



1. Business Problem

1.1 Problem Description

Netflix is all about connecting people to the movies they love. To help customers find those movies, the recommendation system: CinematchSM. Its job is to predict whether someone will enjoy a movie based on other movies. Netflix uses those predictions to make personal movie recommendations based on each. **Cinematch** is doing pretty well, it can always be made better.

Now there are a lot of interesting alternative approaches to how Cinematch works that Netflix haven't tried. Some aren't. We're curious whether any of these can beat Cinematch by making better predictions. Because a better approach could make a big difference to our customers and our business.

Credits: <https://www.netflixprize.com/rules.html>

1.2 Problem Statement

Netflix provided a lot of anonymous rating data, and a prediction accuracy bar that is 10% better than the training data set. (Accuracy is a measurement of how closely predicted ratings of movies match subsequent user ratings.)

1.3 Sources

- <https://www.netflixprize.com/rules.html>
- <https://www.kaggle.com/netflix-inc/netflix-prize-data>
- Netflix blog: <https://medium.com/netflix-techblog/netflix-recommendations-beyond-the-5-stars-11c903d09136>
- surprise library: <http://surpriselib.com/> (we use many models from this library)
- surprise library doc: http://surprise.readthedocs.io/en/stable/getting_started.html (we use many models from this library)
- installing surprise: <https://github.com/NicolasHug/Surprise#installation>
- Research paper: <http://courses.ischool.berkeley.edu/i290-dm/s11/SECURE/a1-koren.pdf> (most of the models in Surprise are based on this paper)
- SVD Decomposition : <https://www.youtube.com/watch?v=P5mlg91as1c>

1.4 Real world/Business Objectives and constraints

Objectives:

1. Predict the rating that a user would give to a movie that he has not yet rated.
2. Minimize the difference between predicted and actual rating (RMSE and MAPE)

Constraints:

1. Some form of interpretability.

2. Machine Learning Problem

2.1 Data

2.1.1 Data Overview

Get the data from : <https://www.kaggle.com/netflix-inc/netflix-prize-data/data>

Data files :

- combined_data_1.txt
- combined_data_2.txt
- combined_data_3.txt
- combined_data_4.txt
- movie_titles.csv

The first line of each file [combined_data_1.txt, combined_data_2.txt, combined_data_3.txt,

CustomerID,Rating,Date

MovieIDs range from 1 to 17770 sequentially.

CustomerIDs range from 1 to 2649429, with gaps. There are 480189 users.

Ratings are on a five star (integral) scale from 1 to 5.

Dates have the format YYYY-MM-DD.

2.1.2 Example Data point

1:

1488844,3,2005-09-06
822109,5,2005-05-13
885013,4,2005-10-19
30878,4,2005-12-26
823519,3,2004-05-03
893988,3,2005-11-17
124105,4,2004-08-05
1248029,3,2004-04-22
1842128,4,2004-05-09
2238063,3,2005-05-11
1503895,4,2005-05-19
2207774,5,2005-06-06
2590061,3,2004-08-12
2442,3,2004-04-14
543865,4,2004-05-28
1209119,4,2004-03-23
804919,4,2004-06-10
1086807,3,2004-12-28
1711859,4,2005-05-08
372233,5,2005-11-23
1080361,3,2005-03-28
1245640,3,2005-12-19
558634,4,2004-12-14
2165002,4,2004-04-06
1181550,3,2004-02-01
1227322,4,2004-02-06
427928,4,2004-02-26
814701,5,2005-09-29
808731,4,2005-10-31
662870,5,2005-08-24
337541,5,2005-03-23
786312,3,2004-11-16
1133214,4,2004-03-07
1537427,4,2004-03-29
1209954,5,2005-05-09
2381599,3,2005-09-12
525356,2,2004-07-11
1910569,4,2004-04-12
2263586,4,2004-08-20
2421815,2,2004-02-26
1009622,1,2005-01-19

1481961,2,2005-05-24
401047,4,2005-06-03
2179073,3,2004-08-29
1434636,3,2004-05-01
93986,5,2005-10-06
1308744,5,2005-10-29
2647871,4,2005-12-30
1905581,5,2005-08-16
2508819,3,2004-05-18
1578279,1,2005-05-19
1159695,4,2005-02-15
2588432,3,2005-03-31
2423091,3,2005-09-12
470232,4,2004-04-08
2148699,2,2004-06-05
1342007,3,2004-07-16
466135,4,2004-07-13
2472440,3,2005-08-13
1283744,3,2004-04-17
1927580,4,2004-11-08
716874,5,2005-05-06
4326,4,2005-10-29

2.2 Mapping the real world problem to a Machine Learning Problem

2.2.1 Type of Machine Learning Problem

For a given movie and user we need to predict the rating would be given by him/her to the movie
The given problem is a Recommendation problem
It can also be seen as a Regression problem

2.2.2 Performance metric

- Mean Absolute Percentage Error: https://en.wikipedia.org/wiki/Mean_absolute_percentage_error
- Root Mean Square Error: https://en.wikipedia.org/wiki/Root-mean-square_deviation

2.2.3 Machine Learning Objective and Constraints

1. Minimize RMSE.
2. Try to provide some interpretability.

```
# 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
```

Double-click (or enter) to edit

3. Exploratory Data Analysis

3.1 Preprocessing

3.1.1 Converting / Merging whole data to required format: u_i, m_j, r_ij

```
start = datetime.now()
if not os.path.isfile('data.csv'):
    # Create a file 'data.csv' before reading it
    # Read all the files in netflix and store them in one big file('data.csv')
    # We're reading from each of the four files and appending each rating to a global file 'tr
    data = open('data.csv', mode='w')

    row = list()
    files=['data_folder/combined_data_1.txt','data_folder/combined_data_2.txt',
           'data_folder/combined_data_3.txt', 'data_folder/combined_data_4.txt']
```

```

for file in files:
    print("Reading ratings from {}...".format(file))
    with open(file) as f:
        for line in f:
            del row[:] # you don't have to do this.
            line = line.strip()
            if line.endswith(':'):
                # All below are ratings for this movie, until another movie appears.
                movie_id = line.replace(':', '')
            else:
                row = [x for x in line.split(',')]
                row.insert(0, movie_id)
                data.write(','.join(row))
                data.write('\n')
    print("Done.\n")
data.close()
print('Time taken :', datetime.now() - start)

```

Double-click (or enter) to edit

```

print("creating the dataframe from data.csv file..")
df = pd.read_csv('data.csv', sep=',',
                  names=['movie', 'user', 'rating', 'date'])
df.date = pd.to_datetime(df.date)
print('Done.\n')

# we are arranging the ratings according to time.
print('Sorting the dataframe by date..')
df.sort_values(by='date', inplace=True)
print('Done..')

```

 creating the dataframe from data.csv file..
Done.

Sorting the dataframe by date..
Done..

df.head()

	movie	user	rating	date
56431994	10341	510180	4	1999-11-11
9056171	1798	510180	5	1999-11-11
58698779	10774	510180	3	1999-11-11
48101611	8651	510180	2	1999-11-11
81893208	14660	510180	2	1999-11-11

```
at.describe()['rating']
```

👤 count 1.004805e+08
mean 3.604290e+00
std 1.085219e+00
min 1.000000e+00
25% 3.000000e+00
50% 4.000000e+00
75% 4.000000e+00
max 5.000000e+00
Name: rating, dtype: float64

3.1.2 Checking for NaN values

```
# just to make sure that all Nan containing rows are deleted..  
print("No of Nan values in our dataframe : ", sum(df.isnull().any()))
```

👤 No of Nan values in our dataframe : 0

Double-click (or enter) to edit

3.1.3 Removing Duplicates

```
dup_bool = df.duplicated(['movie','user','rating'])  
dups = sum(dup_bool) # by considering all columns..( including timestamp)  
print("There are {} duplicate rating entries in the data..".format(dups))
```

👤 There are 0 duplicate rating entries in the data..

Double-click (or enter) to edit

3.1.4 Basic Statistics (#Ratings, #Users, and #Movies)

```
print("Total data ")  
print("-"*50)  
print("\nTotal no of ratings : ", df.shape[0])  
print("Total No of Users : ", len(np.unique(df.user)))  
print("Total No of movies : ", len(np.unique(df.movie)))
```

👤 Total data

```
-----  
Total no of ratings : 100480507  
Total No of Users : 480189  
Total No of movies : 17770
```

3.2 Splitting data into Train and Test(80:20)

```
# if not os.path.isfile('train.csv'):
#     # create the dataframe and store it in the disk for offline purposes..
#     df.iloc[:int(df.shape[0]*0.80)].to_csv("train.csv", index=False)

# if not os.path.isfile('test.csv'):
#     # create the dataframe and store it in the disk for offline purposes..
#     df.iloc[int(df.shape[0]*0.80):].to_csv("test.csv", index=False)

train_df = pd.read_csv("train.csv", parse_dates=['date'])
test_df = pd.read_csv("test.csv")
```

3.2.1 Basic Statistics in Train data (#Ratings, #Users, and #Movies)

```
# movies = train_df.movie.value_counts()
# users = train_df.user.value_counts()
print("Training data ")
print("-"*50)
print("\nTotal no of ratings :",train_df.shape[0])
print("Total No of Users   :", len(np.unique(train_df.user)))
print("Total No of movies  :", len(np.unique(train_df.movie)))
```



Training data

```
-----  
Total no of ratings : 80384405  
Total No of Users   : 405041  
Total No of movies  : 17424
```

3.2.2 Basic Statistics in Test data (#Ratings, #Users, and #Movies)

```
print("Test data ")
print("-"*50)
print("\nTotal no of ratings :",test_df.shape[0])
print("Total No of Users   :", len(np.unique(test_df.user)))
print("Total No of movies  :", len(np.unique(test_df.movie)))
```



Test data

```
-----  
Total no of ratings : 20096102  
Total No of Users   : 349312  
Total No of movies  : 17757
```

3.3 Exploratory Data Analysis on Train data

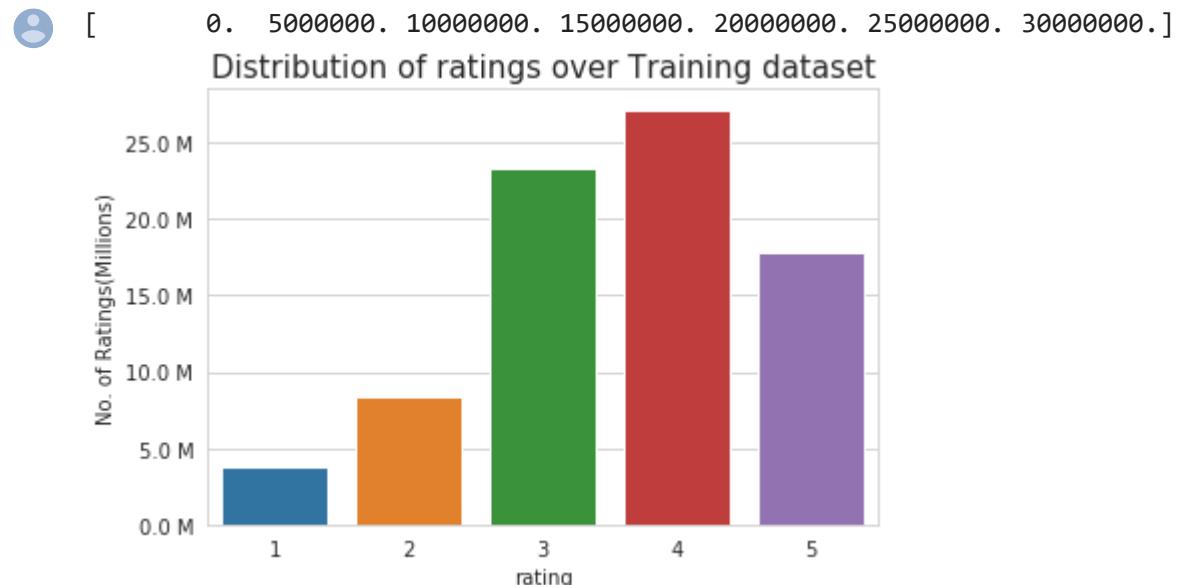
Double-click (or enter) to edit

```
# method to make y-axis more readable
def human(num, units = 'M'):
    units = units.lower()
    num = float(num)
    if units == 'k':
        return str(num/10**3) + " K"
    elif units == 'm':
        return str(num/10**6) + " M"
    elif units == 'b':
        return str(num/10**9) + " B"
```

3.3.1 Distribution of ratings

```
%matplotlib inline
fig, ax = plt.subplots()
plt.title('Distribution of ratings over Training dataset', fontsize=15)
sns.countplot(train_df.rating)
print(ax.get_yticks())
ax.set_yticklabels([human(item, 'M') for item in ax.get_yticks()])
ax.set_ylabel('No. of Ratings(Millions)')

plt.show()
```



Add new column (week day) to the data set for analysis.

```
# It is used to skip the warning ''SettingWithCopyWarning'''
```

```
pd.options.mode.chained_assignment = None # default='warn'
```

```
train_df['day_of_week'] = train_df.date.dt.weekday_name
```

```
train_df.tail()
```

	movie	user	rating	date	day_of_week
80384400	12074	2033618	4	2005-08-08	Monday
80384401	862	1797061	3	2005-08-08	Monday
80384402	10986	1498715	5	2005-08-08	Monday
80384403	14861	500016	4	2005-08-08	Monday
80384404	5926	1044015	5	2005-08-08	Monday

3.3.2 Number of Ratings per a month

```
ax = train_df.resample('m', on='date')['rating'].count().plot()
ax.set_title('No of ratings per month (Training data)')
plt.xlabel('Month')
plt.ylabel('No of ratings(per month)')
ax.set_yticklabels([human(item, 'M') for item in ax.get_yticks()])
plt.show()
```



Double-click (or enter) to edit

3.3.3 Analysis on the Ratings given by user

```
no_of_rated_movies_per_user = train_df.groupby(by='user')['rating'].count().sort_values(ascen
```

```
no_of_rated_movies_per_user.head()
```

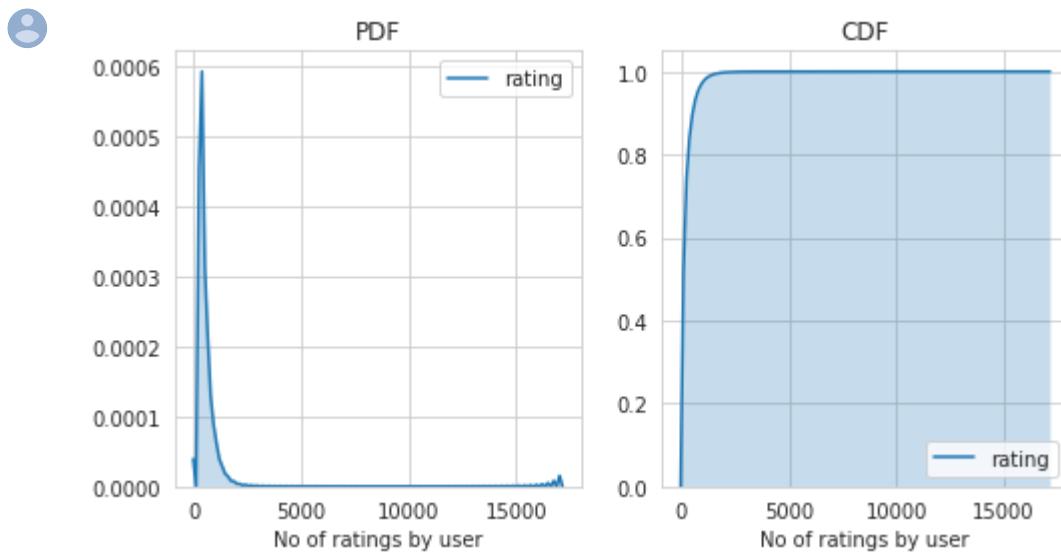
```
user
305344    17112
2439493   15896
387418    15402
1639792   9767
1461435   9447
Name: rating, dtype: int64
```

```
fig = plt.figure(figsize=plt.figaspect(.5))

ax1 = plt.subplot(121)
sns.kdeplot(no_of_rated_movies_per_user, shade=True, ax=ax1)
plt.xlabel('No of ratings by user')
plt.title("PDF")

ax2 = plt.subplot(122)
sns.kdeplot(no_of_rated_movies_per_user, shade=True, cumulative=True, ax=ax2)
plt.xlabel('No of ratings by user')
plt.title('CDF')

plt.show()
```



```
no_of_rated_movies_per_user.describe()
```

```
count    405041.000000
mean     198.459921
std      290.793238
min      1.000000
25%     34.000000
50%     89.000000
75%    245.000000
max    17112.000000
Name: rating, dtype: float64
```

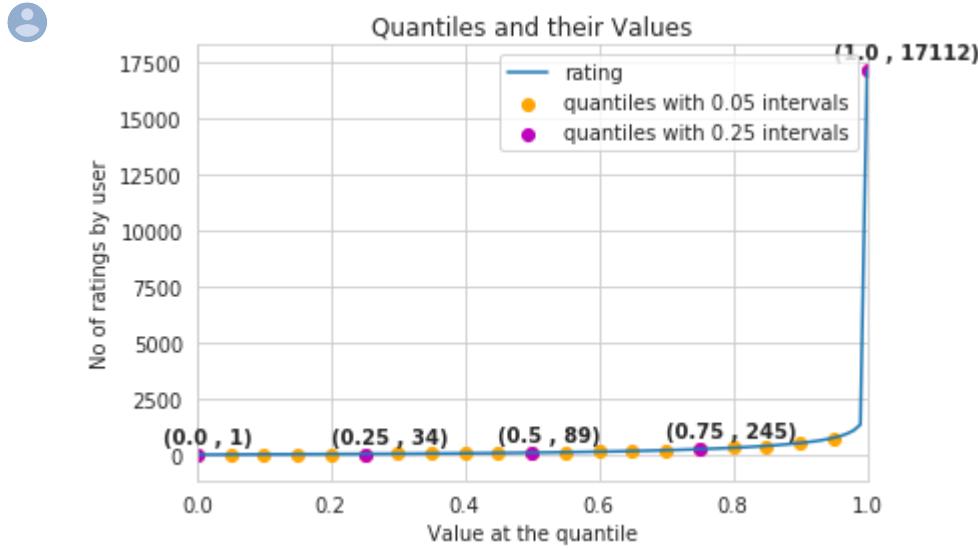
There, is something interesting going on with the quantiles..

```
quantiles = no_of_rated_movies_per_user.quantile(np.arange(0,1.01,0.01), interpolation='higher')

plt.title("Quantiles and their Values")
quantiles.plot()
# quantiles with 0.05 difference
plt.scatter(x=quantiles.index[::5], y=quantiles.values[::5], c='orange', label="quantiles with 0.05 intervals")
# quantiles with 0.25 difference
plt.scatter(x=quantiles.index[::25], y=quantiles.values[::25], c='m', label = "quantiles with 0.25 intervals")
plt.ylabel('No of ratings by user')
plt.xlabel('Value at the quantile')
plt.legend(loc='best')

# annotate the 25th, 50th, 75th and 100th percentile values....
for x,y in zip(quantiles.index[::25], quantiles[::25]):
    plt.annotate(s="({} , {})".format(x,y), xy=(x,y), xytext=(x-0.05, y+500)
                 ,fontweight='bold')

plt.show()
```



```
quantiles[::5]
```



```
0.00      1
0.05      7
0.10     15
0.15     21
0.20     27
0.25     34
0.30     41
0.35     50
0.40     60
0.45     73
0.50     89
0.55    109
0.60    133
0.65    163
0.70    199
0.75    245
0.80    307
0.85    392
0.90    520
0.95    749
1.00  17112
Name: rating, dtype: int64
```

how many ratings at the last 5% of all ratings??

```
print('\n No of ratings at last 5 percentile : {} \n'.format(sum(no_of_rated_movies_per_user>=
```



