1. Business Problem

1.1 Problem Description

Netflix is all about connecting people to the movies they love. To help customers find those movies, they developed world-class movie recommendation system: CinematchSM. Its job is to predict whether someone will enjoy a movie based on how much they liked or disliked other movies. Netflix use those predictions to make personal movie recommendations based on each customer's unique tastes. And while **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 are described in the literature, some aren't. We're curious whether any of these can beat Cinematch by making better predictions. Because, frankly, if there is a much better approach it 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 what Cinematch can do on the same training data set. (Accuracy is a measurement of how closely predicted ratings of movies match subsequent actual 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-part-1-55838468f429 (very nice blog)
- 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 our work was inspired by 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 ahs 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, combined_data_4.txt] contains the movie id followed by a colon. Eac
h subsequent line in the file corresponds to a rating from a customer and its date in th
e following format:

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 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.

In [1]:

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

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

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

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

3. Exploratory Data Analysis

3.1 Preprocessing

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

In [2]:

```
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 are reading from each of the four files and appendig each rating to a global file 'train.
csv'
    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)
```

```
Time taken : 0:00:00.000488
```

In [3]:

```
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..
```

In [4]:

```
df.head()
```

Out[4]:

	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

In [5]:

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

Out[5]:

```
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

In [6]:

```
# 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
```

3.1.3 Removing Duplicates

In [7]:

```
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..
```

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

In [8]:

```
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 Spliting data into Train and Test(80:20)

In [2]:

```
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)

In [3]:

```
# 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)

In [4]:

```
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

In [12]:

```
# 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

In [13]:

```
fig, ax = plt.subplots()
plt.title('Distribution of ratings over Training dataset', fontsize=15)
sns.countplot(train_df.rating)
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.

In [14]:

```
# 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()
```

Out[14]:

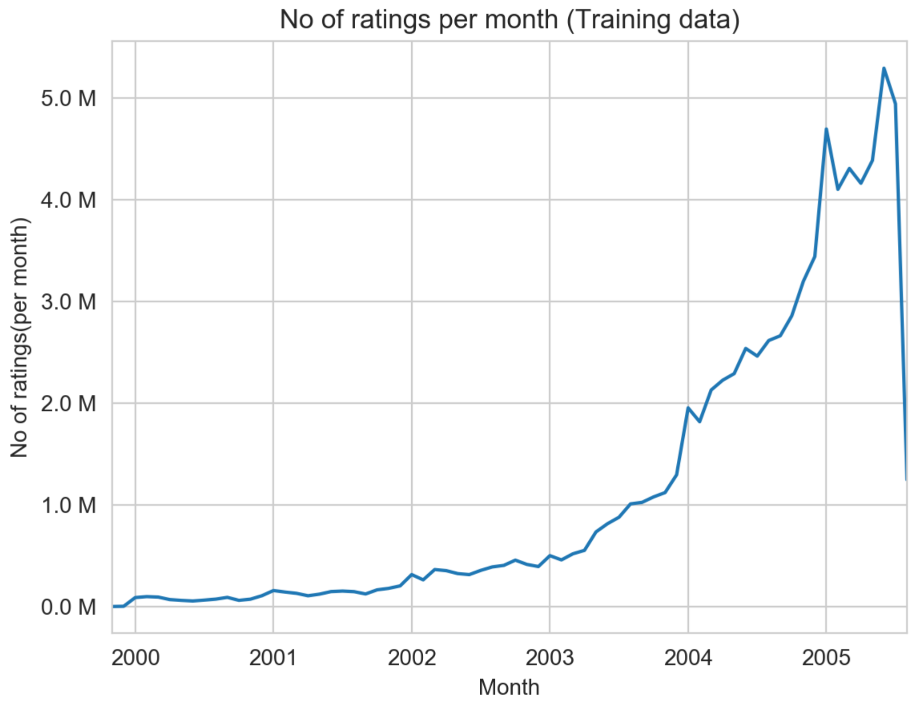
	movie	user	rating	date	day_of_week
80384400	12074	2033618	4	2005-08-08	Monday
80384401	862	1797061	3	2005-08-08	Monday

	movie	user	rating	date	day_of_week
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

In [15]:

```
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()
```



3.3.3 Analysis on the Ratings given by user

In [16]:

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

no_of_rated_movies_per_user.head()
```

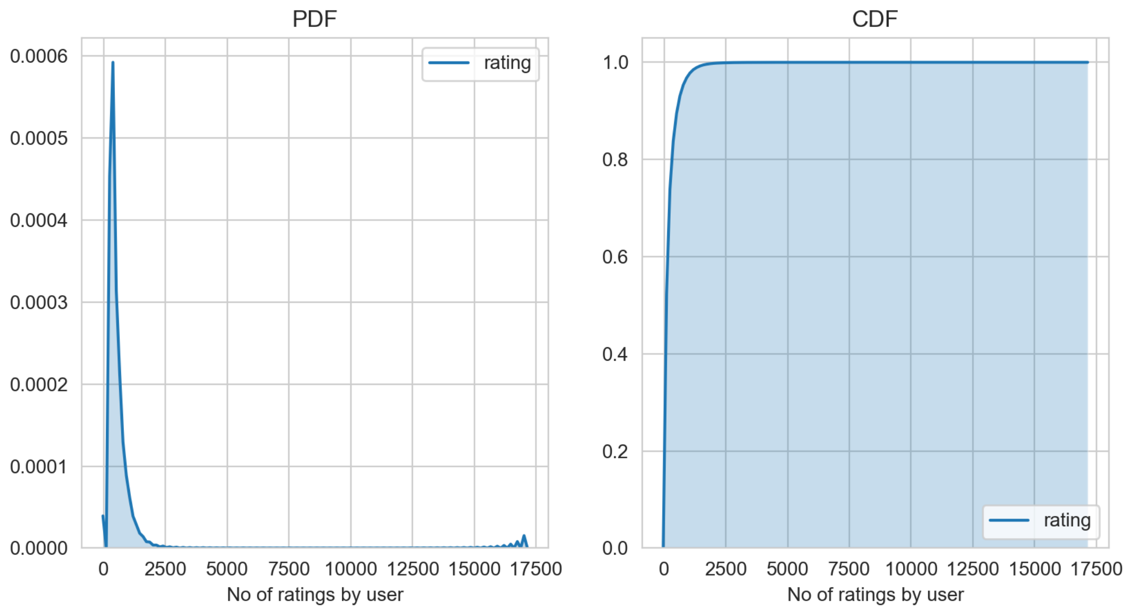
Out[16]:

