In [1]:

```
# this is just to know how much time will it take to run this entire ipython notebook
from datetime import datetime
# globalstart = datetime.now()
import pandas as pd
import numpy as np
import matplotlib
matplotlib.use('nbagg')

import matplotlib.pyplot as plt
plt.rcParams.update({'figure.max_open_warning': 0})

import seaborn as sns
sns.set_style('whitegrid')
import os
from scipy import sparse
from scipy.sparse import csr_matrix

from sklearn.decomposition import TruncatedSVD
from sklearn.metrics.pairwise import cosine_similarity
import random
```

In [2]:

```
start = datetime.now()
if os.path.isfile('train_sparse_matrix.npz'):
    print("It is present in your pwd, getting it from disk....")
    # just get it from the disk instead of computing it
    train_sparse_matrix = sparse.load_npz('train_sparse_matrix.npz')
    print("DONE..")
else:
    print("We are creating sparse_matrix from the dataframe..")
    # create sparse_matrix and store it for after usage.
    # csr_matrix(data_values, (row_index, col_index), shape_of_matrix)
    # It should be in such a way that, MATRIX[row, col] = data
    train_sparse_matrix = sparse.csr_matrix((train_df.rating.values, (train_df.user.values,
                                             train_df.movie.values)),)

    print('Done. It\'s shape is : (user, movie) : ',train_sparse_matrix.shape)
    print('Saving it into disk for furthur usage..')
    # save it into disk
    sparse.save_npz("train_sparse_matrix.npz", train_sparse_matrix)
    print('Done..\n')

print(datetime.now() - start)
```

```
It is present in your pwd, getting it from disk....
DONE..
0:00:07.699237
```

In [3]:

```
start = datetime.now()
if os.path.isfile('test_sparse_matrix.npz'):
    print("It is present in your pwd, getting it from disk....")
    # just get it from the disk instead of computing it
    test_sparse_matrix = sparse.load_npz('test_sparse_matrix.npz')
    print("DONE..")
else:
    print("We are creating sparse_matrix from the dataframe..")
    # create sparse_matrix and store it for after usage.
    # csr_matrix(data_values, (row_index, col_index), shape_of_matrix)
    # It should be in such a way that, MATRIX[row, col] = data
    test_sparse_matrix = sparse.csr_matrix((test_df.rating.values, (test_df.user.values,
                                            test_df.movie.values)))

    print('Done. It\'s shape is : (user, movie) : ',test_sparse_matrix.shape)
    print('Saving it into disk for furthur usage..')
    # save it into disk
```

```
    # save it into disk
    sparse.save_npz("test_sparse_matrix.npz", test_sparse_matrix)
    print('Done..\n')

print(datetime.now() - start)
```

```
It is present in your pwd, getting it from disk....
DONE..
0:00:02.108804
```

In [4]:

```
def get_sample_sparse_matrix(sparse_matrix, no_users, no_movies, path, verbose = True):
    """
        It will get it from the ''path'' if it is present  or It will create
        and store the sampled sparse matrix in the path specified.
    """

    # get (row, col) and (rating) tuple from sparse_matrix...
    row_ind, col_ind, ratings = sparse.find(sparse_matrix)
    users = np.unique(row_ind)
    movies = np.unique(col_ind)

    print("Original Matrix : (users, movies) -- ({} {})".format(len(users), len(movies)))
    print("Original Matrix : Ratings -- {}\n".format(len(ratings)))

    # It just to make sure to get same sample everytime we run this program..
    # and pick without replacement....
    np.random.seed(15)
    sample_users = np.random.choice(users, no_users, replace=False)
    sample_movies = np.random.choice(movies, no_movies, replace=False)
    # get the boolean mask or these sampled_items in originl row/col_inds..
    mask = np.logical_and( np.isin(row_ind, sample_users),
                      np.isin(col_ind, sample_movies) )

    sample_sparse_matrix = sparse.csr_matrix((ratings[mask], (row_ind[mask], col_ind[mask])),
                                             shape=(max(sample_users)+1, max(sample_movies)+1))

    if verbose:
        print("Sampled Matrix : (users, movies) -- ({} {})".format(len(sample_users), len(sample_movies
)))
        print("Sampled Matrix : Ratings --", format(ratings[mask].shape[0]))

    print('Saving it into disk for furthur usage..')
    # save it into disk
    sparse.save_npz(path, sample_sparse_matrix)
    if verbose:
            print('Done..\n')

    return sample_sparse_matrix
```