No of ratings at last 5 percentile : 20305

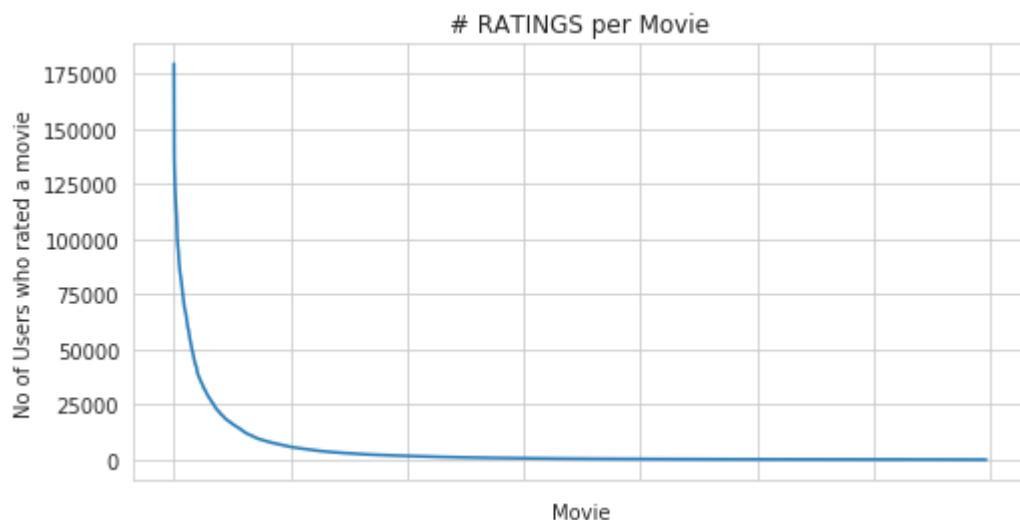
3.3.4 Analysis of ratings of a movie given by a user

```
no_of_ratings_per_movie = train_df.groupby(by='movie')['rating'].count().sort_values(ascending=False)

fig = plt.figure(figsize=plt.figaspect(.5))
ax = plt.gca()
plt.plot(no_of_ratings_per_movie.values)
plt.title('# RATINGS per Movie')
plt.xlabel('Movie')
plt.ylabel('No of Users who rated a movie')
ax.set_xticklabels([])

plt.show()
```





- **It is very skewed.. just like number of ratings given per user.**

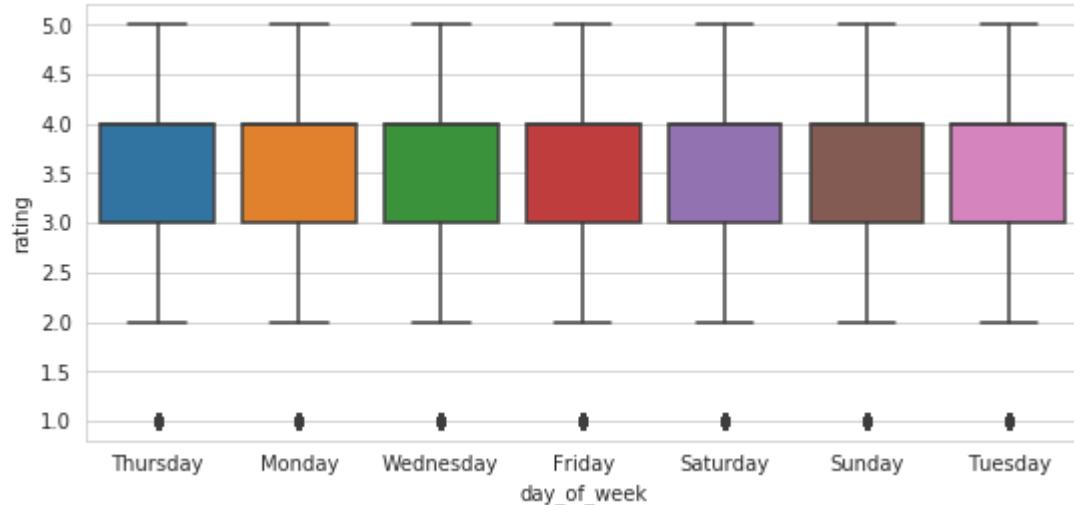
- There are some movies (which are very popular) which are rated by huge number of users.
- But most of the movies (like 90%) got some hundreds of ratings.

3.3.5 Number of ratings on each day of the week

```
fig, ax = plt.subplots()
sns.countplot(x='day_of_week', data=train_df, ax=ax)
plt.title('No of ratings on each day...')
plt.ylabel('Total no of ratings')
plt.xlabel('')
ax.set_yticklabels([human(item, 'M') for item in ax.get_yticks()])
plt.show()
```



```
start = datetime.now()
fig = plt.figure(figsize=plt.figaspect(.45))
sns.boxplot(y='rating', x='day_of_week', data=train_df)
plt.show()
print(datetime.now() - start)
```



0:00:26.629109

```
avg_week_df = train_df.groupby(by=[ 'day_of_week'])['rating'].mean()
print(" Average ratings")
print("-"*30)
print(avg_week_df)
print("\n")
```



Average ratings

```
day_of_week
Friday      3.585274
Monday      3.577250
Saturday    3.591791
Sunday      3.594144
Thursday    3.582463
Tuesday     3.574438
Wednesday   3.583751
Name: rating, dtype: float64
```

Double-click (or enter) to edit

3.3.6 Creating sparse matrix from data frame



3.3.6.1 Creating sparse matrix from train data frame

```
!wget --header="Host: doc-00-c0-docs.googleusercontent.com" --header="User-Agent: Mozilla/5.0

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..
 0:00:03.959323

The Sparsity of Train Sparse Matrix

```
us,mv = train_sparse_matrix.shape
elem = train_sparse_matrix.count_nonzero()

print("Sparsity Of Train matrix : {} % ".format( (1-(elem/(us*mv))) * 100) )
```



Sparsity Of Train matrix : 99.8292709259195 %

3.3.6.2 Creating sparse matrix from test data frame

```
!wget --header="Host: doc-0c-c0-docs.googleusercontent.com" --header="User-Agent: Mozilla/5.0
```



```
--2019-10-05 11:53:28-- https://doc-0c-c0-docs.googleusercontent.com/docs/securesc/3ss6
Resolving doc-0c-c0-docs.googleusercontent.com (doc-0c-c0-docs.googleusercontent.com)...
Connecting to doc-0c-c0-docs.googleusercontent.com (doc-0c-c0-docs.googleusercontent.com)
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/x-zip]
Saving to: 'test_sparse_matrix.npz'

test_sparse_matrix. [ <=> ] 43.45M 32.1MB/s in 1.4s

2019-10-05 11:53:40 (32.1 MB/s) - 'test_sparse_matrix.npz' saved [45559912]
```

```
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
    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..
0:00:01.095032

The Sparsity of Test data Matrix

```
us,mv = test_sparse_matrix.shape
elem = test_sparse_matrix.count_nonzero()

print("Sparsity Of Test matrix : {} % ".format( (1-(elem/(us*mv))) * 100) )

 Sparsity Of Test matrix : 99.95731772988694 %
```

3.3.7 Finding Global average of all movie ratings, Average rating per user, and Average rating per movie

```
# 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
    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
    # create a dictionary of users and their average ratings..
    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
```

3.3.7.1 finding global average of all movie ratings

```
train_averages = dict()
# get the global average of ratings in our train set.
train_global_average = train_sparse_matrix.sum()/train_sparse_matrix.count_nonzero()
train_averages['global'] = train_global_average
train_averages
```

 {'global': 3.582890686321557}

3.3.7.2 finding average rating per user

```
train_averages['user'] = get_average_ratings(train_sparse_matrix, of_users=True)
print('\nAverage rating of user 10 :',train_averages['user'][10])
```

 Average rating of user 10 : 3.3781094527363185

3.3.7.3 finding average rating per movie

```
train_averages['movie'] = get_average_ratings(train_sparse_matrix, of_users=False)
```

```
print('\n Average rating of movie 15 :',train_averages['movie'][15])
```



Average rating of movie 15 : 3.3038461538461537

Double-click (or enter) to edit

3.3.7.4 PDF's & CDF's of Avg.Ratings of Users & Movies (In Train Data)

```
start = datetime.now()
# draw pdfs for average rating per user and average
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=plt.figaspect(.5))
fig.suptitle('Avg Ratings per User and per Movie', fontsize=15)

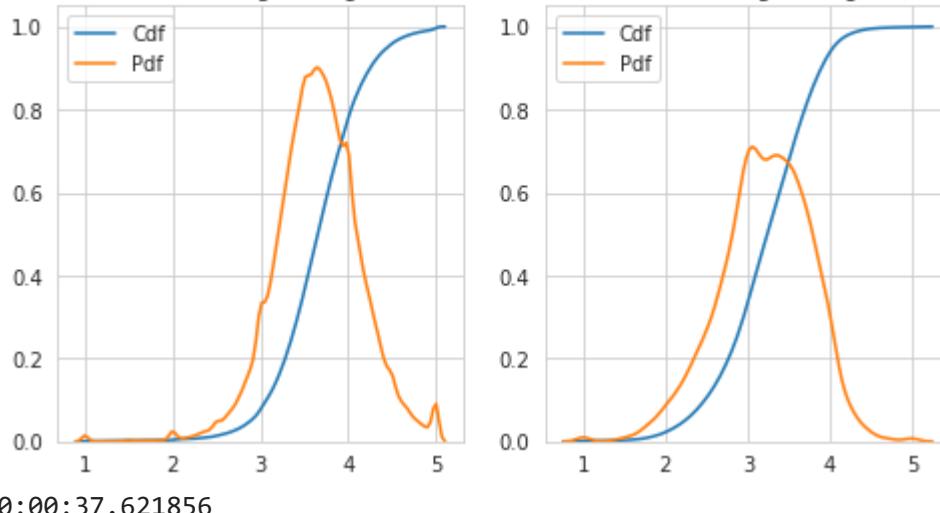
ax1.set_title('Users-Avg-Ratings')
# get the list of average user ratings from the averages dictionary..
user_averages = [rat for rat in train_averages['user'].values()]
sns.distplot(user_averages, ax=ax1, hist=False,
             kde_kws=dict(cumulative=True), label='Cdf')
sns.distplot(user_averages, ax=ax1, hist=False, label='Pdf')

ax2.set_title('Movies-Avg-Rating')
# get the list of movie_average_ratings from the dictionary..
movie_averages = [rat for rat in train_averages['movie'].values()]
sns.distplot(movie_averages, ax=ax2, hist=False,
             kde_kws=dict(cumulative=True), label='Cdf')
sns.distplot(movie_averages, ax=ax2, hist=False, label='Pdf')

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



Avg Ratings per User and per Movie
Users-Avg-Ratings Movies-Avg-Rating



0:00:37.621856

3.3.8 Cold Start problem

3.3.8.1 Cold Start problem with Users

```
total_users = len(np.unique(df.user))
users_train = len(train_averages['user'])
new_users = total_users - users_train

print('\nTotal number of Users :', total_users)
print('\nNumber of Users in Train data :', users_train)
print("\nNo of Users that didn't appear in train data: {}({} %) \n ".format(new_users,
                                                                           np.round((new_users/t
```

We might have to handle **new users (75148)** who didn't appear in train data.

3.3.8.2 Cold Start problem with Movies

```
total_movies = len(np.unique(df.movie))
movies_train = len(train_averages['movie'])
new_movies = total_movies - movies_train

print('\nTotal number of Movies :', total_movies)
print('\nNumber of Users in Train data :', movies_train)
print("\nNo of Movies that didn't appear in train data: {}({} %) \n ".format(new_movies,
                                                                           np.round((new_movies/t
```

We might have to handle **346 movies** (small comparatively) in test data

Double-click (or enter) to edit

3.4 Computing Similarity matrices

3.4.1 Computing User-User Similarity matrix

1. Calculating User User Similarity_Matrix is **not very easy**(unless you have huge Computing Power users being large).

- You can try if you want to. Your system could crash or the program stops with **Memory Error**.

3.4.1.1 Trying with all dimensions (17k dimensions per user)

```

from sklearn.metrics.pairwise import cosine_similarity

def compute_user_similarity(sparse_matrix, compute_for_few=False, top = 100, verbose=False, v
                            draw_time_taken=True):
    no_of_users, _ = sparse_matrix.shape
    # get the indices of non zero rows(users) from our sparse matrix
    row_ind, col_ind = sparse_matrix.nonzero()
    row_ind = sorted(set(row_ind)) # we don't have to
    time_taken = list() # time taken for finding similar users for an user..

    # we create rows, cols, and data lists.., which can be used to create sparse matrices
    rows, cols, data = list(), list(), list()
    if verbose: print("Computing top",top,"similarities for each user..")

    start = datetime.now()
    temp = 0

    for row in row_ind[:top] if compute_for_few else row_ind:
        temp = temp+1
        prev = datetime.now()

        # get the similarity row for this user with all other users
        sim = cosine_similarity(sparse_matrix.getrow(row), sparse_matrix).ravel()
        # We will get only the top ''top'' most similar users and ignore rest of them..
        top_sim_ind = sim.argsort()[-top:]
        top_sim_val = sim[top_sim_ind]

        # add them to our rows, cols and data
        rows.extend([row]*top)
        cols.extend(top_sim_ind)
        data.extend(top_sim_val)
        time_taken.append(datetime.now().timestamp() - prev.timestamp())
        if verbose:
            if temp%verb_for_n_rows == 0:
                print("computing done for {} users [ time elapsed : {} ]"
                      .format(temp, datetime.now()-start))

    # lets create sparse matrix out of these and return it
    if verbose: print('Creating Sparse matrix from the computed similarities')
    #return rows, cols, data

    if draw_time_taken:
        plt.plot(time_taken, label = 'time taken for each user')
        plt.plot(np.cumsum(time_taken), label='Total time')
        plt.legend(loc='best')
        plt.xlabel('User')
        plt.ylabel('Time (seconds)')
        plt.show()

```

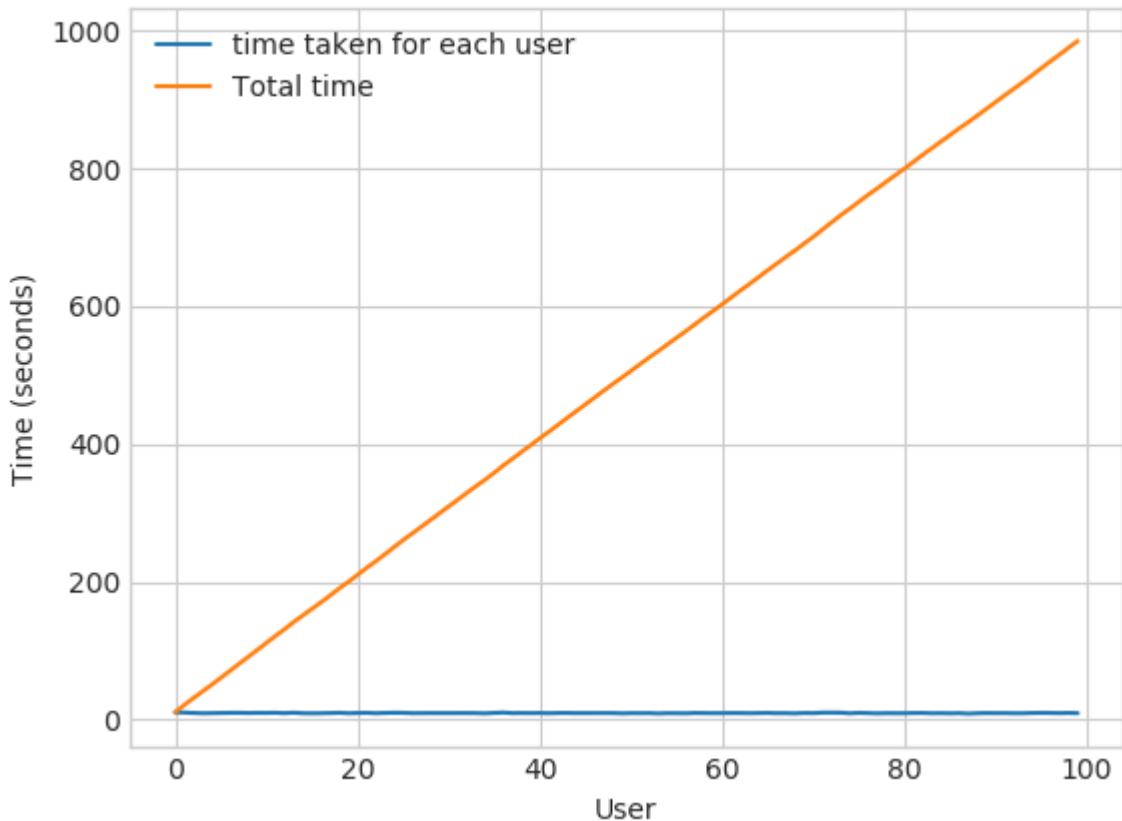
```

return sparse.csr_matrix((data, (rows, cols)), shape=(no_of_users, no_of_users)), time_ta

start = datetime.now()
u_u_sim_sparse, _ = compute_user_similarity(train_sparse_matrix, compute_for_few=True, top =
verbose=True)
print("-"*100)
print("Time taken :",datetime.now()-start)

```

👤 Computing top 100 similarities for each user..
computing done for 20 users [time elapsed : 0:03:20.300488]
computing done for 40 users [time elapsed : 0:06:38.518391]
computing done for 60 users [time elapsed : 0:09:53.143126]
computing done for 80 users [time elapsed : 0:13:10.080447]
computing done for 100 users [time elapsed : 0:16:24.711032]
Creating Sparse matrix from the computed similarities



Time taken : 0:16:33.618931

3.4.1.2 Trying with reduced dimensions (Using TruncatedSVD for dimensionality reduction)

Double-click (or enter) to edit

- We have **405,041 users** in our training set and computing similarities between them..(**17K dimensions**)

- From above plot, It took roughly **8.88 sec** for computing similar users for **one user**
- We have **405,041 users** with us in training set.
- $405041 \times 8.88 = 3596764.08 \text{ sec} = 59946.068 \text{ min} = 999.101133333 \text{ hours} = 41.62$
 - Even if we run on 4 cores parallelly (a typical system now a days), It will still take almost **1000 hours**

IDEA: Instead, we will try to reduce the dimensions using SVD, so that **it might** speed up the process.

