -0.917391
99991       671    5669   4.0           0.082609
99992       671    5816   4.0           0.082609
99993       671    5902   3.5          -0.417391
99994       671    5952   5.0           1.082609
99995       671    5989   4.0           0.082609
99996       671    5991   4.5           0.582609
99997       671    5995   4.0           0.082609
99998       671    6212   2.5          -1.417391
99999       671    6268   2.5          -1.417391
100000      671    6269   4.0           0.082609
100001      671    6365   4.0           0.082609
100002      671    6385   2.5          -1.417391
100003      671    6565   3.5          -0.417391

[100004 rows x 4 columns]
```

In [8]:
```
# show user means
user_means
```

Out[8]:
```
            mean  count
userId
1       2.550000     20
2       3.486842     76
3       3.568627     51
4       4.348039    204
5       3.910000    100
6       3.261364     44
7       3.465909     88
8       3.866379    116
9       3.755556     45
10      3.695652     46
11      4.078947     38
12      2.754098     61
13      3.745283     53
14      2.950000     20
15      2.621765   1700
16      4.120690     29
17      3.743802    363
```

18	3.235294	51
19	3.534279	423
20	3.290816	98
21	3.506173	162
22	3.275000	220
23	3.632920	726
24	3.666667	21
25	3.115385	26
26	3.468023	172
27	3.826087	23
28	4.280000	50
29	2.863636	22
30	3.765084	1011
...	...	...
642	3.916667	36
643	3.395833	24
644	3.743590	39
645	3.683333	30
646	4.130178	169
647	4.273333	150
648	3.628906	256
649	3.511111	90
650	3.310345	29
651	3.900000	20
652	4.220974	267
653	4.000000	51
654	4.068690	626
655	4.085714	105
656	4.523438	128
657	3.500000	20
658	4.350000	60
659	3.387324	142
660	4.168478	92
661	3.833333	33
662	3.396552	58
663	3.730769	26
664	3.796724	519
665	3.285714	434
666	2.950000	40
667	3.647059	68
668	3.750000	20
669	3.351351	37
670	3.806452	31
671	3.917391	115

[671 rows x 2 columns]

4 Splitting data

Split the data into training set and test set. Prepare the training set as a user-item ratings matrix.

```
In [10]: # randomly split the data set
         test_size = 0.05
         data_train, data_test = train_test_split(data, test_size=test_size, randon

         # show shape of the data
         data_train.shape, data_test.shape
```

```
Out[10]: ((95003, 4), (5001, 4))
```

```
In [11]: %%time
         # build userId to row mapping dictionary
         user2row = dict()
         row2user = dict()
         for i, user_id in enumerate(user_ids):
             user2row[user_id] = i
             row2user[i] = user_id

         # build movieId to column mapping dictionary
         movie2col = dict()
         col2movie = dict()
         for i, movie_id in enumerate(movie_ids):
             movie2col[movie_id] = i
             col2movie[i] = movie_id
```

```
Wall time: 4.01 ms
```

```
In [12]: %%time
         # turn ratings data in table format into a user-item rating matrix
         # the field will be filled with NaN if user didn't provide a rating
         def data_to_matrix(data):
             mat = np.full((n_users, n_movies), np.nan, dtype=np.float32)
             for idx, row in data.iterrows():
                 mat[user2row[row['userId']], movie2col[row['movieId']]] = row['mea
             return mat

         # prepare the data as a user-item rating matrix for the next step
         train_ratings = data_to_matrix(data_train)
```

```
Wall time: 5.73 s
```

5 Compute similarity matrix

Build the item-item similarity matrix. This section takes most of the processing time.

In [13]:
```
# create a blank similarity matrix containing zeros
%time sim_matrix = np.empty((n_movies, n_movies), dtype=np.float32)
sim_matrix.shape
```

Wall time: 0 ns

Out[13]: (9066, 9066)

In [14]:
```
# remove co-elements from 2 vectors if at least one of them is NaN
def remove_nans(a, b):
    # assuming that a and b are 1-d vectors, create a new axis for both of
    a = a[..., np.newaxis]
    b = b[..., np.newaxis]
    concat = np.concatenate([a, b], axis=1)
    nonan = concat[~np.isnan(concat).any(axis=1)]
    return nonan[:, 0], nonan[:, 1]