In [5]:

```
start = datetime.now()
path = "sample_train_sparse_matrix.npz"
if os.path.isfile(path):
    print("It is present in your pwd, getting it from disk....")
    # just get it from the disk instead of computing it
    sample_train_sparse_matrix = sparse.load_npz(path)
    print("DONE..")
else:
    # get 10k users and 1k movies from available data
    sample_train_sparse_matrix = get_sample_sparse_matrix(train_sparse_matrix, no_users=25000, no_movie
s=3000,
                                             path = path)

print(datetime.now() - start)
```

```
It is present in your pwd, getting it from disk....
DONE..
0:00:00.249600
```

In [6]:

```
start = datetime.now()

path = "sample_test_sparse_matrix.npz"
if os.path.isfile(path):
    print("It is present in your pwd, getting it from disk....")
    # just get it from the disk instead of computing it
    sample_test_sparse_matrix = sparse.load_npz(path)
    print("DONE..")
else:
    # get 5k users and 500 movies from available data
    sample_test_sparse_matrix = get_sample_sparse_matrix(test_sparse_matrix, no_users=25000, no_movies=
3000,
                                                         path = path)
print(datetime.now() - start)
```

```
It is present in your pwd, getting it from disk....
DONE..
0:00:00.199865
```

In [7]:

```
# get the user averages in dictionary (key: user_id/movie_id, value: avg rating)

def get_average_ratings(sparse_matrix, of_users):

    # average ratings of user/axes
    ax = 1 if of_users else 0 # 1 - User axes,0 - Movie axes

    # ".A1" is for converting Column_Matrix to 1-D numpy array
    # axis = 0 means summing in columnwise, axis = 1 means summing in rowise
    sum_of_ratings = sparse_matrix.sum(axis=ax).A1
    # Boolean matrix of ratings ( whether a user rated that movie or not)
    is_rated = sparse_matrix!=0
    # no of ratings that each user OR movie..
    no_of_ratings = is_rated.sum(axis=ax).A1

    # max_user  and max_movie ids in sparse matrix
    u,m = sparse_matrix.shape
    # creae a dictonary of users and their average ratigns..
    average_ratings = { i : sum_of_ratings[i]/no_of_ratings[i]
                                for i in range(u if of_users else m)
                                   if no_of_ratings[i] !=0}

    # return that dictionary of average ratings
    return average_ratings
```

In [8]:

```
sample_train_averages = dict()
# get the global average of ratings in our train set.
global_average = sample_train_sparse_matrix.sum()/sample_train_sparse_matrix.count_nonzero()
sample_train_averages['global'] = global_average
sample_train_averages
sample_train_averages['user'] = get_average_ratings(sample_train_sparse_matrix, of_users=True)
print('\nAverage rating of user 1515220 :',sample_train_averages['user'][1515220])
sample_train_averages['movie'] =  get_average_ratings(sample_train_sparse_matrix, of_users=False)
print('\n AVerage rating of movie 15153 :',sample_train_averages['movie'][15153])
```

```
Average rating of user 1515220 : 3.923076923076923

 AVerage rating of movie 15153 : 2.752
```

In [9]:

```
print('\n No of ratings in Our Sampled train matrix is : {}\n'.format(sample_train_sparse_matrix.count_
nonzero()))
print('\n No of ratings in Our Sampled test  matrix is : {}\n'.format(sample_test_sparse_matrix.count_n
```

```
onzero()))
```

```
No of ratings in Our Sampled train matrix is : 856986
```

```
No of ratings in Our Sampled test  matrix is : 261693
```

For train

In [10]:

```
# get users, movies and ratings from our samples train sparse matrix
sample_train_users, sample_train_movies, sample_train_ratings = sparse.find(sample_train_sparse_matrix)
```