```
from datetime import datetime
from sklearn.decomposition import TruncatedSVD

start = datetime.now()

# initialize the algorithm with some parameters..
# All of them are default except n_components. n_iter is for Randomized SVD solver.
netflix_svd = TruncatedSVD(n_components=500, algorithm='randomized', random_state=15)
trunc_svd = netflix_svd.fit_transform(train_sparse_matrix)

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

👤 0:29:07.069783

Here,

- $\sum \leftarrow (\text{netflix_svd.singular_values}_\cdot)$
- $\mathbf{V}^T \leftarrow (\text{netflix_svd.components}_\cdot)$
- \mathbf{U} is not returned. instead **Projection_of_X** onto the new vectorspace is returned.
- It uses **randomized svd** internally, which returns **All 3 of them separately**. Use that instead..

```
expl_var = np.cumsum(netflix_svd.explained_variance_ratio_)

fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=plt.figaspect(.5))

ax1.set_ylabel("Variance Explained", fontsize=15)
ax1.set_xlabel("# Latent Factors", fontsize=15)
ax1.plot(expl_var)
# annotate some (latentfactors, expl_var) to make it clear
ind = [1, 2, 4, 8, 20, 60, 100, 200, 300, 400, 500]
ax1.scatter(x = [i-1 for i in ind], y = expl_var[[i-1 for i in ind]], c='ff3300')
for i in ind:
    ax1.annotate(s = "({}, {})".format(i, np.round(expl_var[i-1], 2)), xy=(i-1, expl_var[i-1])
                 xytext = (i+20, expl_var[i-1] - 0.01), fontweight='bold')

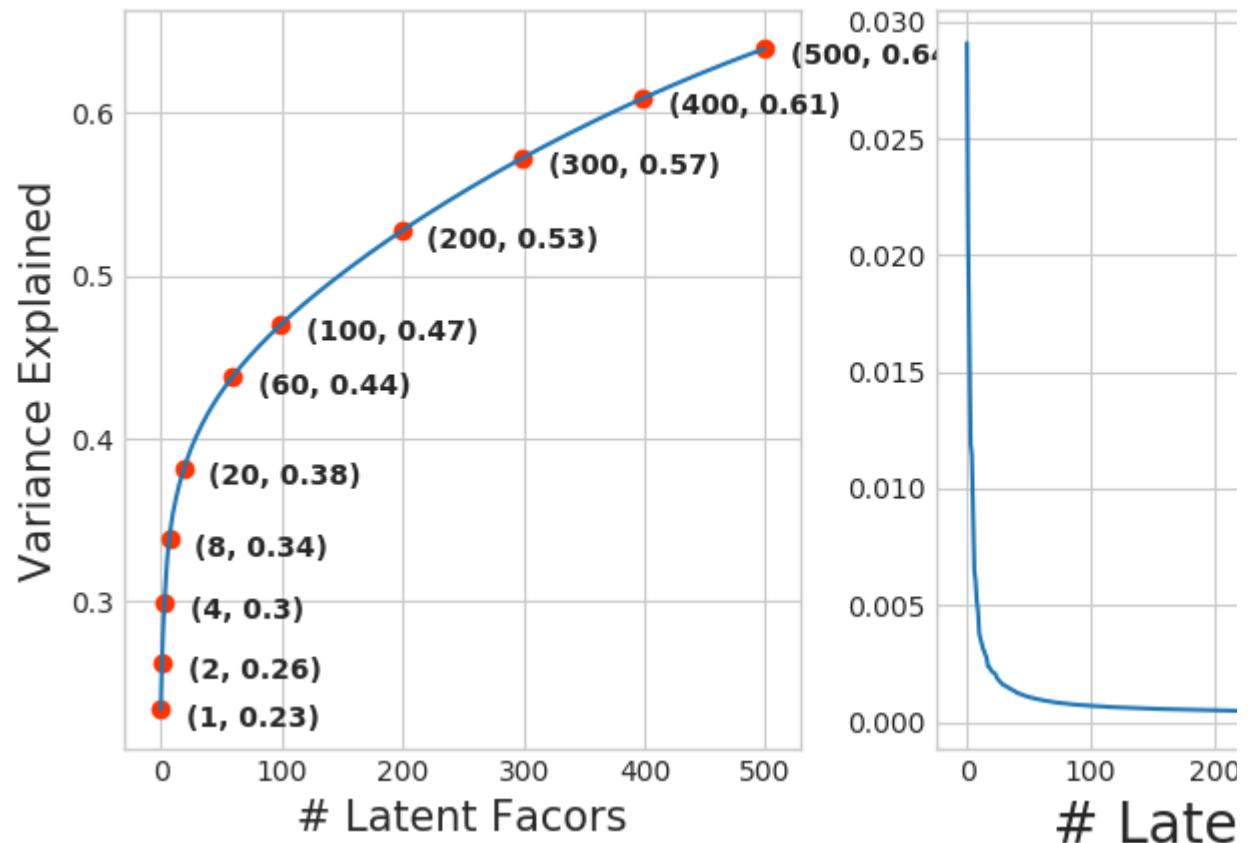
change_in_expl_var = [expl_var[i+1] - expl_var[i] for i in range(len(expl_var)-1)]
ax2.plot(change_in_expl_var)
```

```

ax2.set_ylabel("Gain in Var_Expl with One Additional LF", fontsize=10)
ax2.yaxis.set_label_position("right")
ax2.set_xlabel("# Latent Facors", fontsize=20)

plt.show()

```



```

for i in ind:
    print("({}, {})".format(i, np.round(expl_var[i-1], 2)))

```



```

(1, 0.23)
(2, 0.26)
(4, 0.3)
(8, 0.34)
(20, 0.38)
(60, 0.44)
(100, 0.47)
(200, 0.53)
(300, 0.57)
(400, 0.61)
(500, 0.64)

```

I think 500 dimensions is good enough

- By just taking **(20 to 30)** latent factors, explained variance that we could get is **20 %**.
- To take it to **60%**, we have to take **almost 400 latent factors**. It is not fare.
- It basically is the **gain of variance explained**, if we **add one additional latent factor to it**.
- By adding one by one latent factor to it, the **_gain in explained variance** with that addition is determined that way).
- **LHS Graph:**
 - **x** --- (No of latent factors),
 - **y** --- (The variance explained by taking x latent factors)
- **More decrease in the line (RHS graph) :**
 - We are getting more explained variance than before.
- **Less decrease in that line (RHS graph) :**
 - We are not getting benifitted from adding latent factor further. This is what is shown in the RHS graph.
- **RHS Graph:**
 - **x** --- (No of latent factors),
 - **y** --- (Gain n Expl_Var by taking one additional latent factor)

```
# Let's project our Original U_M matrix into into 500 Dimensional space...
start = datetime.now()
trunc_matrix = train_sparse_matrix.dot(netflix_svd.components_.T)
print(datetime.now() - start)
```

 0:00:45.670265

```
type(trunc_matrix), trunc_matrix.shape
```

 (numpy.ndarray, (2649430, 500))

- Let's convert this to actual sparse matrix and store it for future purposes

```
if not os.path.isfile('trunc_sparse_matrix.npz'):
    # create that sparse sparse matrix
    trunc_sparse_matrix = sparse.csr_matrix(trunc_matrix)
    # Save this truncated sparse matrix for later usage..
    sparse.save_npz('trunc_sparse_matrix', trunc_sparse_matrix)
else:
    trunc_sparse_matrix = sparse.load_npz('trunc_sparse_matrix.npz')
```

```
trunc_sparse_matrix.shape
```

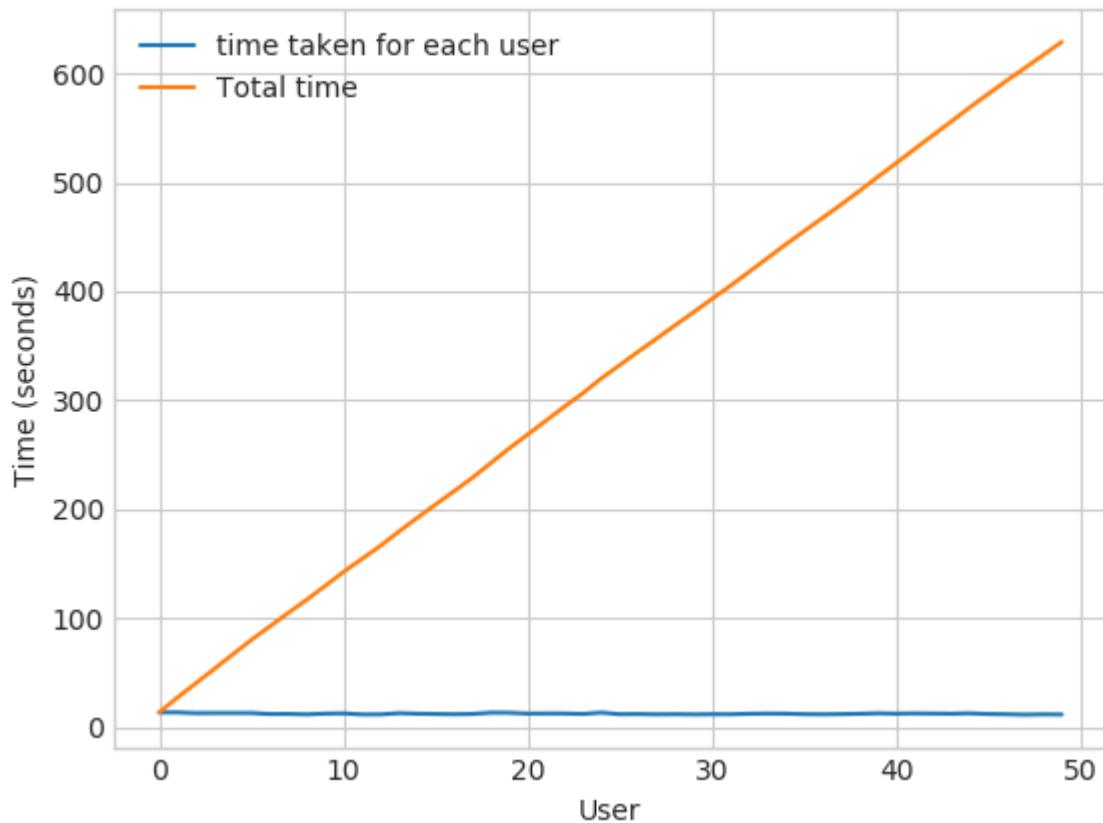
 (2649430, 500)

```

start = datetime.now()
trunc_u_u_sim_matrix, _ = compute_user_similarity(trunc_sparse_matrix, compute_for_few=True,
                                                verb_for_n_rows=10)
print("-"*50)
print("time:",datetime.now()-start)

```

Computing top 50 similarities for each user..
 computing done for 10 users [time elapsed : 0:02:09.746324]
 computing done for 20 users [time elapsed : 0:04:16.017768]
 computing done for 30 users [time elapsed : 0:06:20.861163]
 computing done for 40 users [time elapsed : 0:08:24.933316]
 computing done for 50 users [time elapsed : 0:10:28.861485]
 Creating Sparse matrix from the computed similarities



time: 0:10:52.658092

: This is taking more time for each user than Original one.

- from above plot, It took almost **12.18** for computing similar users for **one user**
- We have **405041 users** with us in training set.
- $405041 \times 12.18 = 4933399.38 \text{ sec} = 82223.323 \text{ min} = 1370.38871 \text{ h} = 57.099529861 \text{ days...}$

- Even we run on 4 cores parallelly (a typical system now a days), It will still take almost ___(

- **Why did this happen...??**

- Just think about it. It's not that difficult.

----- (sparse & dense.....get it ??) -----

Is there any other way to compute user user similarity..??

-An alternative is to compute similar users for a particular user, whenever required (**ie., Run time**) - Which tells us whether we already computed or not.. - **If not** : - Compute top (let's just say, 1000) most similar users to our datastructure, so that we can just access it(similar users) without recomputing it again. - Directly from our datastructure, which has that information. - In production time, We might have to recalculate this long time ago. Because user preferences changes over time. If we could maintain some kind of Time (recompute it). - **Which datastructure to use**: - It is purely implementation dependant. - One simple approach is to use **Dictionaries**. - **key** : *userid* - **value**: *Again a dictionary* - **key** : *Similar User* - **value**: *Similarity Value*

3.4.2 Computing Movie-Movie Similarity matrix

```
!wget --header="Host: doc-0s-c0-docs.googleusercontent.com" --header="User-Agent: Mozilla/5.0
```

```
--2019-10-05 11:54:43-- https://doc-0s-c0-docs.googleusercontent.com/docs/securesc/3ss6
Resolving doc-0s-c0-docs.googleusercontent.com (doc-0s-c0-docs.googleusercontent.com)...
Connecting to doc-0s-c0-docs.googleusercontent.com (doc-0s-c0-docs.googleusercontent.com)
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/x-zip]
Saving to: 'm_m_sim_sparse.npz'

m_m_sim_sparse.npz [=====] 2.54G 43.7MB/s in 46s

2019-10-05 11:55:29 (56.6 MB/s) - 'm_m_sim_sparse.npz' saved [2732245649]
```

```
start = datetime.now()
if not os.path.isfile('m_m_sim_sparse.npz'):
    print("It seems you don't have that file. Computing movie_movie similarity...")
    start = datetime.now()
    m_m_sim_sparse = cosine_similarity(X=train_sparse_matrix.T, dense_output=False)
    print("Done..")
    # store this sparse matrix in disk before using it. For future purposes.
    print("Saving it to disk without the need of re-computing it again.. ")
    sparse.save_npz("m_m_sim_sparse.npz", m_m_sim_sparse)
    print("Done..")
else:
```

```
print("It is there, We will get it.")
m_m_sim_sparse = sparse.load_npz("m_m_sim_sparse.npz")
print("Done ...")

print("It's a ",m_m_sim_sparse.shape," dimensional matrix")

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

It is there, We will get it.
Done ...
It's a (17771, 17771) dimensional matrix
0:00:28.862700

```
m_m_sim_sparse.shape
```

(17771, 17771)

- Even though we have similarity measure of each movie, with all other movies, We generally don't need all of them.
- Most of the times, only top_xxx similar items matters. It may be 10 or 100.
- We take only those top similar movie ratings and store them in a separate dictionary.

```
m_m_sim_sparse[17768].toarray().ravel().argsort()[:-1]
```

array([17768, 10600, 16348, ..., 16875, 5158, 0])

```
movie_ids = np.unique(m_m_sim_sparse.nonzero()[1])
```

```
start = datetime.now()
similar_movies = dict()
for movie in movie_ids:
    # get the top similar movies and store them in the dictionary
    sim_movies = m_m_sim_sparse[movie].toarray().ravel().argsort()[:-1][1:]
    similar_movies[movie] = sim_movies[:100]
print(datetime.now() - start)

# just testing similar movies for movie_15
similar_movies[15]
```

```
0:00:30.468251
array([ 8279,  8013, 16528,  5927, 13105, 12049,  4424, 10193, 17590,
       4549,  3755,  590, 14059, 15144, 15054,  9584,  9071,  6349,
      16402, 3973, 1720, 5370, 16309,  9376,  6116,  4706, 2818,
      778, 15331, 1416, 12979, 17139, 17710,  5452,  2534,  164,
     15188, 8323, 2450, 16331,  9566, 15301, 13213, 14308, 15984,
    10597, 6426, 5500, 7068, 7328,  5720,  9802,  376, 13013,
     8003, 10199, 3338, 15390,  9688, 16455, 11730,  4513,  598,
    12762, 2187, 509, 5865,  9166, 17115, 16334, 1942,  7282,
   17584, 4376, 8988, 8873, 5921, 2716, 14679, 11947, 11981,
     4649, 565, 12954, 10788, 10220, 10963,  9427, 1690,  5107,
    7859, 5969, 1510, 2429,   847,  7845,  6410, 13931,  9840,
   3706])
```

Double-click (or enter) to edit

3.4.3 Finding most similar movies using similarity matrix

Does Similarity really works as the way we expected...?__

Let's pick some random movie and check for its similar movies....

```
!wget --header="Host: doc-0k-c0-docs.googleusercontent.com" --header="User-Agent: Mozilla/5.0
```

```
--2019-10-05 12:23:19-- https://doc-0k-c0-docs.googleusercontent.com/docs/securesc/3ss6
Resolving doc-0k-c0-docs.googleusercontent.com (doc-0k-c0-docs.googleusercontent.com)...
Connecting to doc-0k-c0-docs.googleusercontent.com (doc-0k-c0-docs.googleusercontent.com)
HTTP request sent, awaiting response... 200 OK
Length: 577547 (564K) [text/csv]
Saving to: 'movie_titles.csv'

movie_titles.csv    100%[=====] 564.01K  --.-KB/s    in 0.004s

2019-10-05 12:23:20 (157 MB/s) - 'movie_titles.csv' saved [577547/577547]
```

```
# First Let's load the movie details into soe dataframe..
# movie details are in 'netflix/movie_titles.csv'
```

```
movie_titles = pd.read_csv("movie_titles.csv", sep=',', header = None,
                           names=['movie_id', 'year_of_release', 'title'], verbose=True,
                           index_col = 'movie_id', encoding = "ISO-8859-1")

movie_titles.head()
```



Tokenization took: 5.30 ms
 Type conversion took: 14.96 ms
 Parser memory cleanup took: 0.01 ms

movie_id	year_of_release	title
1	2003.0	Dinosaur Planet
2	2004.0	Isle of Man TT 2004 Review
3	1997.0	Character
4	1994.0	Paula Abdul's Get Up & Dance
5	2004.0	The Rise and Fall of ECW

Similar Movies for 'Vampire Journals'

```
mv_id = 67

print("\nMovie ----->", movie_titles.loc[mv_id].values[1])

print("\nIt has {} Ratings from users.".format(train_sparse_matrix[:,mv_id].getnnz()))

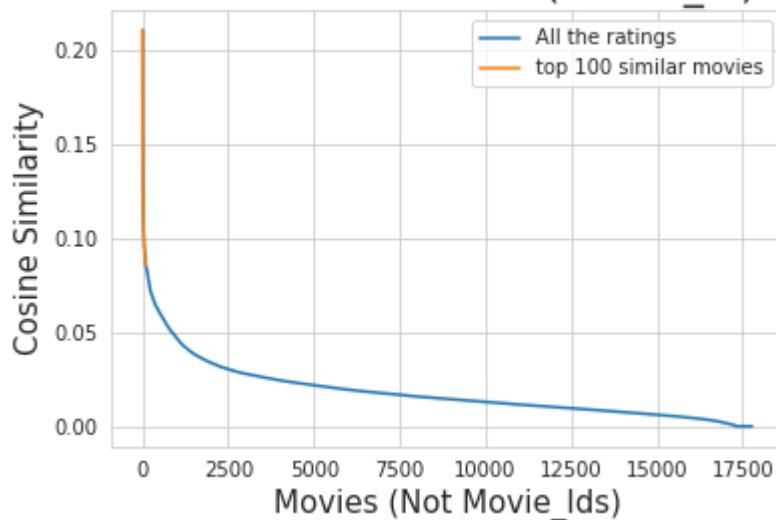
print("\nWe have {} movies which are similar to this and we will get only top most..".format(similarities = m_m_sim_sparse[mv_id].toarray().ravel())
      [similar_indices = similarities.argsort()[:-1][1:]]
      [similarities[similar_indices]]
      [sim_indices = similarities.argsort()[:-1][1:]] # It will sort and reverse the array and ignore
      # and return its indices(movie_ids)

plt.plot(similarities[sim_indices], label='All the ratings')
plt.plot(similarities[sim_indices[:100]], label='top 100 similar movies')
```

```
plt.xlabel("Movies (Not Movie_Ids)", fontsize=15)
plt.ylabel("Cosine Similarity", fontsize=15)
plt.legend()
plt.show()
```



Similar Movies of 67(movie_id)



Double-click (or enter) to edit

Double-click (or enter) to edit

Top 10 similar movies

```
movie_titles.loc[sim_indices[:10]]
```



Double-click (or enter) to edit

Similarly, we can **find similar users** and compare how similar they are.

Double-click (or enter) to edit

Double-click (or enter) to edit

16279 2002.0 Vampires: Los Muertos

4. Machine Learning Models



2000.0 Dracula Has Risen Again

```
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 -- {}".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
```

4.1 Sampling Data

4.1.1 Build sample train data from the train data

```
!wget --header="Host: doc-10-c0-docs.googleusercontent.com" --header="User-Agent: Mozilla/5.0
```

 --2019-10-06 06:47:25-- <https://doc-10-c0-docs.googleusercontent.com/docs/securesc/3ss6>
 Resolving doc-10-c0-docs.googleusercontent.com (doc-10-c0-docs.googleusercontent.com)...
 Connecting to doc-10-c0-docs.googleusercontent.com (doc-10-c0-docs.googleusercontent.com)
 HTTP request sent, awaiting response... 403 Forbidden
 2019-10-06 06:47:25 ERROR 403: Forbidden.