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

In [17]:

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

```python
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()
```



In [18]:

```python
no_of_rated_movies_per_user.describe()
```

Out[18]:

```
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..*

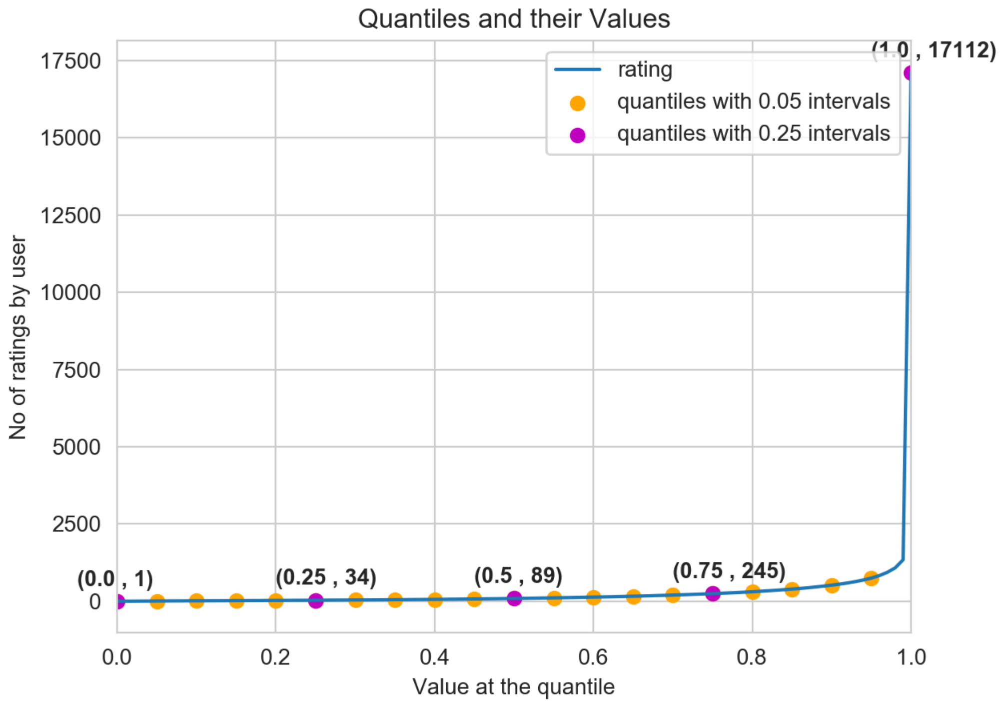
In [19]:

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

In [20]:

```python
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")
```

```python
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()
```



In [21]:

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

Out[21]:

```
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??

In [22]:

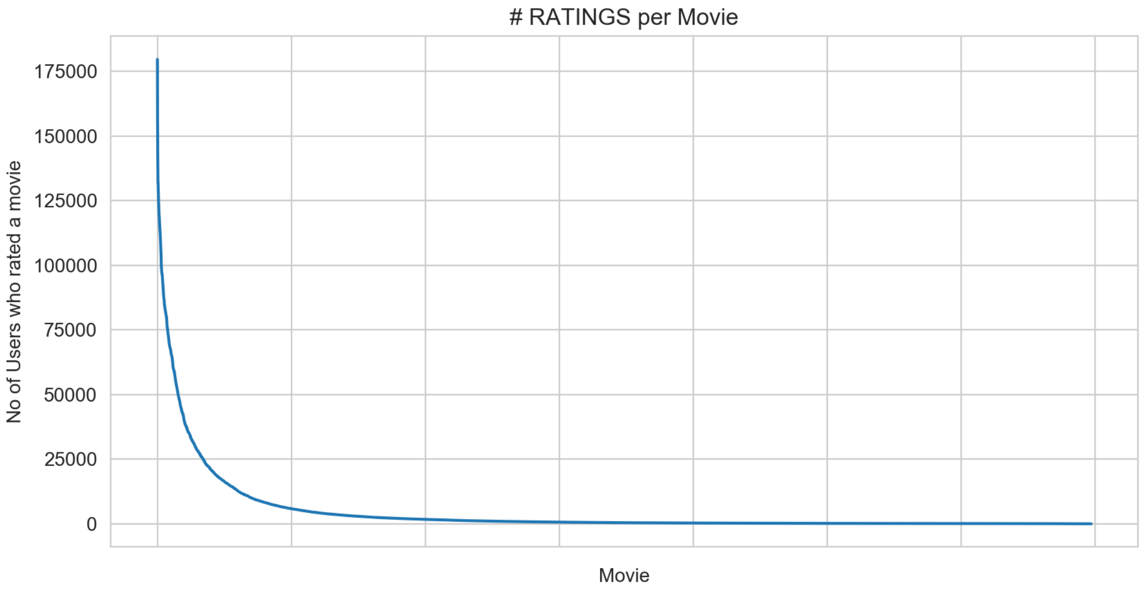
```
print('\n No of ratings at last 5 percentile : {}\n'.format(sum(no_of_rated_movies_per_user>= 749)
) )
```

```
 No of ratings at last 5 percentile : 20305
```