In [11]:

```
###############################################################
# It took me almost 10 hours to prepare this train dataset.#
###############################################################
start = datetime.now()
if os.path.isfile('reg_train.csv'):
    print("File already exists you don't have to prepare again..." )
else:
    print('preparing {} tuples for the dataset..\n'.format(len(sample_train_ratings)))
    with open('reg_train.csv', mode='w') as reg_data_file:
        count = 0
        for (user, movie, rating)  in zip(sample_train_users, sample_train_movies, sample_train_ratings
):
            st = datetime.now()
#             print(user, movie)
            #-------------------- Ratings of "movie" by similar users of "user" ---------------------
            # compute the similar Users of the "user"
            user_sim = cosine_similarity(sample_train_sparse_matrix[user], sample_train_sparse_matrix).
ravel()
            top_sim_users = user_sim.argsort()[::-1][1:] # we are ignoring 'The User' from its similar
users.
            # get the ratings of most similar users for this movie
            top_ratings = sample_train_sparse_matrix[top_sim_users, movie].toarray().ravel()
            # we will make it's length "5" by adding movie averages to .
            top_sim_users_ratings = list(top_ratings[top_ratings != 0][:5])
            top_sim_users_ratings.extend([sample_train_averages['movie'][movie]]*(5 - len(top_sim_users
_ratings)))
        #     print(top_sim_users_ratings, end=" ")


            #-------------------- Ratings by "user"  to similar movies of "movie" ---------------------
-
            # compute the similar movies of the "movie"
            movie_sim = cosine_similarity(sample_train_sparse_matrix[:,movie].T, sample_train_sparse_ma
trix.T).ravel()
            top_sim_movies = movie_sim.argsort()[::-1][1:] # we are ignoring 'The User' from its simila
r users.
            # get the ratings of most similar movie rated by this user..
            top_ratings = sample_train_sparse_matrix[user, top_sim_movies].toarray().ravel()
            # we will make it's length "5" by adding user averages to.
            top_sim_movies_ratings = list(top_ratings[top_ratings != 0][:5])
            top_sim_movies_ratings.extend([sample_train_averages['user'][user]]*(5-len(top_sim_movies_r
atings)))
        #     print(top_sim_movies_ratings, end=" : -- ")

            #-----------------prepare the row to be stores in a file-----------------#
            row = list()
            row.append(user)
            row.append(movie)
            # Now add the other features to this data...
            row.append(sample_train_averages['global']) # first feature
            # next 5 features are similar_users "movie" ratings
            row.extend(top_sim_users_ratings)
            # next 5 features are "user" ratings for similar_movies
            row.extend(top_sim_movies_ratings)
            # Avg_user_rating
```

```
            # Avg_user rating
            row.append(sample_train_averages['user'][user])
            # Avg_movie rating
            row.append(sample_train_averages['movie'][movie])

            # finalley, The actual Rating of this user-movie pair...
            row.append(rating)
            count = count + 1

            # add rows to the file opened..
            reg_data_file.write(','.join(map(str, row)))
            reg_data_file.write('\n')
            if (count)%10000 == 0:
                # print(','.join(map(str, row)))
                print("Done for {} rows----- {}".format(count, datetime.now() - start))


print(datetime.now() - start)
```

```
preparing 856986 tuples for the dataset..

Done for 10000 rows----- 1:00:16.111474
Done for 20000 rows----- 2:01:24.226434
Done for 30000 rows----- 3:01:59.348857
Done for 40000 rows----- 4:02:20.276774