```
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=10000
                                                          path = path)

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

 It is present in your pwd, getting it from disk....
 DONE..
 0:00:00.047255

4.1.2 Build sample test data from the test data

```
!wget --header="Host: doc-0c-c0-docs.googleusercontent.com" --header="User-Agent: Mozilla/5.0
```



```
--2019-10-05 13:54:57-- https://doc-0c-c0-docs.googleusercontent.com/docs/securesc/3ss6
Resolving doc-0c-c0-docs.googleusercontent.com (doc-0c-c0-docs.googleusercontent.com)...
Connecting to doc-0c-c0-docs.googleusercontent.com (doc-0c-c0-docs.googleusercontent.com)
HTTP request sent, awaiting response... 200 OK
Length: 31012 (30K) [application/x-zip]
Saving to: 'sample_test_sparse_matrix.npz'

sample_test_sparse_ 100%[=====] 30.29K --.-KB/s in 0s

2019-10-05 13:55:00 (146 MB/s) - 'sample_test_sparse_matrix.npz' saved [31012/31012]
```

```
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=5000, n
                                                       path = "sample/small/sample_test_sparse_matrix.npz"
print(datetime.now() - start)
```

 It is present in your pwd, getting it from disk....
DONE..
0:00:00.039890

Double-click (or enter) to edit

4.2 Finding Global Average of all movie ratings, Average rating per User Movie (from sampled train)

```
sample_train_averages = dict()
```

4.2.1 Finding Global Average of all movie ratings

```
# 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
```

{'global': 3.581679377504138}

4.2.2 Finding Average rating per User

```
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])
```

Average rating of user 1515220 : 3.9655172413793105

4.2.3 Finding Average rating per Movie

```
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 movie 15153 : 2.6458333333333335

Double-click (or enter) to edit

4.3 Featurizing data

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

No of ratings in Our Sampled train matrix is : 129286

No of ratings in Our Sampled test matrix is : 7333

4.3.1 Featurizing data for regression problem

4.3.1.1 Featurizing train data

```
# 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)
```

```
!wget --header="Host: doc-04-c0-docs.googleusercontent.com" --header="User-Agent: Mozilla/5.0
```

...

```
--2019-10-06 09:32:04-- https://doc-04-c0-docs.googleusercontent.com/docs/securesc/3ss6
Resolving doc-04-c0-docs.googleusercontent.com (doc-04-c0-docs.googleusercontent.com)...
Connecting to doc-04-c0-docs.googleusercontent.com (doc-04-c0-docs.googleusercontent.com)
HTTP request sent, awaiting response... 403 Forbidden
2019-10-06 09:32:05 ERROR 403: Forbidden.
```

```
#####
# 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)
            top_sim_users = user_sim.argsort()[:-1][1:] # we are ignoring 'The User' from it
            # 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_matrix)
            top_sim_movies = movie_sim.argsort()[:-1][1:] # we are ignoring 'The User' from it
            # 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_ratings)))
            # 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...
            reg_data_file.write(','.join(str(i) for i in row) + '\n')

            count += 1
            if count % 100000 == 0:
                print(st - start)
                print(count)
                print('preparing {} tuples for the dataset..\n'.format(len(sample_train_ratings)))
```

```

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

# finally, 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 129286 tuples for the dataset..

```

Done for 10000 rows---- 0:53:13.974716
Done for 20000 rows---- 1:47:58.228942
Done for 30000 rows---- 2:42:46.963119
Done for 40000 rows---- 3:36:44.807894
Done for 50000 rows---- 4:28:55.311500
Done for 60000 rows---- 5:24:18.493104
Done for 70000 rows---- 6:17:39.669922
Done for 80000 rows---- 7:11:23.970879
Done for 90000 rows---- 8:05:33.787770
Done for 100000 rows---- 9:00:25.463562
Done for 110000 rows---- 9:51:28.530010
Done for 120000 rows---- 10:42:05.382141
11:30:13.699183

```

Reading from the file to make a Train_dataframe

```

reg_train = pd.read_csv('reg_train.csv', names = ['user', 'movie', 'GAvg', 'sur1', 'sur2', 's
reg_train.head()

```



	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5
0	53406	33	3.581679	4.0	5.0	5.0	4.0	1.0	5.0	2.0	5.0	3.0	1.0
1	99540	33	3.581679	5.0	5.0	5.0	4.0	5.0	3.0	4.0	4.0	3.0	5.0
2	99865	33	3.581679	5.0	5.0	4.0	5.0	3.0	5.0	4.0	4.0	5.0	4.0
3	101620	33	3.581679	2.0	3.0	5.0	5.0	4.0	4.0	3.0	3.0	4.0	5.0
4	112974	33	3.581679	5.0	5.0	5.0	5.0	5.0	3.0	5.0	5.0	5.0	3.0

- **GAvg** : Average rating of all the ratings
- **Similar users rating of this movie:**
 - sur1, sur2, sur3, sur4, sur5 (top 5 similar users who rated that movie..)
- **Similar movies rated by this user:**
 - smr1, smr2, smr3, smr4, smr5 (top 5 similar movies rated by this movie..)
- **UAvg** : User's Average rating
- **MAvg** : Average rating of this movie
- **rating** : Rating of this movie by this user.

Double-click (or enter) to edit

4.3.1.2 Featurizing test data

```
# get users, movies and ratings from the Sampled Test
sample_test_users, sample_test_movies, sample_test_ratings = sparse.find(sample_test_sparse_m

sample_train_averages['global']

 3.581679377504138

start = datetime.now()

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

    print('preparing {} tuples for the dataset..\n'.format(len(sample_test_ratings)))
    with open('sample/small/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_
    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_s
    top_sim_users = user_sim.argsort()[:-1][1:] # we are ignoring 'The User' fro
    # get the ratings of most similar users for this movie
    top_ratings = sample_train_sparse_matrix[top_sim_users, movie].toarray().rave
    # 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
    # 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 to
    ##### Cold Start Problem #####
    top_sim_users_ratings.extend([sample_train_averages['global']]*(5 - len(top_s
    # 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_t
    top_sim_movies = movie_sim.argsort()[:-1][1:] # we are ignoring 'The User' f
    # get the ratings of most similar movie rated by this user..
    top_ratings = sample_train_sparse_matrix[user, top_sim_movies].toarray().rave
    # 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(to
    # 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_si
    # 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)%1000 == 0:
    #print(','.join(map(str, row)))
    print("Done for {} rows---- {} ".format(count, datetime.now() - start))
print("",datetime.now() - start)

```



preparing 7333 tuples for the dataset..

```

Done for 1000 rows---- 0:04:29.293783
Done for 2000 rows---- 0:08:57.208002
Done for 3000 rows---- 0:13:30.333223
Done for 4000 rows---- 0:18:04.050813
Done for 5000 rows---- 0:22:38.671673
Done for 6000 rows---- 0:27:09.697009
Done for 7000 rows---- 0:31:41.933568
0:33:12.529731

```

Reading from the file to make a test dataframe

```

 --2019-10-05 14:41:14-- https://doc-0o-c0-docs.googleusercontent.com/docs/securesc/3ss6
Resolving doc-0o-c0-docs.googleusercontent.com (doc-0o-c0-docs.googleusercontent.com)...
Connecting to doc-0o-c0-docs.googleusercontent.com (doc-0o-c0-docs.googleusercontent.com)
HTTP request sent, awaiting response... 200 OK
Length: 1798931 (1.7M) [text/csv]
Saving to: 'reg_test.csv'

reg_test.csv          100%[=====] 1.71M --.-KB/s in 0.009s

2019-10-05 14:41:15 (194 MB/s) - 'reg_test.csv' saved [1798931/1798931]

```

```

reg_test_df = pd.read_csv('reg_test.csv', names = ['user', 'movie', 'GAvg', 'sur1', 'sur2', 'smr1', 'smr2', 'smr3', 'smr4', 'UAvg', 'MAvg', 'rating'], header=N
reg_test_df.head(4)

```

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1
0	808635	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679
1	941866	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679
2	1737912	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679
3	1849204	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679

- **GAvg** : Average rating of all the ratings
- **Similar users rating of this movie:**
 - sur1, sur2, sur3, sur4, sur5 (top 5 similiar users who rated that movie..)
- **Similar movies rated by this user:**
 - smr1, smr2, smr3, smr4, smr5 (top 5 similiar movies rated by this movie..)
- **UAvg** : User AVerage rating
- **MAvg** : Average rating of this movie
- **rating** : Rating of this movie by this user.

Double-click (or enter) to edit

4.3.2 Transforming data for Surprise models

```
: In [1]: !pip install scikit-surprise
```

Collecting scikit-surprise

```
  Downloading https://files.pythonhosted.org/packages/f5/da/b5700d96495fb4f092be497f0249
```

```
 |██████████| 6.5MB 3.5MB/s
```

```
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages (f
```

```
Requirement already satisfied: numpy>=1.11.2 in /usr/local/lib/python3.6/dist-packages (
```

```
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.6/dist-packages (f
```

```
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages (fr
```

```
Building wheels for collected packages: scikit-surprise
```

```
  Building wheel for scikit-surprise (setup.py) ... done
```

```
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.0-cp36-cp36m-linux_x86
```

```
  Stored in directory: /root/.cache/pip/wheels/cc/fa/16c93fccce688ae1bde7d979ff102f7b
```

```
Successfully built scikit-surprise
```

```
Installing collected packages: scikit-surprise
```

```
Successfully installed scikit-surprise-1.1.0
```

```
from surprise import Reader, Dataset
```

4.3.2.1 Transforming train data

- We can't give raw data (movie, user, rating) to train the model in Surprise library.
- They have a separate format for TRAIN and TEST data, which will be useful for training the model Surprise.
- We can form the trainset from a file, or from a Pandas DataFrame. <http://surprise.readthedocs.io/en/latest/dataframe.html>

```
# It is to specify how to read the dataframe.
# for our dataframe, we don't have to specify anything extra..
reader = Reader(rating_scale=(1,5))

# create the traindata from the dataframe...
train_data = Dataset.load_from_df(reg_train[['user', 'movie', 'rating']], reader)

# build the trainset from traindata.., It is of dataset format from surprise library..
trainset = train_data.build_full_trainset()
```

4.3.2.2 Transforming test data

- Testset is just a list of (user, movie, rating) tuples. (Order in the tuple is important)

```
testset = list(zip(reg_test_df.user.values, reg_test_df.movie.values, reg_test_df.rating.values))
testset[:3]
```



「(808635, 71, 5), (941866, 71, 4), (1737912, 71, 3)」

4.4 Applying Machine Learning models

Double-click (or enter) to edit

- Global dictionary that stores rmse and mape for all the models....
 - It stores the metrics in a dictionary of dictionaries

keys : model names(string)

value: dict(**key** : metric, **value** : value)

```
models_evaluation_train = dict()
models_evaluation_test = dict()
```



({}, {})

Double-click (or enter) to edit

Utility functions for running regression models

```

algo.fit(x_train, y_train, eval_metric = 'rmse')
print('Done. Time taken : {} \n'.format(datetime.now()-start))
print('Done \n')

# from the trained model, get the predictions....
print('Evaluating the model with TRAIN data...')
start = datetime.now()
y_train_pred = algo.predict(x_train)
# get the rmse and mape of train data...
rmse_train, mape_train = get_error_metrics(y_train.values, y_train_pred)

# store the results in train_results dictionary..
train_results = {'rmse': rmse_train,
                  'mape' : mape_train,
                  'predictions' : y_train_pred}

#####
# get the test data predictions and compute rmse and mape
print('Evaluating Test data')
y_test_pred = algo.predict(x_test)
rmse_test, mape_test = get_error_metrics(y_true=y_test.values, y_pred=y_test_pred)
# store them in our test results dictionary.
test_results = {'rmse': rmse_test,
                  'mape' : mape_test,
                  'predictions':y_test_pred}

if verbose:
    print('\nTEST DATA')
    print('-'*30)
    print('RMSE : ', rmse_test)
    print('MAPE : ', mape_test)

# return these train and test results...
return train_results, test_results

```

Utility functions for Surprise modes

```

# it is just to makesure that all of our algorithms should produce same results
# everytime they run...

my_seed = 15
random.seed(my_seed)
np.random.seed(my_seed)

#####
# get (actual_list , predicted_list) ratings given list
# of predictions (prediction is a class in Surprise).
#####
def get_ratings(predictions):

```

```

actual = np.array([pred.r_ui for pred in predictions])
pred = np.array([pred.est for pred in predictions])

return actual, pred

#####
# get 'rmse' and 'mape' , given list of prediction objecs
#####
def get_errors(predictions, print_them=False):

    actual, pred = get_ratings(predictions)
    rmse = np.sqrt(np.mean((pred - actual)**2))
    mape = np.mean(np.abs(pred - actual)/actual)

    return rmse, mape*100

#####
# It will return predicted ratings, rmse and mape of both train and test data  #
#####
def run_surprise(algo, trainset, testset, verbose=True):
    """
    It returns two dictionaries, one for train and the other is for test
    Each of them have 3 key-value pairs, which specify 'rmse', 'mape', and 'predicted'
    """

    start = datetime.now()
    # dictionaries that stores metrics for train and test..
    train = dict()
    test = dict()

    # train the algorithm with the trainset
    st = datetime.now()
    print('Training the model...')
    algo.fit(trainset)
    print('Done. time taken : {}'.format(datetime.now()-st))

    # ----- Evaluating train data-----#
    st = datetime.now()
    print('Evaluating the model with train data..')
    # get the train predictions (list of prediction class inside Surprise)
    train_preds = algo.test(trainset.build_testset())
    # get predicted ratings from the train predictions..
    train_actual_ratings, train_pred_ratings = get_ratings(train_preds)
    # get 'rmse' and 'mape' from the train predictions.
    train_rmse, train_mape = get_errors(train_preds)
    print('time taken : {}'.format(datetime.now()-st))

    if verbose:
        print('*'*15)
        print('Train Data')
        print('*'*15)

```

```

print("RMSE : {}\n\nMAPE : {}".format(train_rmse, train_mape))

#store them in the train dictionary
if verbose:
    print('adding train results in the dictionary..')
train['rmse'] = train_rmse
train['mape'] = train_mape
train['predictions'] = train_pred_ratings

#----- Evaluating Test data-----#
st = datetime.now()
print('\nEvaluating for test data...')
# get the predictions( list of prediction classes) of test data
test_preds = algo.test(testset)
# get the predicted ratings from the list of predictions
test_actual_ratings, test_pred_ratings = get_ratings(test_preds)
# get error metrics from the predicted and actual ratings
test_rmse, test_mape = get_errors(test_preds)
print('time taken : {}'.format(datetime.now()-st))

if verbose:
    print('-'*15)
    print('Test Data')
    print('-'*15)
    print("RMSE : {}\n\nMAPE : {}".format(test_rmse, test_mape))
# store them in test dictionary
if verbose:
    print('storing the test results in test dictionary...')
test['rmse'] = test_rmse
test['mape'] = test_mape
test['predictions'] = test_pred_ratings

print('\n'+'-'*45)
print('Total time taken to run this algorithm :', datetime.now() - start)

# return two dictionaries train and test
return train, test

```

Double-click (or enter) to edit

4.4.1 XGBoost with initial 13 features

```

import xgboost as xgb

# prepare Train data
x_train = reg_train.drop(['user','movie','rating'], axis=1)
y_train = reg_train['rating']

# Dropano Test data

```

```
# replace null value
x_test = reg_test_df.drop(['user', 'movie', 'rating'], axis=1)
y_test = reg_test_df['rating']

# initialize Our first XGBoost model...
first_xgb = xgb.XGBRegressor(silent=False, n_jobs=13, random_state=15, n_estimators=100)
train_results, test_results = run_xgboost(first_xgb, x_train, y_train, x_test, y_test)

# store the results in models_evaluations dictionaries
models_evaluation_train['first_algo'] = train_results
models_evaluation_test['first_algo'] = test_results

xgb.plot_importance(first_xgb)
plt.show()
```

Training the model..

```
[16:45:41] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now de
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:587: FutureWarning: Series.base i
if getattr(data, 'base', None) is not None and \
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:588: FutureWarning: Series.base i
data.base is not None and isinstance(data, np.ndarray) \
Done. Time taken : 0:00:03.715909
```

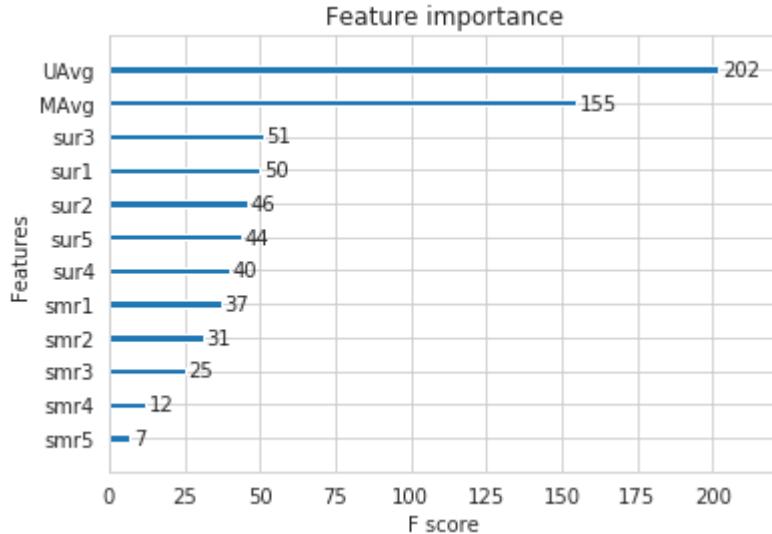
Done

Evaluating the model with TRAIN data...