3.3.4 Analysis of ratings of a movie given by a user

In [23]:

```
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()
```



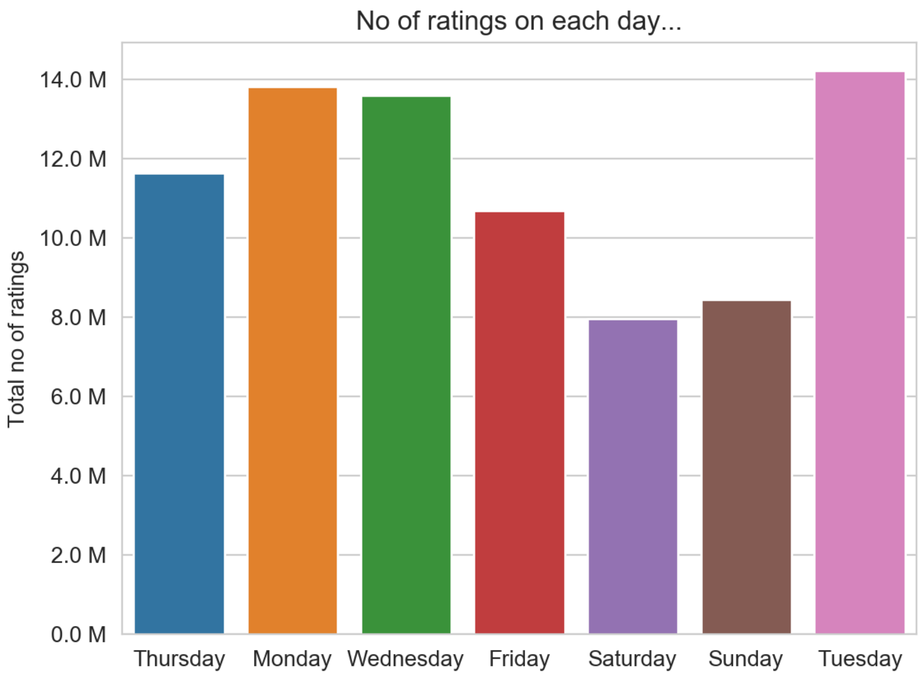
- **It is very skewed.. just like nunmber of ratings given per user.**

```
- 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 hundereds of ratings.
```

3.3.5 Number of ratings on each day of the week

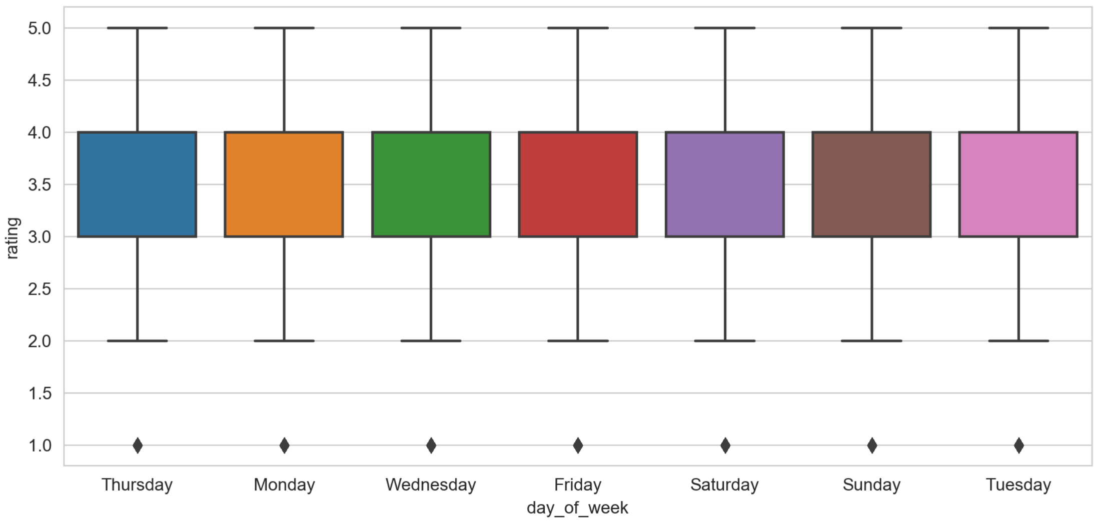
In [24]:

```
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()
```



In [25]:

```
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)
```




```
0:00:12.328294
```

In [103]:

```
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
```

3.3.6 Creating sparse matrix from data frame

3.3.6.1 Creating sparse matrix from train data frame

In [5]:

```
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..
0:00:03.467024
```

The Sparsity of Train Sparse Matrix

In [6]:

```
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

In [7]:

```
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..
0:00:00.826571
```

The Sparsity of Test data Matrix

In [8]:

```
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

In [2]:

```
# 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
    # creae a dictonary of users and their average ratigns..
    average_ratings = { i : sum_of_ratings[i]/no_of_ratings[i]
                                 for i in range(u if of_users else m)
                                    if no_of_ratings[i] !=0}

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

3.3.7.1 finding global average of all movie ratings

In [32]:

```
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
```

Out[32]:

```
{'global': 3.582890686321557}
```

3.3.7.2 finding average rating per user

In [33]:

```
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

In [34]:

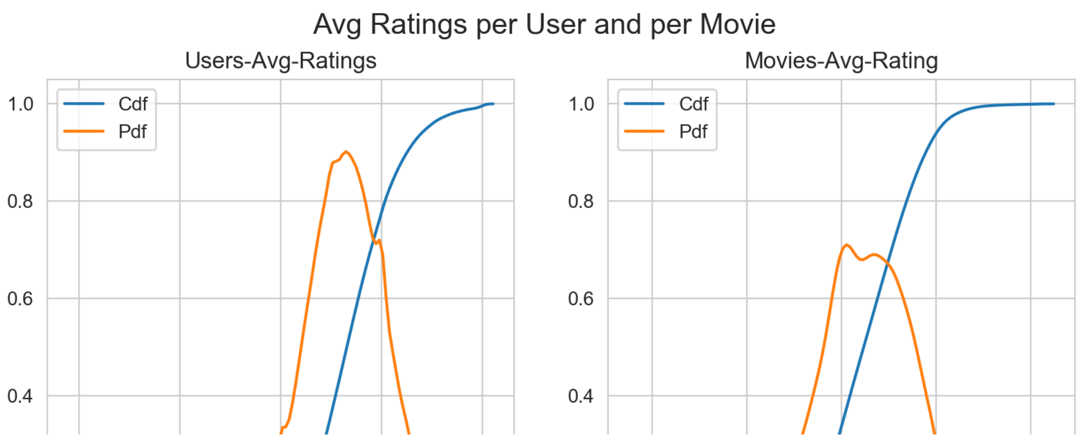
```
train_averages['movie'] =  get_average_ratings(train_sparse_matrix, of_users=False)
print('\n AVerage rating of movie 15 :',train_averages['movie'][15])
```

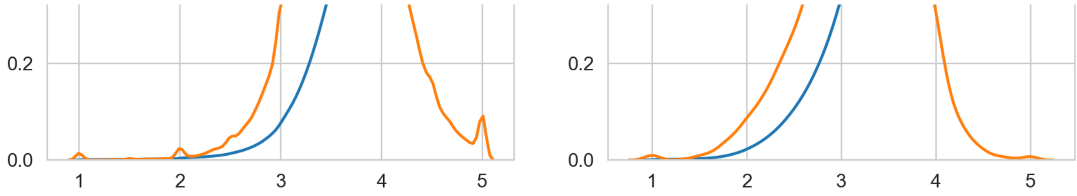
```
 AVerage rating of movie 15 : 3.3038461538461537
```

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

In [35]:

```
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)
```






```
0:00:42.208804
```

3.3.8 Cold Start problem

3.3.8.1 Cold Start problem with Users

In [36]:

```
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/total_users)*100, 2)))
```

```
Total number of Users  : 480189

Number of Users in Train data : 405041

No of Users that didn't appear in train data: 75148(15.65 %)
```

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

3.3.8.2 Cold Start problem with Movies

In [37]:

```
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/total_movies)*100, 2)))
```

```
Total number of Movies  : 17770

Number of Users in Train data : 17424

No of Movies that didn't appear in train data: 346(1.95 %)
```

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

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 and lots of time*) because of number of. usersbeing lare.

   - 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)

In [38]:

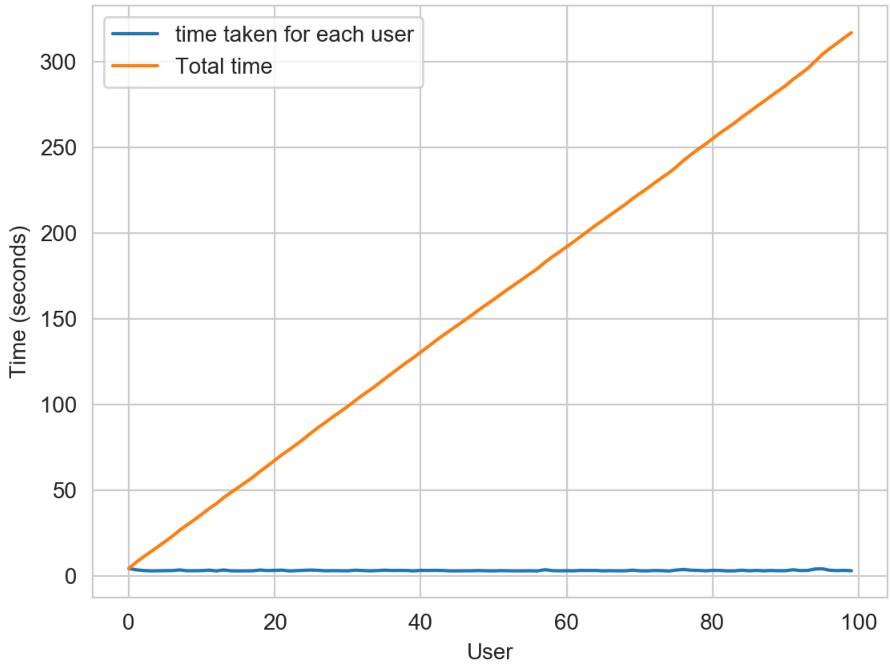
```
from sklearn.metrics.pairwise import cosine_similarity


def compute_user_similarity(sparse_matrix, compute_for_few=False, top = 100, verbose=False, verb_fo
r_n_rows = 20,
                            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_taken
```