Done for 50000 rows----- 5:02:42.899280
Done for 60000 rows----- 6:03:06.157685
Done for 70000 rows----- 7:03:24.127819
Done for 80000 rows----- 8:03:45.400375
Done for 90000 rows----- 9:04:05.371160
Done for 100000 rows----- 10:04:28.935813
Done for 110000 rows----- 11:04:51.049385
Done for 120000 rows----- 12:05:19.705289
Done for 130000 rows----- 13:05:37.918816
Done for 140000 rows----- 14:06:01.346077
Done for 150000 rows----- 15:06:30.446240
Done for 160000 rows----- 16:06:45.872334
Done for 170000 rows----- 17:07:11.366866
Done for 180000 rows----- 18:07:30.243634
Done for 190000 rows----- 19:08:11.344146
Done for 200000 rows----- 20:08:48.670657
Done for 210000 rows----- 21:09:31.188807
Done for 220000 rows----- 22:10:09.857719
Done for 230000 rows----- 23:10:46.165342
Done for 240000 rows----- 1 day, 0:11:30.366981
Done for 250000 rows----- 1 day, 1:12:14.193496
Done for 260000 rows----- 1 day, 2:13:05.882457
Done for 270000 rows----- 1 day, 3:13:28.502089
Done for 280000 rows----- 1 day, 4:13:58.029943
Done for 290000 rows----- 1 day, 5:14:24.478155
Done for 300000 rows----- 1 day, 6:14:53.011221
Done for 310000 rows----- 1 day, 7:15:11.866540
Done for 320000 rows----- 1 day, 8:15:32.730358
Done for 330000 rows----- 1 day, 9:15:57.712568
Done for 340000 rows----- 1 day, 10:16:17.318660
Done for 350000 rows----- 1 day, 11:16:32.864872
Done for 360000 rows----- 1 day, 12:16:53.435769
Done for 370000 rows----- 1 day, 13:17:29.276743
Done for 380000 rows----- 1 day, 14:17:56.026225
Done for 390000 rows----- 1 day, 15:18:21.829737
Done for 400000 rows----- 1 day, 16:18:46.268872
Done for 410000 rows----- 1 day, 17:19:06.259193
Done for 420000 rows----- 1 day, 18:19:33.779168
Done for 430000 rows----- 1 day, 19:20:09.704944
Done for 440000 rows----- 1 day, 20:20:37.345387
Done for 450000 rows----- 1 day, 21:21:03.881119
Done for 460000 rows----- 1 day, 22:21:32.945024
Done for 470000 rows----- 1 day, 23:22:22.019602
Done for 480000 rows----- 2 days, 0:22:48.708510
Done for 490000 rows----- 2 days, 1:23:17.809891
Done for 500000 rows----- 2 days, 2:23:48.424257
Done for 510000 rows----- 2 days, 3:24:13.532205
Done for 520000 rows----- 2 days, 4:24:49.609085
Done for 530000 rows----- 2 days, 5:25:17.058869
Done for 540000 rows----- 2 days, 6:25:42.799320
Done for 550000 rows----- 2 days, 7:26:02.522845
```

```
Done for 560000 rows----- 2 days, 8:26:20.321404
Done for 570000 rows----- 2 days, 9:26:43.312441
Done for 580000 rows----- 2 days, 10:26:58.919377
Done for 590000 rows----- 2 days, 11:27:17.262258
Done for 600000 rows----- 2 days, 12:27:44.451688
Done for 610000 rows----- 2 days, 13:28:04.333995
Done for 620000 rows----- 2 days, 14:28:25.666998
Done for 630000 rows----- 2 days, 15:30:37.438966
Done for 640000 rows----- 2 days, 16:30:49.786729
Done for 650000 rows----- 2 days, 17:30:59.133406
Done for 660000 rows----- 2 days, 18:31:23.885352
Done for 670000 rows----- 2 days, 19:31:49.587578
Done for 680000 rows----- 2 days, 20:32:13.326090
Done for 690000 rows----- 2 days, 21:32:39.499279
Done for 700000 rows----- 2 days, 22:33:00.565638
Done for 710000 rows----- 2 days, 23:33:18.891881
Done for 720000 rows----- 3 days, 0:34:13.416294
Done for 730000 rows----- 3 days, 1:45:26.643599
Done for 740000 rows----- 3 days, 2:46:52.884561
Done for 750000 rows----- 3 days, 3:47:51.848771
Done for 760000 rows----- 3 days, 4:48:26.217192
Done for 770000 rows----- 3 days, 5:49:07.982985
Done for 780000 rows----- 3 days, 6:49:46.805273
Done for 790000 rows----- 3 days, 7:50:22.033274
Done for 800000 rows----- 3 days, 8:50:54.612062
Done for 810000 rows----- 3 days, 9:51:32.268196
Done for 820000 rows----- 3 days, 10:52:10.131807
Done for 830000 rows----- 3 days, 11:52:51.480238
Done for 840000 rows----- 3 days, 12:53:22.484378
Done for 850000 rows----- 3 days, 13:53:44.850343
3 days, 14:36:01.530250
```