# show examples of how to use remove_nans()
a = np.array([-1,2      ,np.nan,4])
b = np.array([-2,np.nan,3      ,5])
remove_nans(a, b)
```

Out[14]: (array([-1., 4.]), array([-2., 5.]))

In [15]:
```
# calculate a similarity value given 2 vectors
# the output is a value between -1 and 1
# min_co_elements is the number that determine whether to output NaN
# or output the similarity value, if co-elements are too low, the similari
# will not be a good estimate, e.g. if there is only 1 co-element then the
# will only be either -1 or 1, that's sometimes not desirable, so a thresh
def calsim(item1, item2, min_co_elements=1):
    item1, item2 = remove_nans(item1, item2)
    if item1.size == 0 or item1.size < min_co_elements: # item1 and item2
        return np.nan
#     print(item1.size)
    dot = item1.dot(item2)
    # find magnitude A.K.A. length of the vector by taking sqrt of the sum
    norm1 = np.linalg.norm(item1)
    norm2 = np.linalg.norm(item2)
    return dot / (norm1 * norm2)

# show example of how to use calsim()
calsim(a, b)
```

Out[15]: 0.99083016804429913

In [17]:
```
# either load or run the next cell to compute similarity matrix
sim_matrix = np.load('sim_matrix.npy')
```

```
In [ ]: %%time
        # calculate all the similarities
        for item1 in range(n_movies):
            item1vector = train_ratings[:, item1]
            for item2 in range(item1, n_movies):
                item2vector = train_ratings[:, item2]
                sim = calsim(item1vector, item2vector, min_co_elements=2)
                sim_matrix[item1, item2] = sim
                sim_matrix[item2, item1] = sim
            if (item1+1) % 50 == 0 or item1+1 == n_movies:
                print("Progress: {}/{} ({:.2f} %) items calculated".format(item1+1,
```

```
In [ ]: %%time
        # this sim matrix takes a lot of time to compute,
        # so saving it to the disk will help saving time in the future
        np.save('sim_matrix', sim_matrix)
```

```
In [18]: print('Fractions of similarity matrix that are NaN:', np.isnan(sim_matrix)
```

```
Fractions of similarity matrix that are NaN: 0.936411215661
```

6 Recommendation

Test recommendation using the item-item similarity matrix built previously.

1. We first need to define a predict() function then use it repeatedly to predict rating of every movie of a given user.
2. We then sort the predictions and show movies with top predictions

```
In [19]: # define a predict function which receives row and column in the ratings r
         # then output a rating value (without mean addition), or np.nan if there a
         # user_item is a tuple (user_row, movie_column)
         # sim_threshold is the similarity threshold of each item,
         # if the item exceeds this value, it will be chosen for averaging the outc
         def predict(ratings, user_item, sim_threshold, debug=True):
             desired_user, desired_item = user_item
             rating_sum = 0.
             total_sim = 0.
             for item in range(ratings.shape[1]):
                 s = sim_matrix[item, desired_item]
                 rating = ratings[desired_user, item]
                 if np.isnan(s) or s < sim_threshold or item == desired_item or np.
                     continue
                 rating_sum += s * rating
                 total_sim += s
                 if debug:
                     print('sim and rating of item {}:'.format(item), s, rating)
             return rating_sum / total_sim if total_sim else np.nan
```

In [20]:
```
# this is the similarity threshold value, as the only hyperparameter avai
sim_threshold = 0.
```

In [21]:
```
predict(train_ratings, (0, 30), sim_threshold), train_ratings[0, 30]
```

```
sim and rating of item 906: 0.117642 -0.55
sim and rating of item 1017: 0.0505805 -0.55
sim and rating of item 1111: 0.815062 -0.05
sim and rating of item 1140: 0.0289128 -1.55
sim and rating of item 1665: 0.064384 1.45
sim and rating of item 1708: 0.550293 0.45
sim and rating of item 1815: 0.73969 -0.55
sim and rating of item 2925: 0.0604668 0.45
```

Out[21]:
```
(-0.08929476924149483, -0.050000001)
```

In [22]:
```
# load the movie names
movie_file = "ml-latest-small/movies.csv"
movie_df = pd.read_csv(movie_file, header=0)
movie_df.head()
```

Out[22]:
```
   movieId                               title  \
0        1                    Toy Story (1995)
1        2                      Jumanji (1995)
2        3             Grumpier Old Men (1995)
3        4            Waiting to Exhale (1995)
4        5  Father of the Bride Part II (1995)