In [39]:

```
start = datetime.now()
```

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

```
Computing top 100 similarities for each user..
computing done for 20 users [  time elapsed : 0:01:04.201483  ]
computing done for 40 users [  time elapsed : 0:02:07.396758  ]
computing done for 60 users [  time elapsed : 0:03:09.118349  ]
computing done for 80 users [  time elapsed : 0:04:12.017459  ]
computing done for 100 users [  time elapsed : 0:05:17.046071  ]
Creating Sparse matrix from the computed similarities
```




```
----------------------------------------------------------------------------------------------------
Time taken : 0:05:26.782410
```

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

- We have **405,041 users** in out training set and computing similarities between them..(**17K dimensional vector..**) is time consuming..
- From above plot, It took roughly **8.88 sec** for computing simlilar users for **one user**
- We have **405,041 users** with us in training set.
- $405041 \times 8.88 = 3596764.08 \text{sec} = 59946.068 \text{ min}$
  - Even if we run on 4 cores parallelly (a typical system now a days), It will still take almost **10 and 1/2** days.

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

In [40]:

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

start = datetime.now()

# initilaize the algorithm with some parameters..
# All of them are default except n_components. n_itr 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)
```

```
0:35:59.412853
```

Here,

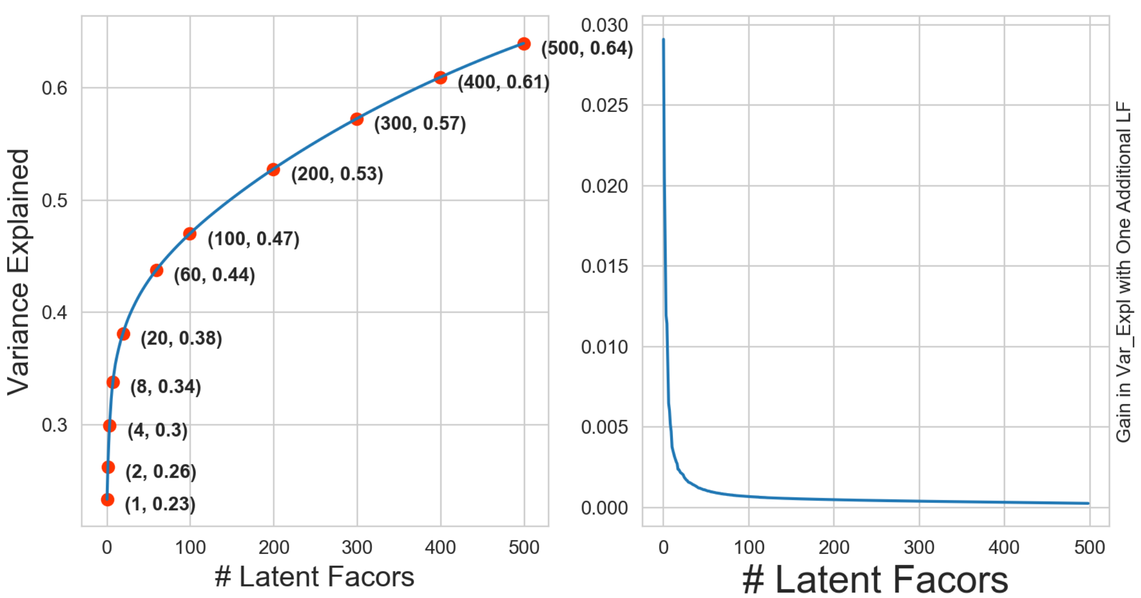
- \sum \longleftarrow (netflix_svd.**singular_values_**)
- \bigvee^T \longleftarrow (netflix_svd.**components_**)
- \bigcup is not returned. instead **Projection_of_X** onto the new vectorspace is returned.
- It uses **randomized svd** internally, which returns **All 3 of them saperately**. Use that instead..

In [41]:

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

In [42]:

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

ax1.set_ylabel("Variance Explained", fontsize=15)
ax1.set_xlabel("# Latent Facors", fontsize=15)
ax1.plot(expl_var)
# annote 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()
```



In [43]:

```
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 factore too it, the **_gain in expained variance** with that addition is decreasing. (Obviously, because they are sorted that way).
- ***LHS Graph***:
  - **x** --- (No of latent factos),
  - **y** --- (The variance explained by taking x latent factors)

- **More decrease in the line (RHS graph)** :
  - We are getting more expained variance than before.
- **Less decrease in that line (RHS graph)** :
  - We are not getting benifitted from adding latent factor furthur. This is what is shown in the plots.

- ***RHS Graph***:
  - **x** --- (No of latent factors),
  - **y** --- (Gain n Expl_Var by taking one additional latent factor)

In [44]:

```
# 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)
```

```
0:00:27.910038
```

In [45]:

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

Out[45]:

```
(numpy.ndarray, (2649430, 500))
```

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

In [46]:

```
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:
```

```
else:
    trunc_sparse_matrix = sparse.load_npz('trunc_sparse_matrix.npz')
```

In [47]:

```
trunc_sparse_matrix.shape
```

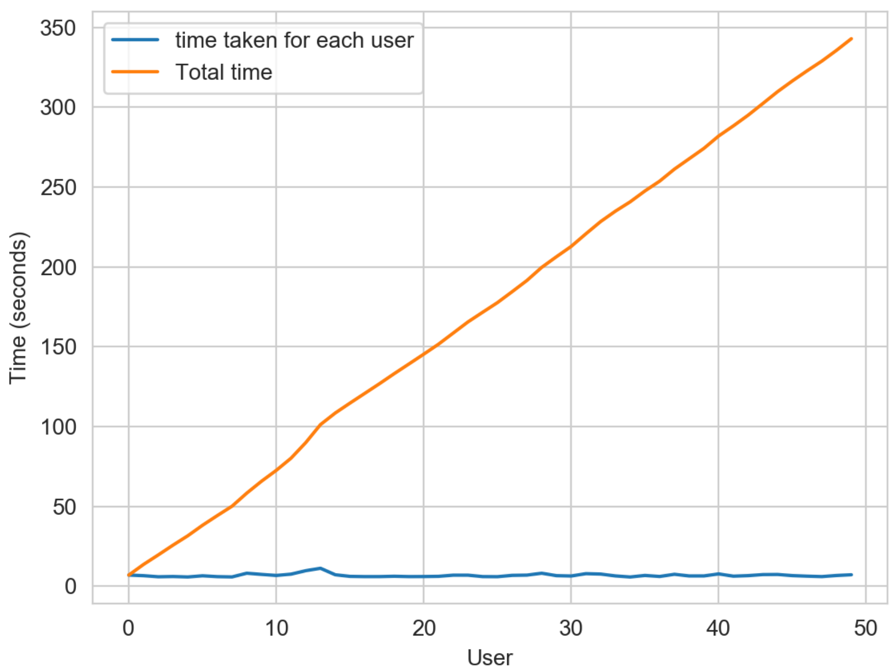
Out[47]:

```
(2649430, 500)
```

In [48]:

```
start = datetime.now()
trunc_u_u_sim_matrix, _ = compute_user_similarity(trunc_sparse_matrix, compute_for_few=True, top=50
, verbose=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 : 0:01:05.806740  ]
computing done for 20 users [  time elapsed : 0:02:19.255957  ]
computing done for 30 users [  time elapsed : 0:03:26.396183  ]
computing done for 40 users [  time elapsed : 0:04:34.244082  ]
computing done for 50 users [  time elapsed : 0:05:42.938093  ]
Creating Sparse matrix from the computed similarities
```




```
--------------------------------------------------
time: 0:06:12.348658
```

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

- from above plot, It took almost **12.18** for computing simlilar users for **one user**
- We have **405041 users** with us in training set.
- { 405041 \times 12.18 ==== 4933399.38 \sec } ==== 82223.323 \min ==== 1370.388716667 \text{ hours} ==== 57.099529861 \text{ days}...
    - Even we run on 4 cores parallelly (a typical system now a days), It will still take almost **(14 - 15)** days.

- **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, whenenver required (**ie., Run time**)

```
- We maintain a binary Vector for users, which tells us whether we already computed or
not..
- ***If not*** :
    - Compute top (let's just say, 1000) most similar users for this given user, and add
this to our datastructure, so that we can just access it(similar users) without recomputing
it again.
    -
- ***If It is already Computed***:
    - Just get it directly from our datastructure, which has that information.
    - In production time, We might have to recompute similarities, if it is computed a long
time ago. Because user preferences changes over time. If we could maintain some kind of
Timer, which when expires, we have to update it ( recompute it ).
    -
- ***Which datastructure to use:***
    - It is purely implementation dependant.
    - One simple method is to maintain a **Dictionary Of Dictionaries**.
        -
        - **key    :** _userid_
        - __value__: _Again a dictionary_
            - __key__  : _Similar User_
            - __value__: _Similarity Value_
```

3.4.2 Computing Movie-Movie Similarity matrix

In [49]:

```python
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
0:00:22.785549
```

In [50]:

```python
m_m_sim_sparse.shape
```

Out[50]:

```
(17771, 17771)
```

- Even though we have similarity measure of each movie, with all other movies, We generally don't care much about least similar movies.
- 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 saperate dictionary.

In [51]:

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

In [52]:

```
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]
```

```
0:00:28.634101
```

Out[52]:

```
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])
```

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....

In [53]:

```
# First Let's load the movie details into soe dataframe..
# movie details are in 'netflix/movie_titles.csv'
# removed data_folder/
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: 12.22 ms
Type conversion took: 12.59 ms
Parser memory cleanup took: 0.00 ms
```

Out[53]:

year_of_release title

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'

In [54]:

```
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 similarto this  and we will get only top most..".format(m_m_s
im_sparse[:,mv_id].getnnz()))
```

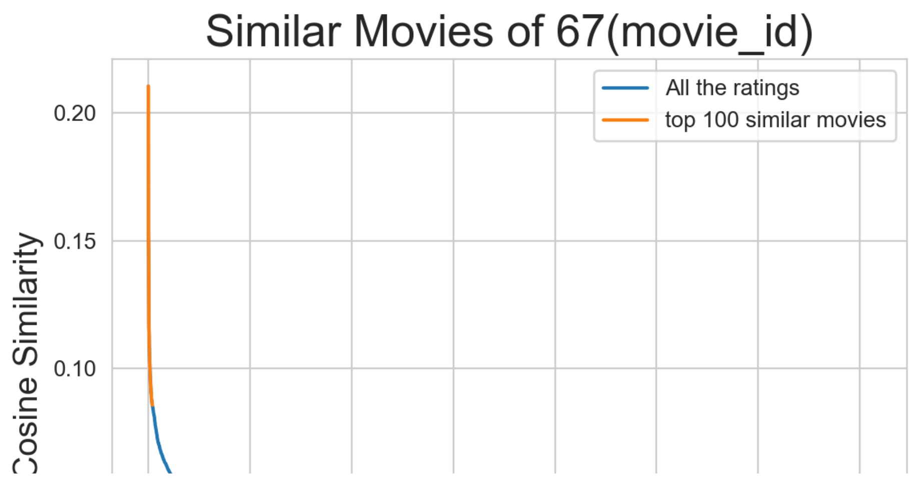
```
Movie -----> Vampire Journals

It has 270 Ratings from users.