Evaluating Test data

TEST DATA

```
RMSE : 1.076373581778953
MAPE : 34.48223172520999
```



Double-click (or enter) to edit

4.4.2 Surprise BaselineModel

```
from surprise import BaselineOnly
```

__Predicted_rating : (baseline prediction) __

- http://surprise.readthedocs.io/en/stable/basic_algorithms.html#surprise.prediction_algorithms.baseline_only.BaselineOnly

$$\hat{r}_{ui} = b_{ui} = \mu + b_u + b_i$$

- μ : Average of all trainings in training data.
- b_u : User bias
- b_i : Item bias (movie biases)

__Optimization function (Least Squares Problem) __

- http://surprise.readthedocs.io/en/stable/prediction_algorithms.html#baselines-estimates-configuration

$$\sum_{r_{ui} \in R_{train}} (r_{ui} - (\mu + b_u + b_i))^2 + \lambda (b_u^2 + b_i^2) . \text{ [minimize } b_u, b_i \text{]}$$

```
# options are to specify..., how to compute those user and item biases
bsl_options = {'method': 'sgd',
               'learning_rate': .001
               }
bsl_algo = BaselineOnly(bsl_options=bsl_options)
# run this algorithm..., It will return the train and test results..
bsl_train_results, bsl_test_results = run_surprise(bsl_algo, trainset, testset, verbose=True)

# Just store these error metrics in our models_evaluation datastructure
models_evaluation_train['bsl_algo'] = bsl_train_results
models_evaluation_test['bsl_algo'] = bsl_test_results
```



```

Training the model...
Estimating biases using sgd...
Done. time taken : 0:00:00.681706

Evaluating the model with train data..
time taken : 0:00:00.983806
-----
Train Data
-----
RMSE : 0.9347153928678286

MAPE : 29.389572652358183

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.187955
-----
Test Data
-----
RMSE : 1.0730330260516174

MAPE : 35.04995544572911

storing the test results in test dictionary..

-----
Total time taken to run this algorithm : 0:00:01.855434

```

Double-click (or enter) to edit

4.4.3 XGBoost with initial 13 features + Surprise Baseline predictor

Updating Train Data

```

# add our baseline_predicted value as our feature..
reg_train['bslpr'] = models_evaluation_train['bsl_algo']['predictions']
reg_train.head(2)

```

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5
0	53406	33	3.581679	4.0	5.0	5.0	4.0	1.0	5.0	2.0	5.0	3.0	1.0
1	99540	33	3.581679	5.0	5.0	5.0	4.0	5.0	3.0	4.0	4.0	3.0	5.0

Updating Test Data

```
# add that baseline predicted ratings with Surprise to the test data as well
```

```
reg_test_df['bslpr'] = models_evaluation_test['bsl_algo']['predictions']
```

```
reg_test_df.head(2)
```

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1
0	808635	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679
1	941866	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679

```
# prepare train data
```

```
x_train = reg_train.drop(['user', 'movie', 'rating'], axis=1)  
y_train = reg_train['rating']
```

```
# Prepare Test data
```

```
x_test = reg_test_df.drop(['user', 'movie', 'rating'], axis=1)  
y_test = reg_test_df['rating']
```

```
# initialize Our first XGBoost model...
```

```
xgb_bsl = xgb.XGBRegressor(silent=False, n_jobs=13, random_state=15, n_estimators=100)  
train_results, test_results = run_xgboost(xgb_bsl, x_train, y_train, x_test, y_test)
```

```
# store the results in models_evaluations dictionaries
```

```
models_evaluation_train['xgb_bsl'] = train_results  
models_evaluation_test['xgb_bsl'] = test_results
```

```
xgb.plot_importance(xgb_bsl)  
plt.show()
```



Training the model..

```
[17:58:39] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now de
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:587: FutureWarning: Series.base i
  if getattr(data, 'base', None) is not None and \
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:588: FutureWarning: Series.base i
    data.base is not None and isinstance(data, np.ndarray) \
Done. Time taken : 0:00:03.821758
```

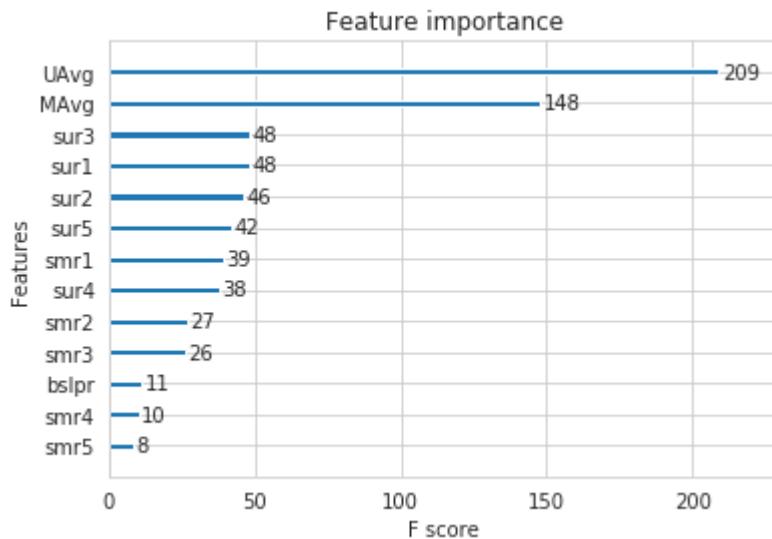
Done

Evaluating the model with TRAIN data...

Evaluating Test data

TEST DATA

RMSE : 1.0765603714651855
MAPE : 34.4648051883444



Double-click (or enter) to edit

Double-click (or enter) to edit

4.4.4 Surprise KNNBaseline predictor

```
from surprise import KNNBaseline
```

- KNN BASELINE
 - http://surprise.readthedocs.io/en/stable/knn_inspired.html#surprise.prediction_algorithms
- PEARSON_BASELINE SIMILARITY

- http://surprise.readthedocs.io/en/stable/similarities.html#surprise.similarities.pearson_b
- SHRINKAGE
 - 2.2 Neighborhood Models in <http://courses.ischool.berkeley.edu/i290-dm/s11/SECURE/a1>
- **predicted Rating** : (based on User-User similarity)

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{v \in N_i^k(u)} \text{sim}(u, v) \cdot (r_{vi} - b_{vi})}{\sum_{v \in N_i^k(u)} \text{sim}(u, v)}$$

- **b_{ui}** - Baseline prediction of (user,movie) rating
- **$N_i^k(u)$** - Set of **K similar** users (neighbours) of **user (u)** who rated **movie(i)**
- **sim (u, v)** - **Similarity** between users **u and v**
 - Generally, it will be cosine similarity or Pearson correlation coefficient.
 - But we use **shrunk Pearson-baseline correlation coefficient**, which is based on the pearson predictions instead of mean rating of user/item)

Double-click (or enter) to edit

- Predicted rating (based on Item Item similarity):

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{j \in N_u^k(i)} \text{sim}(i, j) \cdot (r_{uj} - b_{uj})}{\sum_{j \in N_u^k(i)} \text{sim}(i, j)}$$

- Notations follows same as above (user user based predicted rating)

4.4.4.1 Surprise KNNBaseline with user user similarities

```
# we specify , how to compute similarities and what to consider with sim_options to our algorithm
sim_options = {'user_based': True,
               'name': 'pearson_baseline',
               'shrinkage': 100,
               'min_support': 2
               }

# we keep other parameters like regularization parameter and learning_rate as default values.
bsl_options = {'method': 'sgd'}

knn_bsl_u = KNNBaseline(k=40, sim_options = sim_options, bsl_options = bsl_options)
knn_bsl_u_train_results, knn_bsl_u_test_results = run_surprise(knn_bsl_u, trainset, testset,
                                                              

# Just store these error metrics in our models evaluation datastructure
https://colab.research.google.com/drive/1SiovwbMeSmqZFuzo0FrzmicRh93fhPMh#scrollTo=1kmagNKcxrjf&printMode=true
```

```
models_evaluation_train['knn_bsl_u'] = knn_bsl_u_train_results
models_evaluation_test['knn_bsl_u'] = knn_bsl_u_test_results
```



```
Training the model...
Estimating biases using sgd...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Done. time taken : 0:00:33.384808

Evaluating the model with train data..
time taken : 0:01:36.952478
-----
Train Data
-----
RMSE : 0.33642097416508826

MAPE : 9.145093375416348

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.063878
-----
Test Data
-----
RMSE : 1.0726493739667242

MAPE : 35.02094499698424

storing the test results in test dictionary..

-----
Total time taken to run this algorithm : 0:02:10.403022
```

4.4.4.2 Surprise KNNBaseline with movie movie similarities

```
# we specify , how to compute similarities and what to consider with sim_options to our algos

# 'user_based' : False => this considers the similarities of movies instead of users

sim_options = {'user_based' : False,
               'name': 'pearson_baseline',
               'shrinkage': 100,
               'min_support': 2
             }

# we keep other parameters like regularization parameter and learning_rate as default values.
bsl_options = {'method': 'sgd'}

knn_bsl_m = KNNBaseline(k=40, sim_options = sim_options, bsl_options = bsl_options)

knn_bsl_m_train_results, knn_bsl_m_test_results = run_surprise(knn_bsl_m, trainset, testset,
```

```
# Just store these error metrics in our models_evaluation datastructure
models_evaluation_train['knn_bsl_m'] = knn_bsl_m_train_results
models_evaluation_test['knn_bsl_m'] = knn_bsl_m_test_results
```



Training the model...

Estimating biases using sgd...

Computing the pearson_baseline similarity matrix...

Done computing similarity matrix.

Done. time taken : 0:00:01.218644

Evaluating the model with train data..

time taken : 0:00:08.774982

Train Data

RMSE : 0.32584796251610554

MAPE : 8.447062581998374

adding train results in the dictionary..

Evaluating for test data...

time taken : 0:00:00.061034

Test Data

RMSE : 1.072758832653683

MAPE : 35.02269653015042

storing the test results in test dictionary..

Total time taken to run this algorithm : 0:00:10.056233

Double-click (or enter) to edit

4.4.5 XGBoost with initial 13 features + Surprise Baseline predictor + KNNBaseline

- ◦ ▪ First we will run XGBoost with predictions from both KNN's (that uses User_User and previous features.
- ◦ ▪ Then we will run XGBoost with just predictions from both knn models and predictions

Preparing Train data

```
# add the predicted values from both knns to this dataframe
reg_train['knn_bsl_u'] = models_evaluation_train['knn_bsl_u']['predictions']
reg_train['knn_bsl_m'] = models_evaluation_train['knn_bsl_m']['predictions']
```

```
reg_train.head(2)
```

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5
0	53406	33	3.581679	4.0	5.0	5.0	4.0	1.0	5.0	2.0	5.0	3.0	1.0
1	99540	33	3.581679	5.0	5.0	5.0	4.0	5.0	3.0	4.0	4.0	3.0	5.0

Preparing Test data

```
reg_test_df['knn_bsl_u'] = models_evaluation_test['knn_bsl_u']['predictions']
reg_test_df['knn_bsl_m'] = models_evaluation_test['knn_bsl_m']['predictions']
```

```
reg_test_df.head(2)
```

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1
0	808635	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679
1	941866	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679

```
# prepare the train data....
```

```
x_train = reg_train.drop(['user', 'movie', 'rating'], axis=1)
y_train = reg_train['rating']
```

```
# prepare the train data....
```

```
x_test = reg_test_df.drop(['user', 'movie', 'rating'], axis=1)
y_test = reg_test_df['rating']
```

```
# declare the model
```

```
xgb_knn_bsl = xgb.XGBRegressor(n_jobs=10, random_state=15)
train_results, test_results = run_xgboost(xgb_knn_bsl, x_train, y_train, x_test, y_test)
```

```
# store the results in models_evaluations dictionaries
```

```
models_evaluation_train['xgb_knn_bsl'] = train_results
models_evaluation_test['xgb_knn_bsl'] = test_results
```

```
xgb.plot_importance(xgb_knn_bsl)
plt.show()
```



Training the model..

```
[18:21:48] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now de
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:587: FutureWarning: Series.base i
  if getattr(data, 'base', None) is not None and \
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:588: FutureWarning: Series.base i
  data.base is not None and isinstance(data, np.ndarray) \
Done. Time taken : 0:00:04.385651
```

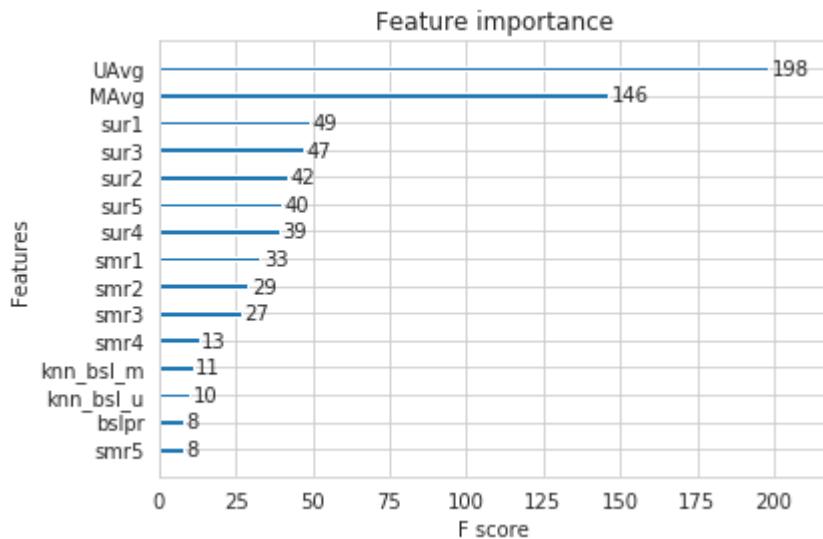
Done

Evaluating the model with TRAIN data...

Evaluating Test data

TEST DATA

RMSE : 1.0767793575625662
MAPE : 34.44745951378593



4.4.6 Matrix Factorization Techniques

4.4.6.1 SVD Matrix Factorization User Movie interactions

```
from surprise import SVD
```

http://surprise.readthedocs.io/en/stable/matrix_factorization.html#surprise.prediction_algorithms.m

- __ Predicted Rating : __

○

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u$$

- q_i - Representation of item(movie) in latent factor space
 - p_u - Representation of user in new latent factor space
- A BASIC MATRIX FACTORIZATION MODEL in <https://datajobs.com/data-science-repo/Recomm>
 - Optimization problem with user item interactions and regularization (to avoid overfitting)
 - $$\sum_{r_{ui} \in R_{train}} (r_{ui} - \hat{r}_{ui})^2 + \lambda (b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2)$$

```
# initialize the model
svd = SVD(n_factors=100, biased=True, random_state=15, verbose=True)
svd_train_results, svd_test_results = run_surprise(svd, trainset, testset, verbose=True)

# Just store these error metrics in our models_evaluation datastructure
models_evaluation_train['svd'] = svd_train_results
models_evaluation_test['svd'] = svd_test_results
```



```
Training the model...
Processing epoch 0
Processing epoch 1
Processing epoch 2
Processing epoch 3
Processing epoch 4
Processing epoch 5
Processing epoch 6
Processing epoch 7
Processing epoch 8
Processing epoch 9
Processing epoch 10
Processing epoch 11
Processing epoch 12
Processing epoch 13
Processing epoch 14
Processing epoch 15
Processing epoch 16
Processing epoch 17
Processing epoch 18
Processing epoch 19
Done. time taken : 0:00:07.982695
```

```
Evaluating the model with train data..
time taken : 0:00:01.278887
```

```
-----
Train Data
```

```
-----
RMSE : 0.6574721240954099
```

```
MAPE : 19.704901088660474
```

```
adding train results in the dictionary..
```

```
Evaluating for test data...
time taken : 0:00:00.059883
```

```
-----
Test Data
```

```
-----
RMSE : 1.0726046873826458
```

```
MAPE : 35.01953535988152
```

```
storing the test results in test dictionary..
```

```
-----
Total time taken to run this algorithm : 0:00:09.323684
```

Double-click (or enter) to edit

4.4.6.2 SVD Matrix Factorization with implicit feedback from user (user rated movies)

```
from surprise import SVDpp
```

- ----> 2.5 Implicit Feedback in <http://courses.ischool.berkeley.edu/i290-dm/s11/SECURE/a1-kore.html>
- __ Predicted Rating : __
 - - $\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(p_u + |I_u|^{-\frac{1}{2}} \sum_{j \in I_u} y_j \right)$
- I_u -- the set of all items rated by user u
- y_j -- Our new set of item factors that capture implicit ratings.
- **Optimization problem with user item interactions and regularization (to avoid overfitting)**
 - - $\sum_{r_{ui} \in R_{train}} (r_{ui} - \hat{r}_{ui})^2 + \lambda (b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2 + \|y_j\|^2)$

```
# initialize the model
svdpp = SVDpp(n_factors=50, random_state=15, verbose=True)
svdpp_train_results, svdpp_test_results = run_surprise(svdpp, trainset, testset, verbose=True

# Just store these error metrics in our models_evaluation datastructure
models_evaluation_train['svdpp'] = svdpp_train_results
models_evaluation_test['svdpp'] = svdpp_test_results
```



```
Training the model...
```

```
processing epoch 0
processing epoch 1
processing epoch 2
processing epoch 3
processing epoch 4
processing epoch 5
processing epoch 6
processing epoch 7
processing epoch 8
processing epoch 9
processing epoch 10
processing epoch 11
processing epoch 12
processing epoch 13
processing epoch 14
processing epoch 15
processing epoch 16
processing epoch 17
processing epoch 18
processing epoch 19
```

```
Done. time taken : 0:01:57.575127
```

```
Evaluating the model with train data..
```

```
time taken : 0:00:06.322517
```

```
-----
```

```
Train Data
```

```
-----
```

```
RMSE : 0.6032438403305899
```

```
MAPE : 17.49285063490268
```

```
adding train results in the dictionary..
```

```
Evaluating for test data...
```

```
time taken : 0:00:00.059972
```

```
-----
```

```
Test Data
```

```
-----
```

```
RMSE : 1.0728491944183447
```

```
MAPE : 35.03817913919887
```

```
storing the test results in test dictionary..
```

```
-----  
Total time taken to run this algorithm : 0:02:03.959658
```

Double-click (or enter) to edit

Double-click (or enter) to edit

4.4.7 XgBoost with 13 features + Surprise Baseline + Surprise KNNbaseline + MF

Preparing Train data

```
# add the predicted values from both knns to this dataframe
reg_train['svd'] = models_evaluation_train['svd']['predictions']
reg_train['svdpp'] = models_evaluation_train['svdpp']['predictions']

reg_train.head(2)
```

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5
0	53406	33	3.581679	4.0	5.0	5.0	4.0	1.0	5.0	2.0	5.0	3.0	1.0
1	99540	33	3.581679	5.0	5.0	5.0	4.0	5.0	3.0	4.0	4.0	3.0	5.0

__Preparing Test data __

```
reg_test_df['svd'] = models_evaluation_test['svd']['predictions']
reg_test_df['svdpp'] = models_evaluation_test['svdpp']['predictions']

reg_test_df.head(2)
```

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1
0	808635	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679
1	941866	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679

Double-click (or enter) to edit

```
# prepare x_train and y_train
x_train = reg_train.drop(['user', 'movie', 'rating'], axis=1)
y_train = reg_train['rating']

# prepare test data
x_test = reg_test_df.drop(['user', 'movie', 'rating'], axis=1)
y_test = reg_test_df['rating']

xgb_final = xgb.XGBRegressor(n_jobs=10, random_state=15)
train_results, test_results = run_xgboost(xgb_final, x_train, y_train, x_test, y_test)

# store the results in models_evaluations dictionaries
models_evaluation_train['xgb_final'] = train_results
```

```
models_evaluation_test['xgb_final'] = test_results
```

```
xgb.plot_importance(xgb_final)
plt.show()
```



Training the model..

```
[19:02:09] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now de
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:587: FutureWarning: Series.base i
    if getattr(data, 'base', None) is not None and \
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:588: FutureWarning: Series.base i
    data.base is not None and isinstance(data, np.ndarray) \
Done. Time taken : 0:00:05.356824
```

Done

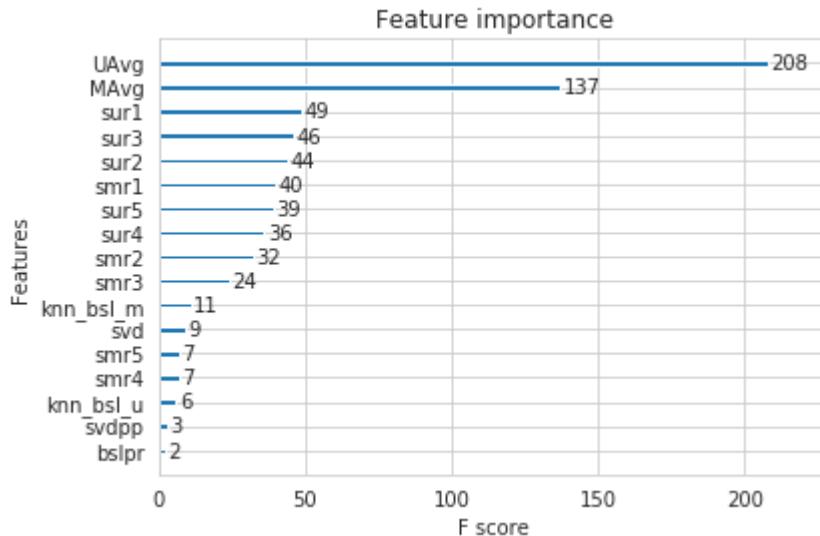
Evaluating the model with TRAIN data...