                                        genres
0  Adventure|Animation|Children|Comedy|Fantasy
1                   Adventure|Children|Fantasy
2                               Comedy|Romance
3                         Comedy|Drama|Romance
4                                       Comedy
```

In [23]:
```
# desired_user is the user row that we want to recommend
# return recommended item indices sorted by rating descendingly, and the a
def recommend(ratings, desired_user, sim_threshold):
    scores = []
    for item in range(ratings.shape[1]):
        score = ratings[desired_user, item]
        if np.isnan(score):
            score = predict(ratings, (desired_user, item), sim_threshold,
        else:
            score = -np.infty # we don't want to recommend movies that use
        scores.append(score)
    scores = np.array(scores)
    scores_argsort = np.argsort(scores)[::-1]
```

```
        scores_sort = np.sort(scores)[::-1]

        # numpy will put nan into the back of the array after sort
        # when we reverse the array, nan will be at the front
        # we want to move nan into the back again
        # so we use a numpy trick which rolls the array value
        # source: https://stackoverflow.com/a/35038821/2593810
        no_of_nan = np.count_nonzero(np.isnan(scores))
        scores_argsort = np.roll(scores_argsort, -no_of_nan)
        scores_sort = np.roll(scores_sort, -no_of_nan)
        return scores_argsort, scores_sort

    def recommend_msg(user_row, scores_argsort, scores_sort, how_many=10):
        m = user_means.loc[row2user[user_row]]['mean']
        print('User mean rating:', m)
        msg = pd.DataFrame(columns=['movieId', 'title', 'genres', 'rating'])
        for i in range(how_many):
            col = scores_argsort[i]
            movie_id = col2movie[col]
            movie = movie_df.loc[movie_df['movieId'] == movie_id].iloc[0]
            msg.loc[i+1] = [movie_id, movie['title'], movie['genres'], scores_
        msg['movieId'] = msg['movieId'].astype(np.int32)
        return msg
```

In [24]:
```
%%time
user = 0 # the given user
scores_argsort, scores_sort = recommend(train_ratings, user, sim_threshol
```

```
Wall time: 1min 40s
```

In [25]:
```
scores_argsort, scores_sort
```

Out[25]:
```
(array([4880, 1387, 5743, ..., 6415, 7230, 6420], dtype=int64),
 array([ 1.45000012,  1.45000012,  1.45000011, ...,         nan,
                nan,         nan]))
```

In [26]:
```
recommend_msg(user, scores_argsort, scores_sort, how_many=10)
```

```
User mean rating: 2.55
```

Out[26]:
```
   movieId                                     title  \
1     6951                 Cat in the Hat, The (2003)
2     1769            Replacement Killers, The (1998)
3    26231                         Performance (1970)
4    36276       Hidden (a.k.a. Cache) (Caché) (2005)
5     4443                             Outland (1981)
6   139757                      Best of Enemies (2015)
```

```
7       4570                          Big Picture, The (1989)
8       3928   Abbott and Costello Meet Frankenstein (1948)
9       1943              Greatest Show on Earth, The (1952)
10      3181                                    Titus (1999)

                      genres  rating
1            Children|Comedy     4.0
2      Action|Crime|Thriller     4.0
3       Crime|Drama|Thriller     4.0
4     Drama|Mystery|Thriller     4.0
5     Action|Sci-Fi|Thriller     4.0
6                Documentary     4.0
7               Comedy|Drama     4.0
8              Comedy|Horror     4.0
9                      Drama     4.0
10                     Drama     4.0
```

7 Evaluation

Evaluate the error on the test set. The error metric chosen in our work is **MAE**. 1. We need to predict mean centered ratings of every (user,movie) pair in the test data 2. Take the difference between the true ratings and the predicted ratings 3. Take the absolute 4. Take the mean

And that's how the error is computed.

In [27]:
```
# first, let's take a look at some of the test data
data_test.head()
```

Out[27]:

	userId	movieId	rating	meanCenteredRating
19090	128	1028	5.0	1.139319
99678	665	4736	1.0	-2.285714
18455	120	4002	3.0	-0.485507
35755	257	1274	4.0	0.393204
66536	468	6440	4.0	1.034082

In [28]:
```
# predict ratings for the given data table
def predict_table(data_test, sim_threshold, show_progress=True):
    n_test = data_test.shape[0]
    predictions = np.empty((n_test,))
    i = 0
    for idx, row in data_test.iterrows():
        pred = predict(train_ratings, (user2row[row['userId']], movie2col[
        predictions[i] = pred
        if show_progress and ((i+1) % 100 == 0 or i+1 == n_test):
            print("Progress: {}/{} ({:.2f} %) ratings predicted".format(i+
        i += 1
    if show_progress:
        print("Progress: {}/{} ({:.2f} %) ratings predicted".format(i+1, n
    return predictions
```