We have 17284 movies which are similarto this  and we will get only top most..
```

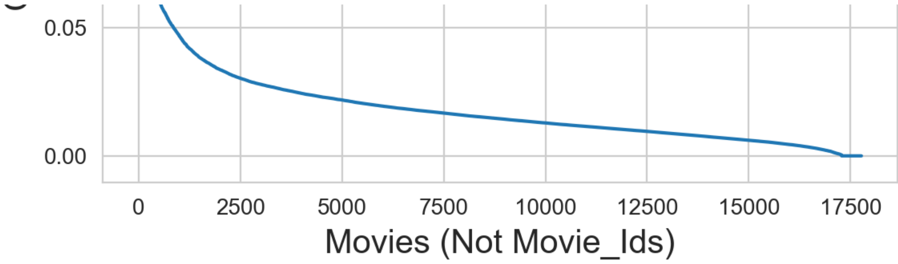
In [55]:

```
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 its
similarity (ie.,1)
                                               # and return its indices(movie_ids)
```

In [56]:

```
plt.plot(similarities[sim_indices], label='All the ratings')
plt.plot(similarities[sim_indices[:100]], label='top 100 similar movies')
plt.title("Similar Movies of {}(movie_id)".format(mv_id), fontsize=20)
plt.xlabel("Movies (Not Movie_Ids)", fontsize=15)
plt.ylabel("Cosine Similarity",fontsize=15)
plt.legend()
plt.show()
```





Top 10 similar movies

In [57]:

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

Out[57]:

movie_id	year_of_release	title
323	1999.0	Modern Vampires
4044	1998.0	Subspecies 4: Bloodstorm
1688	1993.0	To Sleep With a Vampire
13962	2001.0	Dracula: The Dark Prince
12053	1993.0	Dracula Rising
16279	2002.0	Vampires: Los Muertos
4667	1996.0	Vampirella
1900	1997.0	Club Vampire
13873	2001.0	The Breed
15867	2003.0	Dracula II: Ascension

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

4. Machine Learning Models

In [10]:

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

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

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

    # It just to make sure to get same sample everytime we run this program..
```

```
    # and pick without replacement....
    np.random.seed(15)
    sample_users = np.random.choice(users, no_users, replace=False)
    sample_movies = np.random.choice(movies, no_movies, replace=False)
    # get the boolean mask or these sampled_items in originl row/col_inds..
    mask = np.logical_and( np.isin(row_ind, sample_users),
                      np.isin(col_ind, sample_movies) )

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

    if verbose:
        print("Sampled Matrix : (users, movies) -- ({} {})".format(len(sample_users), len(sample_mo
vies)))
        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

In [59]:

```
start = datetime.now()
path = "sample/small/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, no_m
ovies=1000,
                                             path = path)

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

```
It is present in your pwd, getting it from disk....
DONE..
0:00:00.032148
```

4.1.2 Build sample test data from the test data

In [60]:

```
start = datetime.now()

path = "sample/small/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, no_movi
es=500,
                                                 path = "sample/small/sample_test_sparse_matrix.npz
)
print(datetime.now() - start)
```

```
It is present in your pwd, getting it from disk....
DONE..
0:00:00.024873
```

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

In [61]:

```
sample_train_averages = dict()
```

4.2.1 Finding Global Average of all movie ratings

In [62]:

```
# 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
```

Out[62]:

```
{'global': 3.581679377504138}
```

4.2.2 Finding Average rating per User

In [63]:

```
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

In [64]:

```
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
```

4.3 Featurizing data

In [65]:

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

```
 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

In [66]:

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

In [67]:

```
############################################################
# It took me almost 10 hours to prepare this train dataset.#
############################################################
start = datetime.now()
if os.path.isfile('sample/small/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('sample/small/reg_train.csv', mode='w') as reg_data_file:
        count = 0
        for (user, movie, rating)  in zip(sample_train_users, sample_train_movies,
sample_train_ratings):
            st = datetime.now()
        #     print(user, movie)
            #--------------------- Ratings of "movie" by similar users of "user" -----------------
--
            # compute the similar Users of the "user"
            user_sim = cosine_similarity(sample_train_sparse_matrix[user],
sample_train_sparse_matrix).ravel()
            top_sim_users = user_sim.argsort()[::-1][1:] # we are ignoring 'The User' from its simi
lar users.
            # get the ratings of most similar users for this movie
            top_ratings = sample_train_sparse_matrix[top_sim_users, movie].toarray().ravel()
            # we will make it's length "5" by adding movie averages to .
            top_sim_users_ratings = list(top_ratings[top_ratings != 0][:5])
            top_sim_users_ratings.extend([sample_train_averages['movie'][movie]]*(5 -
len(top_sim_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.T).ravel()
            top_sim_movies = movie_sim.argsort()[::-1][1:] # we are ignoring 'The User' from its si
milar users.
            # get the ratings of most similar movie rated by this user..
            top_ratings = sample_train_sparse_matrix[user, top_sim_movies].toarray().ravel()
            # we will make it's length "5" by adding user averages to.
            top_sim_movies_ratings = list(top_ratings[top_ratings != 0][:5])
            top_sim_movies_ratings.extend([sample_train_averages['user']
[user]]*(5-len(top_sim_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...
            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)
```

```
File already exists you don't have to prepare again...
0:00:00.000373
```