Evaluating Test data

TEST DATA

```
-----
```

RMSE : 1.0769599573828592
MAPE : 34.431788329400995



4.4.8 XgBoost with Surprise Baseline + Surprise KNNbaseline + MF Techniques

```
# prepare train data
x_train = reg_train[['knn_bsl_u', 'knn_bsl_m', 'svd', 'svdpp']]
y_train = reg_train['rating']

# test data
x_test = reg_test_df[['knn_bsl_u', 'knn_bsl_m', 'svd', 'svdpp']]
y_test = reg_test_df['rating']
```

```
xgb_all_models = xgb.XGBRegressor(n_jobs=10, random_state=15)
train_results, test_results = run_xgboost(xgb_all_models, x_train, y_train, x_test, y_test)
```

```
# store the results in models_evaluations dictionaries
models_evaluation_train['xgb_all_models'] = train_results
models_evaluation_test['xgb_all_models'] = test_results

xgb.plot_importance(xgb_all_models)
plt.show()
```

Training the model..

```
[19:02:39] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now de
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:587: FutureWarning: Series.base i
  if getattr(data, 'base', None) is not None and \
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:588: FutureWarning: Series.base i
  data.base is not None and isinstance(data, np.ndarray) \
Done. Time taken : 0:00:03.324446
```

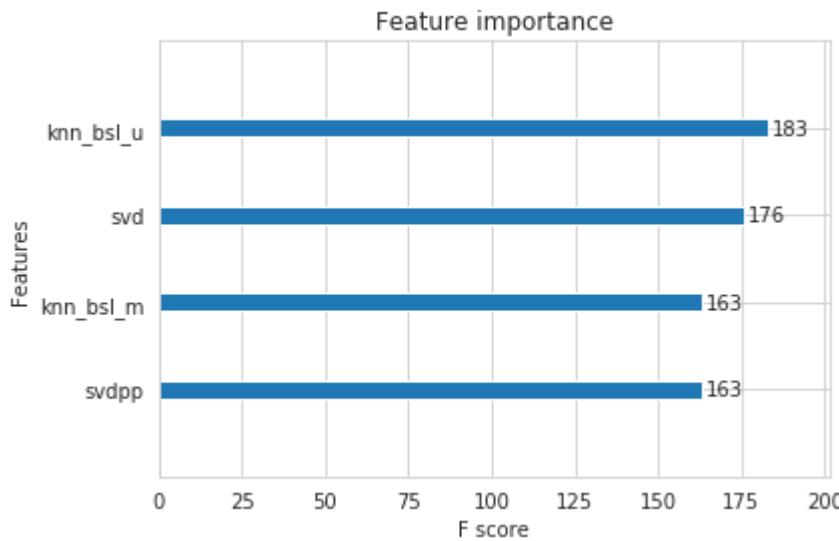
Done

Evaluating the model with TRAIN data...

Evaluating Test data

TEST DATA

```
-----  
RMSE : 1.0753047860953797  
MAPE : 35.07058962951319
```



Double-click (or enter) to edit

Double-click (or enter) to edit

4.5 Comparision between all models

```
# Saving our TEST_RESULTS into a dataframe so that you don't have to run it again
pd.DataFrame(models_evaluation_test).to_csv('small_sample_results.csv')
models = pd.read_csv('small_sample_results.csv', index_col=0)
```

```
models.loc['rmse'].sort_values()
```

svd 1.0726046873826458
knn_bsl_u 1.0726493739667242
knn_bsl_m 1.072758832653683
svdpp 1.0728491944183447
bsl_algo 1.0730330260516174
xgb_all_models 1.0753047860953797
first_algo 1.076373581778953
xgb_bsl 1.0765603714651855
xgb_knn_bsl 1.0767793575625662
xgb_final 1.0769599573828592
Name: rmse, dtype: object

Double-click (or enter) to edit

```
print("-"*100)
print("Total time taken to run this entire notebook ( with saved files) is :",datetime.now()-
```

5. Assignment

- 1.Instead of using 10K users and 1K movies to train the above models, use 25K users and 3K movies
Report the RMSE and MAPE on the test data using larger amount of data and provide a comparison b
NOTE: Please be patient as some of the code snippets make take many hours to complete execution.
- 2.Tune hyperparameters of all the Xgboost models above to improve the RMSE.

```
# 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

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..
0:00:04.361122

```
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
    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..

0:00:01.132164



```
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 -- {}".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
```

4.1 Sampling Data

4.1.1 Build sample train data from the train data

```

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
                                                            path = path)

print(datetime.now() - start)

```

 It is present in your pwd, getting it from disk....
DONE..
0:00:00.076202

4.1.2 Build sample test data from the test data

```

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=13000,
                                                          path = "sample_test_sparse_matrix.npz")
print(datetime.now() - start)

```

 It is present in your pwd, getting it from disk....
DONE..
0:00:00.036214

4.2 Finding Global Average of all movie ratings, Average rating per Use Movie (from sampled train)

```
sample_train_averages = dict()
```

4.2.1 Finding Global Average of all movie ratings

```
# 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
```

 {'global': 3.5875813607223455}

4.2.2 Finding Average rating per User

```
# 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
    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
    # create a dictionary of users and their average ratings..
    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

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])
```

 Average rating of user 1515220 : 3.923076923076923

4.2.3 Finding Average rating per Movie

```
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])
```



Double-click (or enter) to edit

4.3 Featurizing data

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



No of ratings in Our Sampled train matrix is : 856986

No of ratings in Our Sampled test matrix is : 72192

4.3.1 Featurizing data for regression problem

4.3.1.1 Featurizing train data

```
# 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)

#####
# 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)
            top_sim_users = user_sim.argsort()[:-1][1:] # we are ignoring 'The User' from it
            # 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
top_sim_movies = movie_sim.argsort()[:-1][1:] # we are ignoring 'The User' from
# 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_si
#     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
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)

```

Reading from the file to make a Train_dataframe

```

reg_train = pd.read_csv('reg_train.csv', names = ['user', 'movie', 'GAvg', 'sur1', 'sur2', 's
reg_train.head()

```



	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5
0	174683	10	3.587581	5.0	5.0	3.0	4.0	4.0	3.0	5.0	4.0	3.0	2.0
1	233949	10	3.587581	4.0	4.0	5.0	1.0	3.0	2.0	3.0	2.0	3.0	3.0
2	555770	10	3.587581	4.0	5.0	4.0	4.0	5.0	4.0	2.0	5.0	4.0	4.0
3	767518	10	3.587581	2.0	5.0	4.0	4.0	3.0	5.0	5.0	4.0	4.0	3.0
4	894393	10	3.587581	3.0	5.0	4.0	4.0	3.0	4.0	4.0	4.0	4.0	4.0

- **GAvg** : Average rating of all the ratings
- **Similar users rating of this movie:**
 - sur1, sur2, sur3, sur4, sur5 (top 5 similar users who rated that movie..)
- **Similar movies rated by this user:**
 - smr1, smr2, smr3, smr4, smr5 (top 5 similar movies rated by this movie..)
- **UAvg** : User's Average rating
- **MAvg** : Average rating of this movie
- **rating** : Rating of this movie by this user.

Double-click (or enter) to edit

4.3.1.2 Featurizing test data

```
# get users, movies and ratings from the Sampled Test
sample_test_users, sample_test_movies, sample_test_ratings = sparse.find(sample_test_sparse_m

sample_train_averages['global']

 3.5875813607223455

start = datetime.now()

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

    print('preparing {} tuples for the dataset..\n'.format(len(sample_test_ratings)))
    with open('sample/small/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_
    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_s
    top_sim_users = user_sim.argsort()[:-1][1:] # we are ignoring 'The User' fro
    # get the ratings of most similar users for this movie
    top_ratings = sample_train_sparse_matrix[top_sim_users, movie].toarray().rave
    # 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
    # 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 to
    ##### Cold Start Problem #####
    top_sim_users_ratings.extend([sample_train_averages['global']]*(5 - len(top_s
    # 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_t
    top_sim_movies = movie_sim.argsort()[:-1][1:] # we are ignoring 'The User' f
    # get the ratings of most similar movie rated by this user..
    top_ratings = sample_train_sparse_matrix[user, top_sim_movies].toarray().rave
    # 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(to
    # 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_si
    # 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)%1000 == 0:
    #print(','.join(map(str, row)))
    print("Done for {} rows---- {} ".format(count, datetime.now() - start))
print("",datetime.now() - start)

```

__Reading from the file to make a test dataframe __

```

reg_test_df = pd.read_csv('reg_test.csv', names = ['user', 'movie', 'GAvg', 'sur1', 'sur2', 'smr1', 'smr2', 'smr3', 'smr4', 'UAvg', 'MAvg', 'rating'], header=None)
reg_test_df.astype({'UAvg': 'float64'}).dtypes
reg_test_df.head()

```



	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1
0	808635	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679
1	941866	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679
2	1737912	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679
3	1849204	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679
4	28572	111	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679

- **GAvg** : Average rating of all the ratings
- **Similar users rating of this movie:**
 - sur1, sur2, sur3, sur4, sur5 (top 5 simiular users who rated that movie..)
- **Similar movies rated by this user:**
 - smr1, smr2, smr3, smr4, smr5 (top 5 simiular movies rated by this movie..)
- **UAvg** : User AVerage rating
- **MAvg** : Average rating of this movie
- **rating** : Rating of this movie by this user.

4.3.2 Transforming data for Surprise models

```
from surprise import Reader, Dataset
```

4.3.2.1 Transforming train data

- We can't give raw data (movie, user, rating) to train the model in Surprise library.
- They have a saperate format for TRAIN and TEST data, which will be useful for training the mod Surprise.
- We can form the trainset from a file, or from a Pandas DataFrame. <http://surprise.readthedocs.io/en/latest/dataframe.html>

```
# It is to specify how to read the dataframe.
# for our dataframe, we don't have to specify anything extra..
reader = Reader(rating_scale=(1,5))

# create the trainset from the dataframe
https://colab.research.google.com/drive/1SiovwbMeSmqZFuzo0FrzmicRh93fhPMh#scrollTo=1kmagNKcxrf&printMode=true
```

```
# Create the traindata from the dataset...
train_data = Dataset.load_from_df(reg_train[['user', 'movie', 'rating']], reader)

# build the trainset from traindata.., It is of dataset format from surprise library..
trainset = train_data.build_full_trainset()
```

4.3.2.2 Transforming test data

- Testset is just a list of (user, movie, rating) tuples. (Order in the tuple is important)

```
testset = list(zip(reg_test_df.user.values, reg_test_df.movie.values, reg_test_df.rating.values))
testset[:3]
```

👤 [(808635, 71, 5), (941866, 71, 4), (1737912, 71, 3)]

4.4 Applying Machine Learning models

- Global dictionary that stores rmse and mape for all the models....
 - It stores the metrics in a dictionary of dictionaries

 | **keys** : model names(string)

 | **value**: dict(**key** : metric, **value** : value)

```
models_evaluation_train = dict()
models_evaluation_test = dict()

models_evaluation_train, models_evaluation_test
```

👤 ({}, {})

Double-click (or enter) to edit

Utility functions for running regression models

```
# to get rmse and mape given actual and predicted ratings..
def get_error_metrics(y_true, y_pred):
    rmse = np.sqrt(np.mean([(y_true[i] - y_pred[i])**2 for i in range(len(y_pred))]))
    mape = np.mean(np.abs((y_true - y_pred)/y_true)) * 100
    return rmse, mape

#####
#####
```

```
def run_xgboost(algo, x_train, y_train, x_test, y_test, verbose=True):
    """
    It will return train_results and test_results
    """

    # dictionaries for storing train and test results
    train_results = dict()
    test_results = dict()

    # fit the model
    print('Training the model..')
    start = datetime.now()
    algo.fit(x_train, y_train, eval_metric = 'rmse')
    print('Done. Time taken : {} \n'.format(datetime.now()-start))
    print('Done \n')

    # from the trained model, get the predictions....
    print('Evaluating the model with TRAIN data..')
    start = datetime.now()
    y_train_pred = algo.predict(x_train)
    # get the rmse and mape of train data...
    rmse_train, mape_train = get_error_metrics(y_train.values, y_train_pred)

    # store the results in train_results dictionary..
    train_results = {'rmse': rmse_train,
                     'mape' : mape_train,
                     'predictions' : y_train_pred}

    #####
    # get the test data predictions and compute rmse and mape
    print('Evaluating Test data')
    y_test_pred = algo.predict(x_test)
    rmse_test, mape_test = get_error_metrics(y_true=y_test.values, y_pred=y_test_pred)
    # store them in our test results dictionary.
    test_results = {'rmse': rmse_test,
                     'mape' : mape_test,
                     'predictions':y_test_pred}

    if verbose:
        print('\nTEST DATA')
        print('-'*30)
        print('RMSE : ', rmse_test)
        print('MAPE : ', mape_test)

    # return these train and test results...
    return train_results, test_results
```

Utility functions for Surprise modes

```

# it is just to makesure that all of our algorithms should produce same results
# everytime they run...

my_seed = 15
random.seed(my_seed)
np.random.seed(my_seed)

#####
# get (actual_list , predicted_list) ratings given list
# of predictions (prediction is a class in Surprise).
#####
def get_ratings(predictions):
    actual = np.array([pred.r_ui for pred in predictions])
    pred = np.array([pred.est for pred in predictions])

    return actual, pred

#####
# get ''rmse'' and ''mape'' , given list of prediction objecs
#####
def get_errors(predictions, print_them=False):

    actual, pred = get_ratings(predictions)
    rmse = np.sqrt(np.mean((pred - actual)**2))
    mape = np.mean(np.abs(pred - actual)/actual)

    return rmse, mape*100

#####
# It will return predicted ratings, rmse and mape of both train and test data  #
#####
def run_surprise(algo, trainset, testset, verbose=True):
    ...
    return train_dict, test_dict

    It returns two dictionaries, one for train and the other is for test
    Each of them have 3 key-value pairs, which specify ''rmse'', ''mape'', and ''predicte
    ...
    start = datetime.now()
    # dictionaries that stores metrics for train and test..
    train = dict()
    test = dict()

    # train the algorithm with the trainset
    st = datetime.now()
    print('Training the model...')
    algo.fit(trainset)
    print('Done. time taken : {} \n'.format(datetime.now()-st))

    # ----- Evaluating train data-----#
    st = datetime.now()
    print('Evaluating the model with train data')

```

```

print( evaluating the model with train data.. )

# get the train predictions (list of prediction class inside Surprise)
train_preds = algo.test(trainset.build_testset())
# get predicted ratings from the train predictions..
train_actual_ratings, train_pred_ratings = get_ratings(train_preds)
# get ''rmse'' and ''mape'' from the train predictions.
train_rmse, train_mape = get_errors(train_preds)
print('time taken : {}'.format(datetime.now()-st))

if verbose:
    print('-'*15)
    print('Train Data')
    print('-'*15)
    print("RMSE : {}\nMAPE : {}".format(train_rmse, train_mape))

#store them in the train dictionary
if verbose:
    print('adding train results in the dictionary..')
train['rmse'] = train_rmse
train['mape'] = train_mape
train['predictions'] = train_pred_ratings

#----- Evaluating Test data-----#
st = datetime.now()
print('\nEvaluating for test data...')
# get the predictions( list of prediction classes) of test data
test_preds = algo.test(testset)
# get the predicted ratings from the list of predictions
test_actual_ratings, test_pred_ratings = get_ratings(test_preds)
# get error metrics from the predicted and actual ratings
test_rmse, test_mape = get_errors(test_preds)
print('time taken : {}'.format(datetime.now()-st))

if verbose:
    print('-'*15)
    print('Test Data')
    print('-'*15)
    print("RMSE : {}\nMAPE : {}".format(test_rmse, test_mape))
# store them in test dictionary
if verbose:
    print('storing the test results in test dictionary...')
test['rmse'] = test_rmse
test['mape'] = test_mape
test['predictions'] = test_pred_ratings

print('\n'+'-'*45)
print('Total time taken to run this algorithm :', datetime.now() - start)

# return two dictionaries train and test
return train, test

```

Double-click (or enter) to edit

4.4.1 XGBoost with initial 13 features

```
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import TimeSeriesSplit

import warnings
warnings.filterwarnings('ignore')

parameters = {'max_depth':[1,2,3],
              'learning_rate':[0.001,0.01,0.1],
              'n_estimators':[100,300,500,700]}

# prepare Train data
x_train = reg_train.drop(['user','movie','rating'], axis=1)
y_train = reg_train['rating']
# Prepare Test data
x_test = reg_test_df.drop(['user','movie','rating'], axis=1)
y_test = reg_test_df['rating']

start = datetime.now()
# initialize Our first XGBoost model...
first_xgb = xgb.XGBRegressor(nthread=-1,objective = 'reg:squarederror')

# Perform cross validation
gscv = GridSearchCV(first_xgb,
                     param_grid = parameters,
                     scoring="neg_mean_squared_error",
                     cv = TimeSeriesSplit(n_splits=2),
                     n_jobs=-1,
                     verbose = 1)
gscv_result = gscv.fit(x_train, y_train)

# Summarize results
print("Best: %f using %s" % (gscv_result.best_score_, gscv_result.best_params_))
means = gscv_result.cv_results_['mean_test_score']
stds = gscv_result.cv_results_['std_test_score']
params = gscv_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))

print("\nTime Taken: ",start - datetime.now())
```



Fitting 2 folds for each of 36 candidates, totalling 72 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.