```
def eval_error(data_test, predictions):
    return np.abs(data_test['meanCenteredRating'] - predictions).mean()
```

In [29]:
```
%%time
# predicting ratings for every (user,movie) pair in the test data
predictions = predict_table(data_test, sim_threshold)
```

```
Progress: 100/5001 (2.00 %) ratings predicted
Progress: 200/5001 (4.00 %) ratings predicted
Progress: 300/5001 (6.00 %) ratings predicted
Progress: 400/5001 (8.00 %) ratings predicted
Progress: 500/5001 (10.00 %) ratings predicted
Progress: 600/5001 (12.00 %) ratings predicted
Progress: 700/5001 (14.00 %) ratings predicted
Progress: 800/5001 (16.00 %) ratings predicted
Progress: 900/5001 (18.00 %) ratings predicted
Progress: 1000/5001 (20.00 %) ratings predicted
Progress: 1100/5001 (22.00 %) ratings predicted
Progress: 1200/5001 (24.00 %) ratings predicted
Progress: 1300/5001 (25.99 %) ratings predicted
Progress: 1400/5001 (27.99 %) ratings predicted
Progress: 1500/5001 (29.99 %) ratings predicted
Progress: 1600/5001 (31.99 %) ratings predicted
Progress: 1700/5001 (33.99 %) ratings predicted
Progress: 1800/5001 (35.99 %) ratings predicted
Progress: 1900/5001 (37.99 %) ratings predicted
Progress: 2000/5001 (39.99 %) ratings predicted
Progress: 2100/5001 (41.99 %) ratings predicted
Progress: 2200/5001 (43.99 %) ratings predicted
Progress: 2300/5001 (45.99 %) ratings predicted
Progress: 2400/5001 (47.99 %) ratings predicted
Progress: 2500/5001 (49.99 %) ratings predicted
Progress: 2600/5001 (51.99 %) ratings predicted
Progress: 2700/5001 (53.99 %) ratings predicted
Progress: 2800/5001 (55.99 %) ratings predicted
Progress: 2900/5001 (57.99 %) ratings predicted
Progress: 3000/5001 (59.99 %) ratings predicted
Progress: 3100/5001 (61.99 %) ratings predicted
Progress: 3200/5001 (63.99 %) ratings predicted
Progress: 3300/5001 (65.99 %) ratings predicted
Progress: 3400/5001 (67.99 %) ratings predicted
Progress: 3500/5001 (69.99 %) ratings predicted
Progress: 3600/5001 (71.99 %) ratings predicted
Progress: 3700/5001 (73.99 %) ratings predicted
Progress: 3800/5001 (75.98 %) ratings predicted
Progress: 3900/5001 (77.98 %) ratings predicted
Progress: 4000/5001 (79.98 %) ratings predicted
```

```
Progress: 4100/5001 (81.98 %) ratings predicted
Progress: 4200/5001 (83.98 %) ratings predicted
Progress: 4300/5001 (85.98 %) ratings predicted
Progress: 4400/5001 (87.98 %) ratings predicted
Progress: 4500/5001 (89.98 %) ratings predicted
Progress: 4600/5001 (91.98 %) ratings predicted
Progress: 4700/5001 (93.98 %) ratings predicted
Progress: 4800/5001 (95.98 %) ratings predicted
Progress: 4900/5001 (97.98 %) ratings predicted
Progress: 5000/5001 (99.98 %) ratings predicted
Progress: 5001/5001 (100.00 %) ratings predicted
Progress: 5002/5001 (100.02 %) ratings predicted
Wall time: 1min 16s
```

In [30]:
```
data_test['prediction'] = predictions
data_test.head()
```