For test

In [10]:

```
# get users, movies and ratings from our samples train sparse matrix
sample_test_users, sample_test_movies, sample_test_ratings = sparse.find(sample_test_sparse_matrix)
```

In [11]:

```
start = datetime.now()

if os.path.isfile('reg_test.csv'):
    print("It is already created...")
else:

    print('preparing {} tuples for the dataset..\n'.format(len(sample_test_ratings)))
    with open('reg_test.csv', mode='w') as reg_data_file:
        count = 0
        for (user, movie, rating)  in zip(sample_test_users, sample_test_movies, sample_test_ratings):
            st = datetime.now()

        #-------------------- Ratings of "movie" by similar users of "user" ---------------------
            #print(user, movie)
            try:
                # compute the similar Users of the "user"
                user_sim = cosine_similarity(sample_train_sparse_matrix[user], sample_train_sparse_matr
ix).ravel()
                top_sim_users = user_sim.argsort()[::-1][1:] # we are ignoring 'The User' from its simi
lar users.
                # get the ratings of most similar users for this movie
                top_ratings = sample_train_sparse_matrix[top_sim_users, movie].toarray().ravel()
                # we will make it's length "5" by adding movie averages to .
                top_sim_users_ratings = list(top_ratings[top_ratings != 0][:5])
                top_sim_users_ratings.extend([sample_train_averages['movie'][movie]]*(5 - len(top_sim_u
sers_ratings)))
                # print(top_sim_users_ratings, end="--")

            except (IndexError, KeyError):
                # It is a new User or new Movie or there are no ratings for given user for top similar
movies...
                ########## Cold START Problem ##########
```

```
                ########## Cold Start Problem ##########
                top_sim_users_ratings.extend([sample_train_averages['global']]*(5 - len(top_sim_users_r
atings)))
                #print(top_sim_users_ratings)
            except:
                print(user, movie)
                # we just want KeyErrors to be resolved. Not every Exception...
                raise



            #-------------------- Ratings by "user"  to similar movies of "movie" ---------------------
-
            try:
                # compute the similar movies of the "movie"
                movie_sim = cosine_similarity(sample_train_sparse_matrix[:,movie].T, sample_train_spars
e_matrix.T).ravel()
                top_sim_movies = movie_sim.argsort()[::-1][1:] # we are ignoring 'The User' from its si
milar users.
                # get the ratings of most similar movie rated by this user..
                top_ratings = sample_train_sparse_matrix[user, top_sim_movies].toarray().ravel()
                # we will make it's length "5" by adding user averages to.
                top_sim_movies_ratings = list(top_ratings[top_ratings != 0][:5])
                top_sim_movies_ratings.extend([sample_train_averages['user'][user]]*(5-len(top_sim_movi
es_ratings)))
                #print(top_sim_movies_ratings)
            except (IndexError, KeyError):
                #print(top_sim_movies_ratings, end=" : -- ")
                top_sim_movies_ratings.extend([sample_train_averages['global']]*(5-len(top_sim_movies_r
atings)))
                #print(top_sim_movies_ratings)
            except :
                raise

            #-----------------prepare the row to be stores in a file-----------------#
            row = list()
            # add usser and movie name first
            row.append(user)
            row.append(movie)
            row.append(sample_train_averages['global']) # first feature
            #print(row)
            # next 5 features are similar_users "movie" ratings
            row.extend(top_sim_users_ratings)
            #print(row)
            # next 5 features are "user" ratings for similar_movies
            row.extend(top_sim_movies_ratings)
            #print(row)
            # Avg_user rating
            try:
                row.append(sample_train_averages['user'][user])
            except KeyError:
                row.append(sample_train_averages['global'])
            except:
                raise
            #print(row)
            # Avg_movie rating
            try:
                row.append(sample_train_averages['movie'][movie])
            except KeyError:
                row.append(sample_train_averages['global'])
            except:
                raise
            #print(row)
            # finalley, The actual Rating of this user-movie pair...
            row.append(rating)
            #print(row)
            count = count + 1

            # add rows to the file opened..
            reg_data_file.write(','.join(map(str, row)))
            #print(','.join(map(str, row)))
            reg_data_file.write('\n')
            if (count)%10000 == 0:
                #print(','.join(map(str, row)))
                print("Done for {} rows----- {}".format(count, datetime.now() - start))

print("", datetime.now() - start)
```

```
print("",datetime.now() - start)
```

```
preparing 261693 tuples for the dataset..

Done for 10000 rows----- 0:59:25.521386
Done for 20000 rows----- 1:57:42.951883
Done for 30000 rows----- 2:55:54.486925
Done for 40000 rows----- 3:54:18.535002
Done for 50000 rows----- 4:52:41.195504
Done for 60000 rows----- 5:51:03.183363
Done for 70000 rows----- 6:49:34.266370
Done for 80000 rows----- 7:48:00.188456
Done for 90000 rows----- 8:46:30.922874
Done for 100000 rows----- 9:44:46.557478
Done for 110000 rows----- 10:43:07.532189
Done for 120000 rows----- 11:41:28.590011
Done for 130000 rows----- 12:39:47.401122
Done for 140000 rows----- 13:38:03.682860
Done for 150000 rows----- 14:36:26.280760
Done for 160000 rows----- 15:34:45.855514
Done for 170000 rows----- 16:33:02.891295
Done for 180000 rows----- 17:31:19.850502
Done for 190000 rows----- 18:29:38.867130
Done for 200000 rows----- 19:27:57.149108
Done for 210000 rows----- 20:26:19.723569
Done for 220000 rows----- 21:24:32.582090
Done for 230000 rows----- 22:22:47.428998
Done for 240000 rows----- 23:21:11.158134
Done for 250000 rows----- 1 day, 0:19:30.153314
Done for 260000 rows----- 1 day, 1:17:39.549056
 1 day, 1:27:26.751587
```

In [ ]:

```
```