[Parallel(n_jobs=-1)]: Done 42 tasks | elapsed: 7.8min

[Parallel(n_jobs=-1)]: Done 72 out of 72 | elapsed: 15.0min finished

Best: -0.776973 using {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 700}

-9.013130 (0.126672) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 100}

-6.421049 (0.123222) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 300}

-4.673640 (0.113159) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 500}

-3.491194 (0.099686) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 700}

-8.994388 (0.122451) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 100}

-6.371245 (0.110112) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 300}

-4.603119 (0.097108) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 500}

-3.410694 (0.085758) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 700}

-8.976351 (0.118237) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 100}

-6.334924 (0.100342) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 300}

-4.558995 (0.090792) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 500}

-3.360381 (0.080070) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 700}

-2.377304 (0.078300) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 100}

-0.953497 (0.023238) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 300}

-0.864898 (0.015496) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 500}

-0.831809 (0.013428) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 700}

-2.289245 (0.070158) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 100}

-0.871308 (0.021769) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 300}

-0.806107 (0.013422) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 500}

-0.791131 (0.011465) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 700}

-2.236558 (0.066640) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 100}

-0.839010 (0.018284) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 300}

-0.789730 (0.012023) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 500}

-0.782739 (0.011157) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 700}

-0.804835 (0.011545) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 100}

-0.783283 (0.009779) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 300}

-0.782941 (0.009871) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 500}

-0.783033 (0.009998) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 700}

-0.784743 (0.010360) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 100}

-0.780550 (0.011309) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 300}

-0.779785 (0.012077) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 500}

-0.779517 (0.012667) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 700}

-0.781427 (0.011464) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100}

-0.778373 (0.013473) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 300}

-0.777506 (0.015495) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 500}

-0.776973 (0.017370) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 700}

Time Taken: -1 day, 23:44:02.229826

```
first_xgb = xgb.XGBRegressor(max_depth=3,learning_rate = 0.1,n_estimators=700,nthread=-1,obje
first_xgb
```



```
%matplotlib inline
train_results, test_results = run_xgboost(first_xgb, x_train, y_train, x_test, y_test)

# store the results in models_evaluations dictionaries
models_evaluation_train['first_algo'] = train_results
models_evaluation_test['first_algo'] = test_results

xgb.plot_importance(first_xgb)
plt.show()
```

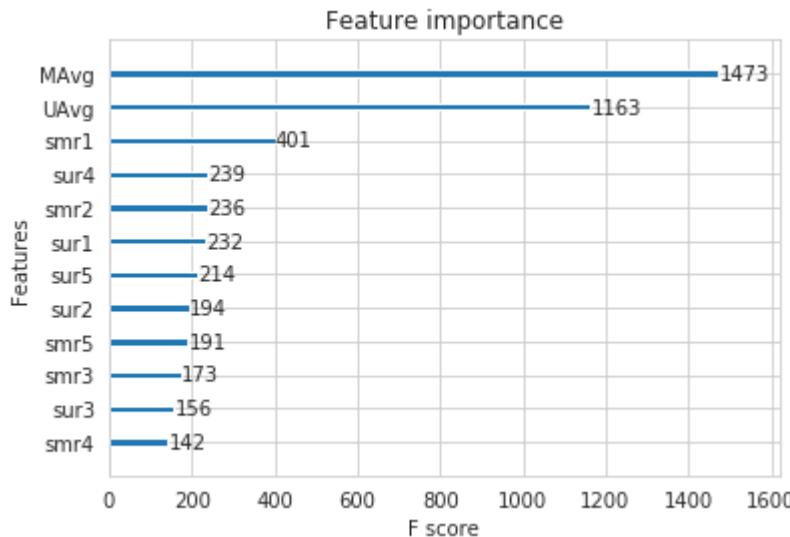
Training the model..
Done. Time taken : 0:00:58.989537

Done

Evaluating the model with TRAIN data...
Evaluating Test data

TEST DATA

RMSE : 1.073628401472761
MAPE : 34.805427336563845



Double-click (or enter) to edit

4.4.2 Surprise BaselineModel

```
from surprise import BaselineOnly
```

__Predicted_rating : (baseline prediction) __

- http://surprise.readthedocs.io/en/stable/basic_algorithms.html#surprise.prediction_algorithms.basel

$$\hat{r}_{ui} = b_{ui} = \mu + b_u + b_i$$

- μ : Average of all trainings in training data.
- b_u : User bias
- b_i : Item bias (movie biases)

— Optimization function (Least Squares Problem) —

- http://surprise.readthedocs.io/en/stable/prediction_algorithms.html#baselines-estimates-configuration

$$\sum_{r_{ui} \in R_{train}} (r_{ui} - (\mu + b_u + b_i))^2 + \lambda (b_u^2 + b_i^2). \text{ [minimize } b_u, b_i]$$

```
# options are to specify.., how to compute those user and item biases
bsl_options = {'method': 'sgd',
               'learning_rate': .001
               }
bsl_algo = BaselineOnly(bsl_options=bsl_options)
# run this algorithm.., It will return the train and test results..
bsl_train_results, bsl_test_results = run_surprise(bsl_algo, trainset, testset, verbose=True)

# Just store these error metrics in our models_evaluation datastructure
models_evaluation_train['bsl_algo'] = bsl_train_results
models_evaluation_test['bsl_algo'] = bsl_test_results
```



```

Training the model...
Estimating biases using sgd...
Done. time taken : 0:00:03.056706

Evaluating the model with train data..
time taken : 0:00:03.325025
-----
Train Data
-----
RMSE : 0.9515545717145464

MAPE : 30.285873291608517

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.066601
-----
Test Data
-----
RMSE : 1.073153983662934

MAPE : 34.91398760097454

storing the test results in test dictionary..

-----
Total time taken to run this algorithm : 0:00:06.450748

```

4.4.3 XGBoost with initial 13 features + Surprise Baseline predictor

Updating Train Data

```

# add our baseline_predicted value as our feature..
reg_train['bslpr'] = models_evaluation_train['bsl_algo']['predictions']
reg_train.head(2)

```

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5
0	174683	10.0	3.587581	5.0	5.0	3.0	4.0	4.0	3.0	5.0	4.0	3.0	2.0
1	233949	10.0	3.587581	4.0	4.0	5.0	1.0	3.0	2.0	3.0	2.0	3.0	2.0

Updating Test Data

```

# add that baseline predicted ratings with Surprise to the test data as well
reg_test_df['bslpr'] = models_evaluation_test['bsl_algo']['predictions']

reg test df.head(2)

```



	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1
0	808635	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679
1	941866	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679

```
import warnings
warnings.filterwarnings('ignore')

parameters = {'max_depth':[1,2,3],
              'learning_rate':[0.001,0.01,0.1],
              'n_estimators':[100,300,500,700]}

# prepare Train data
x_train = reg_train.drop(['user','movie','rating'], axis=1)
y_train = reg_train['rating']
# Prepare Test data
x_test = reg_test_df.drop(['user','movie','rating'], axis=1)
y_test = reg_test_df['rating']

start = datetime.now()
# initialize Our first XGBoost model...
xgb = xgb.XGBRegressor(nthread=-1,objective ='reg:squarederror')

# Perform cross validation
gscv = GridSearchCV(xgb,
                     param_grid = parameters,
                     scoring="neg_mean_squared_error",
                     cv = TimeSeriesSplit(n_splits=2),
                     n_jobs=-1,
                     verbose = 1)
gscv_result = gscv.fit(x_train, y_train)

# Summarize results
print("Best: %f using %s" % (gscv_result.best_score_, gscv_result.best_params_))
means = gscv_result.cv_results_['mean_test_score']
stds = gscv_result.cv_results_['std_test_score']
params = gscv_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))

print("\nTime Taken: ",start - datetime.now())
```



Fitting 2 folds for each of 36 candidates, totalling 72 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.

[Parallel(n_jobs=-1)]: Done 42 tasks | elapsed: 8.9min

[Parallel(n_jobs=-1)]: Done 72 out of 72 | elapsed: 16.9min finished

Best: -0.778214 using {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 500}

-9.013130 (0.126672) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 100}

-6.421049 (0.123222) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 300}

-4.673640 (0.113159) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 500}

-3.491194 (0.099686) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 700}

-8.994388 (0.122451) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 100}

-6.371245 (0.110112) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 300}

-4.603119 (0.097108) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 500}

-3.410694 (0.085758) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 700}

-8.976351 (0.118237) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 100}

-6.334924 (0.100342) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 300}

-4.558995 (0.090792) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 500}

-3.360381 (0.080070) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 700}

-2.377304 (0.078300) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 100}

-0.953497 (0.023238) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 300}

-0.864898 (0.015496) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 500}

-0.831809 (0.013428) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 700}

-2.289245 (0.070158) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 100}

-0.871308 (0.021769) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 300}

-0.806107 (0.013422) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 500}

-0.791131 (0.011465) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 700}

-2.236558 (0.066640) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 100}

-0.839010 (0.018284) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 300}

-0.789738 (0.012031) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 500}

-0.782768 (0.011186) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 700}

-0.804835 (0.011545) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 100}

-0.783283 (0.009779) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 300}

-0.782950 (0.009874) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 500}

-0.783025 (0.009990) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 700}

-0.784743 (0.010360) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 100}

-0.780718 (0.011484) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 300}

-0.779817 (0.011944) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 500}

-0.779754 (0.012555) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 700}

-0.781530 (0.011520) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100}

-0.779123 (0.013686) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 300}

-0.778214 (0.015032) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 500}

-0.778219 (0.016096) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 700}

Time Taken: -1 day, 23:42:09.685811

```
import xgboost as xgb
xgb_bsl = xgb.XGBRegressor(max_depth=3,learning_rate = 0.1,n_estimators=500,nthread=-1,object
xgb_bsl
```



```

XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bynode=1, gamma=0
train_results, test_results = run_xgboost(xgb_bsl, x_train, y_train, x_test, y_test)

# store the results in models_evaluations dictionaries
models_evaluation_train['xgb_bsl'] = train_results
models_evaluation_test['xgb_bsl'] = test_results

xgb.plot_importance(xgb_bsl)
plt.show()

```

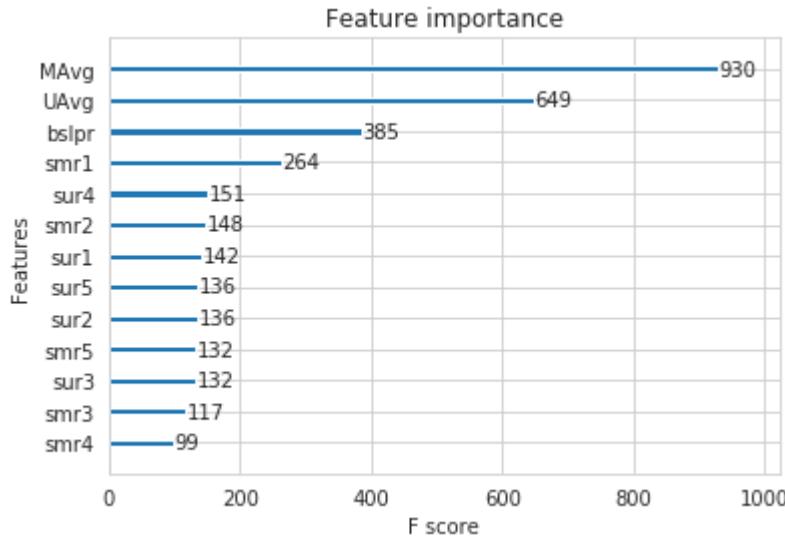
Training the model..
Done. Time taken : 0:00:55.231051

Done

Evaluating the model with TRAIN data...
Evaluating Test data

TEST DATA

RMSE : 1.0731537024868656
MAPE : 34.951827448711356



4.4.4 Surprise KNNBaseline predictor

```
from surprise import KNNBaseline
```

- KNN BASELINE
 - http://surprise.readthedocs.io/en/stable/knn_inspired.html#surprise.prediction_algorithm
- PEARSON_BASELINE SIMILARITY
 - http://surprise.readthedocs.io/en/stable/similarities.html#surprise.similarities.pearson_b

- SHRINKAGE

- 2.2 Neighborhood Models in <http://courses.ischool.berkeley.edu/i290-dm/s11/SECURE/a1>

- **predicted Rating** : (based on User-User similarity)

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{v \in N_i^k(u)} \text{sim}(u, v) \cdot (r_{vi} - b_{vi})}{\sum_{v \in N_i^k(u)} \text{sim}(u, v)}$$

- **b_{ui}** - Baseline prediction of (user,movie) rating
- **$N_i^k(u)$** - Set of **K similar** users (neighbours) of **user (u)** who rated **movie(i)**
- **sim (u, v)** - **Similarity** between users **u and v**
 - Generally, it will be cosine similarity or Pearson correlation coefficient.
 - But we use **shrunk Pearson-baseline correlation coefficient**, which is based on the pearson predictions instead of mean rating of user/item)

Double-click (or enter) to edit

- Predicted rating (based on Item Item similarity):

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{j \in N_u^k(i)} \text{sim}(i, j) \cdot (r_{uj} - b_{uj})}{\sum_{j \in N_u^k(i)} \text{sim}(i, j)}$$

- Notations follows same as above (user user based predicted rating)

4.4.4.1 Surprise KNNBaseline with user user similarities

```
# we specify , how to compute similarities and what to consider with sim_options to our algorithm
sim_options = {'user_based': True,
               'name': 'pearson_baseline',
               'shrinkage': 100,
               'min_support': 2
             }

# we keep other parameters like regularization parameter and learning_rate as default values.
bsl_options = {'method': 'sgd'}
```

`knn_bsl_u = KNNBaseline(k=40, sim_options = sim_options, bsl_options = bsl_options)`

`knn_bsl_u_train_results, knn_bsl_u_test_results = run_surprise(knn_bsl_u, trainset, testset,`

`# Just store these error metrics in our models evaluation datastructure`

<https://colab.research.google.com/drive/1SiovwbMeSmqZFuza0FrzmicRh93fhPMh#scrollTo=1kmagNKcxrjf&printMode=true>

```
models_evaluation_train['knn_bsl_u'] = knn_bsl_u_train_results
models_evaluation_test['knn_bsl_u'] = knn_bsl_u_test_results
```



```
Training the model...
Estimating biases using sgd...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Done. time taken : 0:05:49.541130

Evaluating the model with train data..
time taken : 0:27:04.607174
-----
Train Data
-----
RMSE : 0.4046789299639105

MAPE : 10.882540650588512

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.099489
-----
Test Data
-----
RMSE : 1.072783288145299

MAPE : 34.865902909936445

storing the test results in test dictionary..

-----
Total time taken to run this algorithm : 0:32:54.250485
```

4.4.4.2 Surprise KNNBaseline with movie movie similarities

```
# we specify , how to compute similarities and what to consider with sim_options to our algos

# 'user_based' : False => this considers the similarities of movies instead of users

sim_options = {'user_based' : False,
               'name': 'pearson_baseline',
               'shrinkage': 100,
               'min_support': 2
             }

# we keep other parameters like regularization parameter and learning_rate as default values.
bsl_options = {'method': 'sgd'}

knn_bsl_m = KNNBaseline(k=40, sim_options = sim_options, bsl_options = bsl_options)

knn_bsl_m_train_results, knn_bsl_m_test_results = run_surprise(knn_bsl_m, trainset, testset,
```

```
# Just store these error metrics in our models_evaluation datastructure
models_evaluation_train['knn_bsl_m'] = knn_bsl_m_train_results
models_evaluation_test['knn_bsl_m'] = knn_bsl_m_test_results
```



Training the model...

Estimating biases using sgd...

Computing the pearson_baseline similarity matrix...

Done computing similarity matrix.

Done. time taken : 0:00:04.024821

Evaluating the model with train data..

time taken : 0:00:29.108904

Train Data

RMSE : 0.3247148668107051

MAPE : 8.219604179612201

adding train results in the dictionary..

Evaluating for test data...

time taken : 0:00:00.071158

Test Data

RMSE : 1.0727355769636053

MAPE : 34.857128869652

storing the test results in test dictionary...

Total time taken to run this algorithm : 0:00:33.206507

Double-click (or enter) to edit

4.4.5 XGBoost with initial 13 features + Surprise Baseline predictor + KNNBaseline

- ◦ ▪ First we will run XGBoost with predictions from both KNN's (that uses User_User and previous features.
- ◦ ▪ Then we will run XGBoost with just predictions from both knn models and predictions

Preparing Train data

```
# add the predicted values from both knns to this dataframe
reg_train['knn_bsl_u'] = models_evaluation_train['knn_bsl_u']['predictions']
reg_train['knn_bsl_m'] = models_evaluation_train['knn_bsl_m']['predictions']
```

```
reg_train.head(2)
```

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5
0	174683	10.0	3.587581	5.0	5.0	3.0	4.0	4.0	3.0	5.0	4.0	3.0	2.0
1	233949	10.0	3.587581	4.0	4.0	5.0	1.0	3.0	2.0	3.0	2.0	3.0	3.0

Preparing Test data

```
reg_test_df['knn_bsl_u'] = models_evaluation_test['knn_bsl_u']['predictions']
reg_test_df['knn_bsl_m'] = models_evaluation_test['knn_bsl_m']['predictions']
```

```
reg_test_df.head(2)
```

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1
0	808635	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679
1	941866	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679

```
import warnings
warnings.filterwarnings('ignore')

parameters = {'max_depth':[1,2,3],
              'learning_rate':[0.001,0.01,0.1],
              'n_estimators':[100,300,500,700]}

# prepare Train data
x_train = reg_train.drop(['user','movie','rating'], axis=1)
y_train = reg_train['rating']
# Prepare Test data
x_test = reg_test_df.drop(['user','movie','rating'], axis=1)
y_test = reg_test_df['rating']

start = datetime.now()
# initialize Our first XGBoost model...
xgb = xgb.XGBRegressor(nthread=-1,objective ='reg:squarederror')

# Perform cross validation
gscv = GridSearchCV(xgb,
                     param_grid = parameters,
                     scoring="neg_mean_squared_error",
                     cv = TimeSeriesSplit(n_splits=2),
                     n_jobs=-1,
                     verbose = 1)
gscv_result = gscv.fit(x_train, y_train)
```

```

# Summarize results
print("Best: %f using %s" % (gscv_result.best_score_, gscv_result.best_params_))
means = gscv_result.cv_results_['mean_test_score']
stds = gscv_result.cv_results_['std_test_score']
params = gscv_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))

print("\nTime Taken: ", start - datetime.now())

```



Fitting 2 folds for each of 36 candidates, totalling 72 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.

[Parallel(n_jobs=-1)]: Done 42 tasks | elapsed: 10.9min

[Parallel(n_jobs=-1)]: Done 72 out of 72 | elapsed: 21.1min finished

Best: -0.779502 using {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 300}

-9.013130 (0.126672) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 100}

-6.421049 (0.123222) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 300}

-4.673640 (0.113159) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 500}

-3.491194 (0.099686) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 700}

-8.994388 (0.122451) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 100}

-6.371245 (0.110112) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 300}

-4.603119 (0.097108) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 500}

-3.410694 (0.085758) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 700}

-8.976351 (0.118237) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 100}

-6.334924 (0.100342) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 300}

-4.558995 (0.090792) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 500}

-3.360381 (0.080070) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 700}

-2.377304 (0.078300) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 100}

-0.953497 (0.023238) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 300}

-0.864898 (0.015496) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 500}

-0.831809 (0.013428) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 700}

-2.289245 (0.070158) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 100}

-0.871308 (0.021769) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 300}

-0.806107 (0.013422) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 500}

-0.791131 (0.011465) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 700}

-2.236558 (0.066640) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 100}

-0.839010 (0.018284) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 300}

-0.789776 (0.012069) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 500}

-0.782775 (0.011176) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 700}

-0.804835 (0.011545) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 100}

-0.783283 (0.009779) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 300}

-0.782950 (0.009880) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 500}

-0.783037 (0.009997) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 700}

-0.784743 (0.010360) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 100}

-0.780881 (0.011277) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 300}

-0.780346 (0.011986) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 500}

-0.780116 (0.012194) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 700}

-0.781487 (0.011479) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100}

-0.779502 (0.013090) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 300}

-0.779511 (0.014345) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 500}

-0.779587 (0.015472) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 700}

Time Taken: -1 day, 23:38:17.702620

```

import xgboost as xgb
xgb_knn_hc1 = xgb.XGBRegressor(max_depth=3, learning_rate=0.1, n_estimators=300, nthread=-1, objective='multi:softmax')
https://colab.research.google.com/drive/1SiovwbMeSmqZFuzo0FrzmicRh93fhPMh#scrollTo=1kmagNKcxrjf&printMode=true
91/105

```

— xgb_knn_bsl — xgb.XGBRegressor(colsample_bytree=0.5, learning_rate=0.1, n_estimators=300, random_state=1, subsample=1, verbosity=1)

XGBRegressor(base_score=0.5, booster='gbtree', colsample_bytree=1, colsample_bynode=1, colsample_bylevel=1, gamma=0, importance_type='gain', learning_rate=0.1, max_delta_step=0, max_depth=3, min_child_weight=1, missing=None, n_estimators=300, n_jobs=1, nthread=-1, objective='reg:squarederror', random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None, silent=None, subsample=1, verbosity=1)

```
train_results, test_results = run_xgboost(xgb_knn_bsl, x_train, y_train, x_test, y_test)
```

```
# store the results in models_evaluations dictionaries
models_evaluation_train['xgb_knn_bsl'] = train_results
models_evaluation_test['xgb_knn_bsl'] = test_results
```

```
xgb.plot_importance(xgb_knn_bsl)
plt.show()
```

Training the model..