```
C:\Program Files\Anaconda3\lib\site-packages\ipykernel\__main__.py:1: SettingWithCo
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/i
  if __name__ == '__main__':
```

Out[30]:

	userId	movieId	rating	meanCenteredRating	prediction
19090	128	1028	5.0	1.139319	-0.000777
99678	665	4736	1.0	-2.285714	NaN
18455	120	4002	3.0	-0.485507	0.041654
35755	257	1274	4.0	0.393204	0.155186
66536	468	6440	4.0	1.034082	0.181659

In [31]:
```
data_test['abs_error'] = np.abs(data_test['meanCenteredRating'] - data_tes
data_test.head()
```

```
C:\Program Files\Anaconda3\lib\site-packages\ipykernel\__main__.py:1: SettingWithCo
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/i
  if __name__ == '__main__':
```

Out[31]:

	userId	movieId	rating	meanCenteredRating	prediction	abs_error
19090	128	1028	5.0	1.139319	-0.000777	1.140095
99678	665	4736	1.0	-2.285714	NaN	NaN
18455	120	4002	3.0	-0.485507	0.041654	0.527162
35755	257	1274	4.0	0.393204	0.155186	0.238017
66536	468	6440	4.0	1.034082	0.181659	0.852424

```
In [32]: # mean absolute error
         mae = data_test['abs_error'].mean()
         mae
```

```
Out[32]: 0.6830313136803959
```

8 Error Optimization

Find the best set of hyperparameters that yields the lowest error on the test set. In this work, we use **sim_threshold (similarity threshold)** as the only hyperparameter of the system.

We can find the best **sim_threshold** by iteratively 1. varying its value 2. predict outcome on the test set 3. evaluate the error 4. if the error is less than the least error found so far, save current **sim_threshold** as the best candidate

Repeat this cycle until enough satisfaction is achieved.

```
In [34]: # define a set of avaialble similarity thresholds
         candidate_sim_thresholds = np.linspace(-1, 0.75, num=16)
         candidate_sim_thresholds
```

```
Out[34]: array([-1.        , -0.88333333, -0.76666667, -0.65      , -0.53333333,
                -0.41666667, -0.3       , -0.18333333, -0.06666667,  0.05      ,
                 0.16666667,  0.28333333,  0.4       ,  0.51666667,  0.63333333,
                 0.75      ])
```

```
In [35]: %%time
         errors = np.empty_like(candidate_sim_thresholds, dtype=np.float32)
         for i, sim_threshold in enumerate(candidate_sim_thresholds):
             print('Current similarity threshold:', sim_threshold)
             predictions = predict_table(data_test, sim_threshold, show_progress=Fa
             error = eval_error(data_test, predictions)
             print('Error:', error)
             errors[i] = error
```