Done. Time taken : 0:00:38.850487

Done

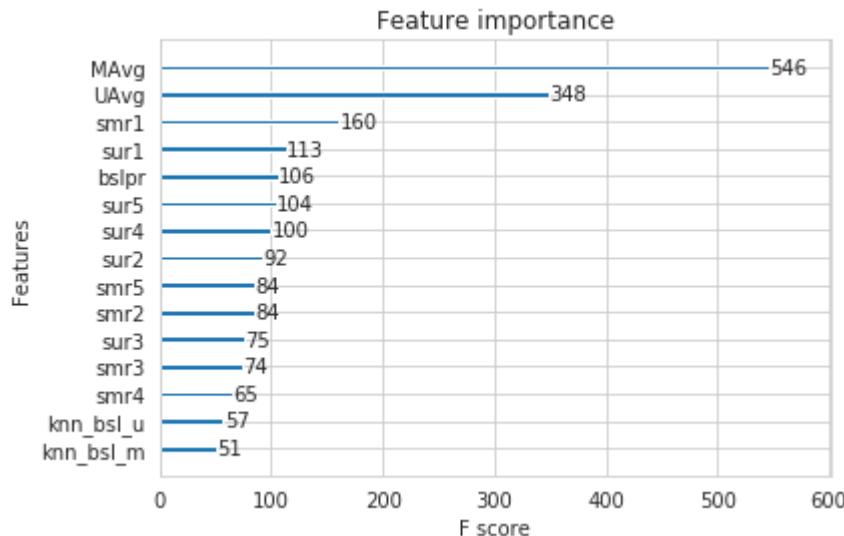
Evaluating the model with TRAIN data...

Evaluating Test data

TEST DATA

RMSE : 1.073461895590836

MAPE : 34.84759701095422



4.4.6 Matrix Factorization Techniques

4.4.6.1 SVD Matrix Factorization User Movie interactions

```
from surprise import SVD
```

http://surprise.readthedocs.io/en/stable/matrix_factorization.html#surprise.prediction_algorithms.m

- __ Predicted Rating : __
 - - $\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u$
 - q_i - Representation of item(movie) in latent factor space
 - p_u - Representation of user in new latent factor space
- A BASIC MATRIX FACTORIZATION MODEL in <https://datajobs.com/data-science-repo/Recomm>
- **Optimization problem with user item interactions and regularization (to avoid overfitting)**
 - - $$\sum_{r_{ui} \in R_{train}} (r_{ui} - \hat{r}_{ui})^2 + \lambda (b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2)$$

```
# initialize the model
svd = SVD(n_factors=100, biased=True, random_state=15, verbose=True)
svd_train_results, svd_test_results = run_surprise(svd, trainset, testset, verbose=True)

# Just store these error metrics in our models_evaluation datastructure
models_evaluation_train['svd'] = svd_train_results
models_evaluation_test['svd'] = svd_test_results
```



```
Training the model...
Processing epoch 0
Processing epoch 1
Processing epoch 2
Processing epoch 3
Processing epoch 4
Processing epoch 5
Processing epoch 6
Processing epoch 7
Processing epoch 8
Processing epoch 9
Processing epoch 10
Processing epoch 11
Processing epoch 12
Processing epoch 13
Processing epoch 14
Processing epoch 15
Processing epoch 16
Processing epoch 17
Processing epoch 18
Processing epoch 19
Done. time taken : 0:00:22.628173
```

```
Evaluating the model with train data..
time taken : 0:00:04.332222
```

```
-----
Train Data
```

```
-----
RMSE : 0.5520132958187114
```

```
MAPE : 15.79506473019488
```

```
adding train results in the dictionary..
```

```
Evaluating for test data...
time taken : 0:00:00.068839
```

```
-----
Test Data
```

```
-----
RMSE : 1.0725814708274273
```

```
MAPE : 34.83365884620077
```

```
storing the test results in test dictionary..
```

```
-----
Total time taken to run this algorithm : 0:00:27.032014
```

Double-click (or enter) to edit

4.4.6.2 SVD Matrix Factorization with implicit feedback from user (user rated movies)

```
from surprise import SVDpp
```

- ----> 2.5 Implicit Feedback in <http://courses.ischool.berkeley.edu/i290-dm/s11/SECURE/a1-kore.html>
- __ Predicted Rating : __
 - - $\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(p_u + |I_u|^{-\frac{1}{2}} \sum_{j \in I_u} y_j \right)$
- I_u -- the set of all items rated by user u
- y_j -- Our new set of item factors that capture implicit ratings.
- **Optimization problem with user item interactions and regularization (to avoid overfitting)**
 - - $\sum_{r_{ui} \in R_{train}} (r_{ui} - \hat{r}_{ui})^2 + \lambda (b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2 + \|y_j\|^2)$

```
# initialize the model
svdpp = SVDpp(n_factors=50, random_state=15, verbose=True)
svdpp_train_results, svdpp_test_results = run_surprise(svdpp, trainset, testset, verbose=True

# Just store these error metrics in our models_evaluation datastructure
models_evaluation_train['svdpp'] = svdpp_train_results
models_evaluation_test['svdpp'] = svdpp_test_results
```



```
Training the model...
```

```
processing epoch 0
processing epoch 1
processing epoch 2
processing epoch 3
processing epoch 4
processing epoch 5
processing epoch 6
processing epoch 7
processing epoch 8
processing epoch 9
processing epoch 10
processing epoch 11
processing epoch 12
processing epoch 13
processing epoch 14
processing epoch 15
processing epoch 16
processing epoch 17
processing epoch 18
processing epoch 19
```

```
Done. time taken : 0:06:35.033840
```

```
Evaluating the model with train data..
```

```
time taken : 0:00:25.340377
```

```
-----
```

```
Train Data
```

```
-----
```

```
RMSE : 0.5280176063074864
```

```
MAPE : 14.748297258604753
```

```
adding train results in the dictionary..
```

```
Evaluating for test data...
```

```
time taken : 0:00:00.315489
```

```
-----
```

```
Test Data
```

```
-----
```

```
RMSE : 1.0733978733157112
```

```
MAPE : 34.85667130606279
```

```
storing the test results in test dictionary...
```

```
-----  
Total time taken to run this algorithm : 0:07:00.691985
```

Double-click (or enter) to edit

Double-click (or enter) to edit

4.4.7 XgBoost with 13 features + Surprise Baseline + Surprise KNNbaseline + MF

Preparing Train data

```
# add the predicted values from both knns to this dataframe
reg_train['svd'] = models_evaluation_train['svd']['predictions']
reg_train['svdpp'] = models_evaluation_train['svdpp']['predictions']

reg_train.head(2)
```

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5
0	174683	10.0	3.587581	5.0	5.0	3.0	4.0	4.0	3.0	5.0	4.0	3.0	2.0
1	233949	10.0	3.587581	4.0	4.0	5.0	1.0	3.0	2.0	3.0	2.0	3.0	3.0

__Preparing Test data __

```
reg_test_df['svd'] = models_evaluation_test['svd']['predictions']
reg_test_df['svdpp'] = models_evaluation_test['svdpp']['predictions']
```

```
reg_test_df.head(2)
```

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1
0	808635	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679
1	941866	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679

Double-click (or enter) to edit

```
import warnings
warnings.filterwarnings('ignore')

parameters = {'max_depth':[1,2,3],
              'learning_rate':[0.001,0.01,0.1],
              'n_estimators':[100,300,500,700]}

# prepare Train data
x_train = reg_train.drop(['user','movie','rating'], axis=1)
y_train = reg_train['rating']
# Prepare Test data
x_test = reg_test_df.drop(['user','movie','rating'], axis=1)
y_test = reg_test_df['rating']

start = datetime.now()
```

```
# initialize Our first XGBoost model...
xgb = xgb.XGBRegressor(nthread=-1,objective ='reg:squarederror')

# Perform cross validation
gscv = GridSearchCV(xgb,
                     param_grid = parameters,
                     scoring="neg_mean_squared_error",
                     cv = TimeSeriesSplit(n_splits=2),
                     n_jobs=-1,
                     verbose = 1)
gscv_result = gscv.fit(x_train, y_train)

# Summarize results
print("Best: %f using %s" % (gscv_result.best_score_, gscv_result.best_params_))
means = gscv_result.cv_results_['mean_test_score']
stds = gscv_result.cv_results_['std_test_score']
params = gscv_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))

print("\nTime Taken: ",start - datetime.now())
```



```
Fitting 2 folds for each of 36 candidates, totalling 72 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed: 13.0min
[Parallel(n_jobs=-1)]: Done  72 out of  72 | elapsed: 25.0min finished
Best: -0.779843 using {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 500}
-9.013130 (0.126672) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 100}
-6.421049 (0.123222) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 300}
-4.673640 (0.113159) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 500}
-3.491194 (0.099686) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 700}
-8.994388 (0.122451) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 100}
-6.371245 (0.110112) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 300}
-4.603119 (0.097108) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 500}
-3.410694 (0.085758) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 700}
-8.976351 (0.118237) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 100}
-6.334924 (0.100342) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 300}
-4.558995 (0.090792) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 500}
-3.360381 (0.080070) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 700}
-2.377304 (0.078300) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 100}
-0.953497 (0.023238) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 300}
-0.864898 (0.015496) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 500}
-0.831809 (0.013428) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 700}
-2.289245 (0.070158) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 100}
-0.871308 (0.021769) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 300}
-0.806107 (0.013422) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 500}
-0.791131 (0.011465) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 700}
-2.236558 (0.066640) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 100}
-0.839010 (0.018284) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 300}
-0.789746 (0.012039) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 500}
-0.782758 (0.011158) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 700}
-0.804835 (0.011545) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 100}
-0.783295 (0.009791) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 300}
-0.782963 (0.009883) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 500}
-0.783024 (0.009962) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 700}
-0.784743 (0.010360) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 100}
-0.781157 (0.011271) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 300}
-0.780657 (0.011925) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 500}
-0.780602 (0.012291) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 700}
-0.781549 (0.011556) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100}
-0.779872 (0.013312) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 300}
-0.779843 (0.014375) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 500}
-0.780174 (0.015329) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 700}
```

Time Taken: -1 day, 23:33:46.860976

```
import xgboost as xgb
xgb_final = xgb.XGBRegressor(max_depth=3,learning_rate = 0.1,n_estimators=500,nthread=-1,obje
xgb_final
```



```

XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bynode=1, gamma=0
train_results, test_results = run_xgboost(xgb_final, x_train, y_train, x_test, y_test)

# store the results in models_evaluations dictionaries
models_evaluation_train['xgb_final'] = train_results
models_evaluation_test['xgb_final'] = test_results

xgb.plot_importance(xgb_final)
plt.show()

```

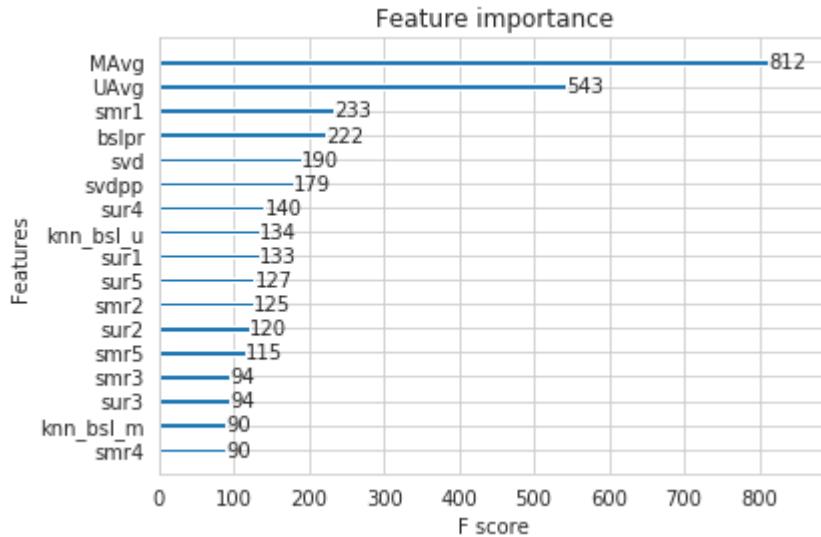
Training the model..
Done. Time taken : 0:01:19.234193

Done

Evaluating the model with TRAIN data...
Evaluating Test data

TEST DATA

RMSE : 1.0750707150728185
MAPE : 34.60135684971853



4.4.8 XgBoost with Surprise Baseline + Surprise KNNbaseline + MF Techniques

```

import warnings
warnings.filterwarnings('ignore')

parameters = {'max_depth':[1,2,3],
              'learning_rate':[0.001,0.01,0.1],
              'n_estimators':[100,300,500,700]}

# prepare train data
x_train = reg_train[['knn_bsl_u', 'knn_bsl_m', 'svd', 'svdpp']]
y_train = reg_train['rating']

```

```
# test data
x_test = reg_test_df[['knn_bsl_u', 'knn_bsl_m', 'svd', 'svdpp']]
y_test = reg_test_df['rating']

start = datetime.now()
# initialize Our first XGBoost model...
xgb = xgb.XGBRegressor(nthread=-1,objective ='reg:squarederror')

# Perform cross validation
gscv = GridSearchCV(xgb,
                     param_grid = parameters,
                     scoring="neg_mean_squared_error",
                     cv = TimeSeriesSplit(n_splits=2),
                     n_jobs=-1,
                     verbose = 1)
gscv_result = gscv.fit(x_train, y_train)

# Summarize results
print("Best: %f using %s" % (gscv_result.best_score_, gscv_result.best_params_))
means = gscv_result.cv_results_['mean_test_score']
stds = gscv_result.cv_results_['std_test_score']
params = gscv_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))

print("\nTime Taken: ",start - datetime.now())
```



Fitting 2 folds for each of 36 candidates, totalling 72 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.

[Parallel(n_jobs=-1)]: Done 42 tasks | elapsed: 7.4min

[Parallel(n_jobs=-1)]: Done 72 out of 72 | elapsed: 14.1min finished

Best: -1.202652 using {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 100}

-9.036935 (0.123927) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 100}

-6.484284 (0.118114) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 300}

-4.767695 (0.108795) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 500}

-3.612604 (0.098130) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 700}

-9.036812 (0.124136) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 100}

-6.484259 (0.118682) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 300}

-4.767739 (0.109466) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 500}

-3.612643 (0.098902) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 700}

-9.036828 (0.124153) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 100}

-6.484156 (0.118835) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 300}

-4.767341 (0.109824) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 500}

-3.612286 (0.099180) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 700}

-2.535158 (0.081989) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 100}

-1.236296 (0.025870) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 300}

-1.204513 (0.016477) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 500}

-1.202826 (0.015180) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 700}

-2.535071 (0.082579) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 100}

-1.236346 (0.025945) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 300}

-1.204649 (0.016525) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 500}

-1.203012 (0.015259) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 700}

-2.534803 (0.082767) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 100}

-1.236426 (0.026042) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 300}

-1.204807 (0.016639) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 500}

-1.203219 (0.015383) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 700}

-1.202652 (0.014986) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 100}

-1.202803 (0.015018) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 300}

-1.202898 (0.015032) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 500}

-1.202968 (0.015045) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 700}

-1.202934 (0.015095) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 100}

-1.203698 (0.015187) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 300}

-1.204436 (0.015432) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 500}

-1.205158 (0.015674) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 700}

-1.203386 (0.015224) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100}

-1.204793 (0.015507) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 300}

-1.206130 (0.015939) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 500}

-1.207390 (0.016250) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 700}

Time Taken: -1 day, 23:45:48.429385

```
import xgboost as xgb
xgb_all_models = xgb.XGBRegressor(max_depth=1, learning_rate = 0.1, n_estimators=100, nthread=-1
xgb_all_models
```



```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bynode=1, gamma=0
train_results, test_results = run_xgboost(xgb_all_models, x_train, y_train, x_test, y_test)

# store the results in models_evaluations dictionaries
models_evaluation_train['xgb_all_models'] = train_results
models_evaluation_test['xgb_all_models'] = test_results

xgb.plot_importance(xgb_all_models)
plt.show()
```

👤 Training the model..

Done. Time taken : 0:00:04.109145

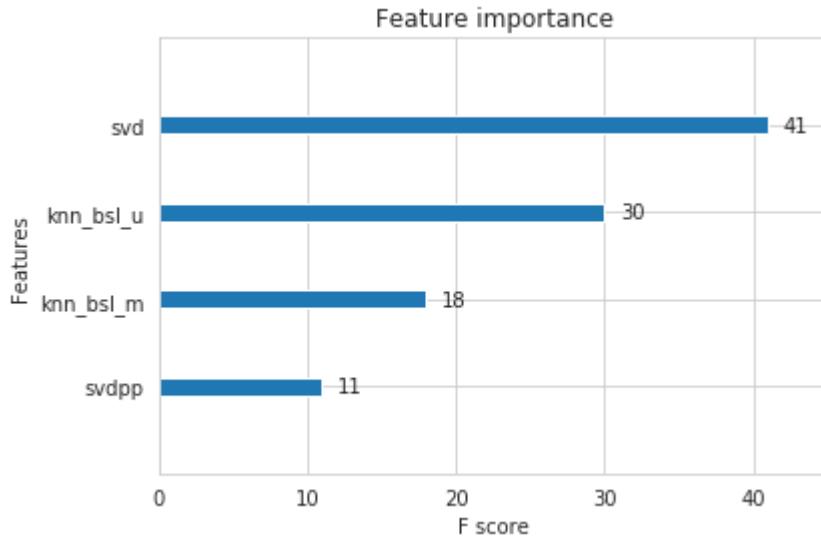
Done

Evaluating the model with TRAIN data...

Evaluating Test data

TEST DATA

RMSE : 1.0751698628570652
MAPE : 35.108292617658705



4.5 Comparision between all models

```
# Saving our TEST_RESULTS into a dataframe so that you don't have to run it again
pd.DataFrame(models_evaluation_test).to_csv('small_sample_results.csv')
models = pd.read_csv('small_sample_results.csv', index_col=0)
models.loc['rmse'].sort_values()
```

👤

```

svd           1.0725814708274273
knn_bsl_m    1.0727355769636053
knn_bsl_u    1.072783288145299
xgb_bsl      1.0731537024868656
bsl_algo     1.073153983662934
svdpp        1.0733978733157112
xgb_knn_bsl 1.073461895590836
first_algo   1.073628401472761
xgb_final    1.0750707150728185
xgb_all_models 1.0751698628570652
Name: rmse, dtype: object

```

▼ Results(PrettyTable):

```

from prettytable import PrettyTable
#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable
x = PrettyTable()
x.field_names = [ "Model", "RMSE"]
x.add_row(["svd", 1.0725])
x.add_row(["knn_bsl_m", 1.0727])
x.add_row(["knn_bsl_u", 1.0727])
x.add_row(["xgb_bsl", 1.0731])
x.add_row(["bsl_algo", 1.0731])
x.add_row(["svdpp", 1.0733])
x.add_row(["xgb_knn_bsl", 1.0734])
x.add_row(["first_algo", 1.0736])
x.add_row(["xgb_final", 1.0750])
x.add_row(["xgb_all_models", 1.0751])
print(x)

```

👤

Model	RMSE
svd	1.0725
knn_bsl_m	1.0727
knn_bsl_u	1.0727
xgb_bsl	1.0731
bsl_algo	1.0731
svdpp	1.0733
xgb_knn_bsl	1.0734
first_algo	1.0736
xgb_final	1.075
xgb_all_models	1.0751

▼ Conclusions:

1. First i constructed reg_train.csv and reg_test.csv with (25k,3k) , (13k,1.5k) respectively
2. Then i performed Then i performed XGboost with 13 features
3. Then on XGBoost with initial 13 features + Surprise Baseline predictor.
4. Then on XGBoost with initial 13 features + Surprise Baseline predictor + KNNBaseline predictor.
5. Also XGBoost with initial 13 features + Surprise Baseline predictor + KNNBaseline predictor + SVD.
6. Also XGBoost with initial 13 features , SVD ,SVD++, Surprise Baseline predictor + KNNBaseline predictor.
7. Got the best score for SVD model