```
Current similarity threshold: -1.0
Error: 7.988810899230984
Current similarity threshold: -0.883333333333
Error: 4.0885070346383445
Current similarity threshold: -0.766666666667
Error: 3.91262495818463
Current similarity threshold: -0.65
Error: 3.022584248995466
Current similarity threshold: -0.533333333333
Error: 1.0224778915303434
Current similarity threshold: -0.416666666667
Error: 0.8640781102381164
Current similarity threshold: -0.3
Error: 0.691688704575591
Current similarity threshold: -0.183333333333
```

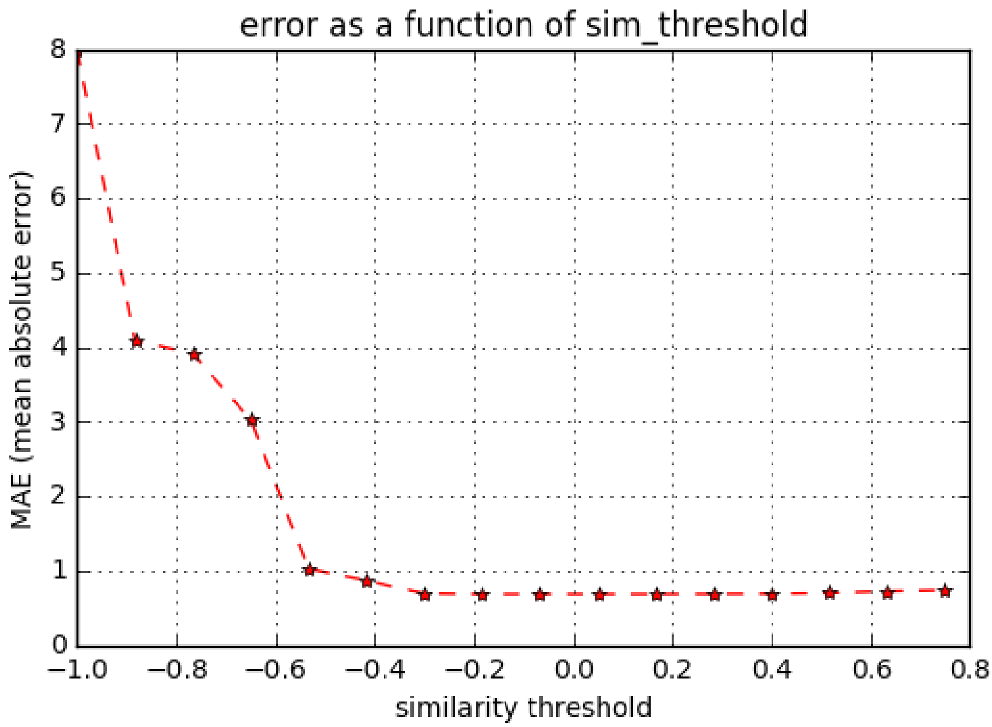
```
Error: 0.6827043885684035
Current similarity threshold: -0.0666666666667
Error: 0.6835753948311755
Current similarity threshold: 0.05
Error: 0.68290199826614
Current similarity threshold: 0.166666666667
Error: 0.6832810060491642
Current similarity threshold: 0.283333333333
Error: 0.6847076670101073
Current similarity threshold: 0.4
Error: 0.687890669476367
Current similarity threshold: 0.516666666667
Error: 0.6984668837100138
Current similarity threshold: 0.633333333333
Error: 0.7205414523340402
Current similarity threshold: 0.75
Error: 0.7359640433077927
Wall time: 19min 18s
```

```
In [36]: best_error_idx = np.argmin(errors)
         best_error = errors[best_error_idx]
         best_sim_threshold = candidate_sim_thresholds[best_error_idx]
         errors

Out[36]: array([ 7.98881102,  4.08850718,  3.91262507,  3.0225842 ,  1.02247787,
                 0.8640781 ,  0.69168872,  0.68270439,  0.68357539,  0.68290198,
                 0.683281  ,  0.68470764,  0.68789065,  0.6984669 ,  0.72054148,
                 0.73596406], dtype=float32)
```

```
In [37]: print('Optimal similarity threshold:', best_sim_threshold)
         print('Optimal error:', best_error)

Optimal similarity threshold: -0.183333333333
Optimal error: 0.682704
```

Plot the error as a function of **sim_threshold**.

```
In [38]: plt.plot(candidate_sim_thresholds, errors, 'r*--')
         plt.xlabel('similarity threshold')
         plt.ylabel('MAE (mean absolute error)')
         plt.grid()
         plt.title('error as a function of sim_threshold')
         plt.show()
```



9 Inference on the Real World

This is the last step, all we have done to this point is now on production.

Our task: Given a user, recommend some movies.

In [39]:
```
# choose a user_id from the data
user_id = 500
```

In [51]:
```
# we are tring to show what movies the user have rated in the past
# we are going to sort the records by rating,
# so we can compare the result to the recommendation provided by the syste
def get_ratings_of_user(user_id):
    user_records = data_train.loc[data_train['userId'] == user_id].sort_va
    user_records.drop(['userId', 'meanCenteredRating'], axis=1, inplace=Tr
    get_movie = lambda movie_id: movie_df.loc[movie_df['movieId'] == movie
    user_records['title'] = user_records['movieId'].apply(lambda movie_id:
    user_records['genres'] = user_records['movieId'].apply(lambda movie_ic
    return user_records

get_ratings_of_user(user_id)
```

Out[51]:
```
       movieId  rating                                              title
71353    38061     5.0                             Kiss Kiss Bang Bang (2005)
```

71234	2324	5.0	Life Is Beautiful (La Vita è bella) (1997)
71168	356	5.0	Forrest Gump (1994)
71330	7669	5.0	Pride and Prejudice (1995)
71346	31433	4.5	Wedding Date, The (2005)
71390	64034	4.5	Boy in the Striped Pajamas, The (Boy in the St...
71366	52287	4.5	Meet the Robinsons (2007)
71227	2139	4.5	Secret of NIMH, The (1982)
71303	4973	4.5	Amelie (Fabuleux destin d'Amélie Poulain, Le) ...
71381	58347	4.5	Penelope (2006)
71322	6942	4.5	Love Actually (2003)
71230	2145	4.5	Pretty in Pink (1986)
71367	52435	4.5	How the Grinch Stole Christmas! (1966)
71387	62718	4.5	Angus, Thongs and Perfect Snogging (2008)
71277	4014	4.5	Chocolat (2000)
71386	62155	4.5	Nick and Norah's Infinite Playlist (2008)
71379	56367	4.5	Juno (2007)
71221	1968	4.5	Breakfast Club, The (1985)
71239	2541	4.5	Cruel Intentions (1999)
71372	54259	4.5	Stardust (2007)
71241	2572	4.5	10 Things I Hate About You (1999)
71361	47518	4.5	Accepted (2006)
71164	318	4.5	Shawshank Redemption, The (1994)
71199	1207	4.0	To Kill a Mockingbird (1962)
71231	2150	4.0	Gods Must Be Crazy, The (1980)
71370	53322	4.0	Ocean's Thirteen (2007)
71198	1197	4.0	Princess Bride, The (1987)
71190	1013	4.0	Parent Trap, The (1961)
71184	700	4.0	Angus (1995)
71256	3174	4.0	Man on the Moon (1999)
...	...	...	...
71296	4750	1.5	3 Ninjas Knuckle Up (1995)
71250	2762	1.5	Sixth Sense, The (1999)
71262	3418	1.5	Thelma & Louise (1991)
71153	2	1.5	Jumanji (1995)
71217	1777	1.5	Wedding Singer, The (1998)
71179	593	1.5	Silence of the Lambs, The (1991)
71224	2054	1.0	Honey, I Shrunk the Kids (1989)
71200	1210	1.0	Star Wars: Episode VI - Return of the Jedi (1983)
71240	2571	1.0	Matrix, The (1999)
71362	48385	1.0	Borat: Cultural Learnings of America for Make ...
71340	8949	1.0	Sideways (2004)
71313	5952	1.0	Lord of the Rings: The Two Towers, The (2002)
71345	30810	1.0	Life Aquatic with Steve Zissou, The (2004)
71237	2470	1.0	Crocodile Dundee (1986)
71272	3863	1.0	Cell, The (2000)
71162	260	1.0	Star Wars: Episode IV - A New Hope (1977)
71188	784	1.0	Cable Guy, The (1996)
71216	1739	1.0	3 Ninjas: High Noon On Mega Mountain (1998)

```
71222     2005     1.0                            Goonies, The (1985)
71196     1097     1.0             E.T. the Extra-Terrestrial (1982)
71201     1219     1.0                                 Psycho (1960)
71204     1265     1.0                          Groundhog Day (1993)
71186      736     1.0                                Twister (1996)
71341     8957     1.0                                    Saw (2004)
71223     2053     0.5                Honey, I Blew Up the Kid (1992)
71165      329     0.5                  Star Trek: Generations (1994)
71213     1580     0.5               Men in Black (a.k.a. MIB) (1997)
71208     1407     0.5                                 Scream (1996)
71248     2710     0.5              Blair Witch Project, The (1999)
71310     5679     0.5                             Ring, The (2002)

                                               genres
71353                  Comedy|Crime|Mystery|Thriller
71234                       Comedy|Drama|Romance|War
71168                       Comedy|Drama|Romance|War
71330                                  Drama|Romance
71346                                 Comedy|Romance
71390                                      Drama|War
71366  Action|Adventure|Animation|Children|Comedy|Sci-Fi
71227            Adventure|Animation|Children|Drama
71303                                 Comedy|Romance
71381                         Comedy|Fantasy|Romance
71322                           Comedy|Drama|Romance
71230                           Comedy|Drama|Romance
71367               Animation|Comedy|Fantasy|Musical
71387                                 Comedy|Romance
71277                                  Drama|Romance
71386                           Comedy|Drama|Romance
71379                           Comedy|Drama|Romance
71221                                   Comedy|Drama
71239                                          Drama
71372               Adventure|Comedy|Fantasy|Romance
71241                                 Comedy|Romance
71361                                         Comedy
71164                                    Crime|Drama
71199                                          Drama
71231                               Adventure|Comedy
71370                                 Crime|Thriller
71198        Action|Adventure|Comedy|Fantasy|Romance
71190                        Children|Comedy|Romance
71184                                         Comedy
71256                                   Comedy|Drama
...                                              ...
71296                                Action|Children
71250                           Drama|Horror|Mystery
71262                          Adventure|Crime|Drama
```

```
71153                       Adventure|Children|Fantasy
71217                                   Comedy|Romance
71179                            Crime|Horror|Thriller
71224          Adventure|Children|Comedy|Fantasy|Sci-Fi
71200                         Action|Adventure|Sci-Fi
71240                          Action|Sci-Fi|Thriller
71362                                           Comedy
71340                            Comedy|Drama|Romance
71313                               Adventure|Fantasy
71345                        Adventure|Comedy|Fantasy
71237                                Adventure|Comedy
71272                            Drama|Horror|Thriller
71162                         Action|Adventure|Sci-Fi
71188                                 Comedy|Thriller
71216                          Action|Children|Comedy
71222         Action|Adventure|Children|Comedy|Fantasy
71196                           Children|Drama|Sci-Fi
71201                                    Crime|Horror
71204                          Comedy|Fantasy|Romance
71186               Action|Adventure|Romance|Thriller
71341                         Horror|Mystery|Thriller
71223                          Children|Comedy|Sci-Fi
71165                          Adventure|Drama|Sci-Fi
71213                            Action|Comedy|Sci-Fi
71208                  Comedy|Horror|Mystery|Thriller
71248                           Drama|Horror|Thriller
71310                         Horror|Mystery|Thriller

[240 rows x 4 columns]
```

In [40]:
```
%%time
user_row = user2row[user_id]
scores_argsort, scores_sort = recommend(train_ratings, user_row, best_sim_
```

```
Wall time: 1min 40s
```

In [52]:
```
# recommend movies that the user have NOT rated yet
recommend_msg(user_row, scores_argsort, scores_sort, how_many=20)
```

```
User mean rating: 2.98192771084
```

Out[52]:
```
    movieId                                        title  \
1      5752                          Gregory's Girl (1981)
2      1596                            Career Girls (1997)
3      6211                                     Ten (2002)
4      2983                       Ipcress File, The (1965)
5     70862                       It Might Get Loud (2008)
```

```
6    50685                                            Waitress (2007)
7     8014  Spring, Summer, Fall, Winter... and Spring (Bo...
8   105246              Mood Indigo (L'écume des jours) (2013)
9     5646                                             Valmont (1989)
10   37853                                    Into the Blue (2005)
11   31408                    Summer Storm (Sommersturm) (2004)
12   25868                                     Ball of Fire (1941)
13    8094                           Bad Day at Black Rock (1955)
14    1783                                            Palmetto (1998)
15  159093                                   Now You See Me 2 (2016)
16   65802                             Paul Blart: Mall Cop (2009)
17    1642       Indian Summer (a.k.a. Alive & Kicking) (1996)
18   26294   My Name Is Nobody (Il Mio nome è Nessuno) (1973)
19   47202                      Secret Life of Words, The (2005)
20    3158  Emperor and the Assassin, The (Jing ke ci qin ...

                                       genres    rating
1                              Comedy|Romance  6.274220
2                                       Drama  5.026819
3                                       Drama  5.000000
4                                    Thriller  4.856831
5                                 Documentary  4.796860
6                        Comedy|Drama|Romance  4.722129
7                                       Drama  4.548294
8                               Drama|Fantasy  4.523443
9                               Drama|Romance  4.500000
10       Action|Adventure|Crime|Thriller  4.500000
11                              Drama|Romance  4.500000
12                             Comedy|Romance  4.500000
13                     Drama|Thriller|Western  4.500000
14  Crime|Drama|Mystery|Romance|Thriller  4.500000
15                     Action|Comedy|Thriller  4.500000
16                        Action|Comedy|Crime  4.500000
17                               Comedy|Drama  4.500000
18                             Comedy|Western  4.500000
19                              Drama|Romance  4.500000
20                                      Drama  4.500000
```