Reading from the file to make a Train_dataframe

In [68]:

```
reg_train = pd.read_csv('sample/small/reg_train.csv', names = ['user', 'movie', 'GAvg', 'sur1', 'su
r2', 'sur3', 'sur4', 'sur5','smr1', 'smr2', 'smr3', 'smr4', 'smr5', 'UAvg', 'MAvg', 'rating'],
header=None)
reg_train.head()
```

Out[68]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg	rating
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	3.370370	4.092437	4
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	3.555556	4.092437	3
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	3.714286	4.092437	5
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	3.584416	4.092437	5
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	3.750000	4.092437	5

- **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.

4.3.1.2 Featurizing test data

In [69]:

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

In [70]:

```
sample_train_averages['global']
```

Out[70]:

3.581679377504138

In [71]:

```
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_ratings):
            st = datetime.now()

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

            except (IndexError, KeyError):
                # It is a new User or new Movie or there are no ratings for given user for top simi
lar movies...
                ########## Cold STart Problem ##########
                top_sim_users_ratings.extend([sample_train_averages['global']]*(5 - 
len(top_sim_users_ratings)))
                #print(top_sim_users_ratings)
            except:
                print(user, movie)
                # we just want KeyErrors to be resolved. Not every Exception...
                raise



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

top_sim_movies_ratings.extend([sample_train_averages['global']]*(5-len(top_sim_movies_ratings)))
                #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)
```

```
It is already created...
```

Reading from the file to make a test dataframe

In [72]:

```
reg_test_df = pd.read_csv('sample/small/reg_test.csv', names = ['user', 'movie', 'GAvg', 'sur1', 'sur2', 'sur3', 'sur4', 'sur5',
                                                          'smr1', 'smr2', 'smr3', 'smr4', 'smr5',
                                                          'UAvg', 'MAvg', 'rating'], header=None)
reg_test_df.head(4)
```

Out[72]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	
0	808635	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.58
1	941866	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.58
2	1737912	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.58
3	1849204	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.58

- **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.

4.3.2 Transforming data for Surprise models

In [73]:

```
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 models like SVD, KNNBaseLineOnly....etc..,in Surprise.
- We can form the trainset from a file, or from a Pandas DataFrame. http://surprise.readthedocs.io/en/stable/getting_started.html#load-dom-dataframe-py

In [74]:

```
# 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 impotant)

In [75]:

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

Out[75]:

```
[(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)

In [76]:

```
models_evaluation_train = dict()
```

```
models_evaluation_test = dict()

models_evaluation_train, models_evaluation_test
```

Out[76]:

```
({}, {})
```

Utility functions for running regression models

In [15]:

```
# 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

In [16]:

```
# 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 ''predicted rat
ings''.
    '''
    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..')
    # 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 : {}\n".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['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 : {}\n".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
```

4.4.1 XGBoost with initial 13 features

In [79]:

```
import xgboost as xgb
```

In [80]:

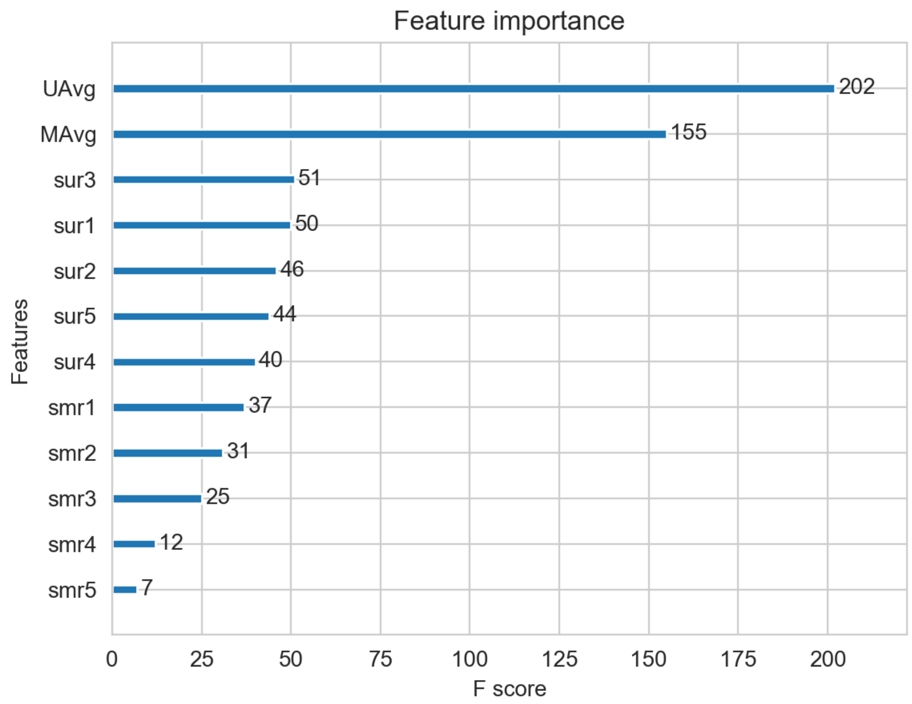
```
# prepare Train data
x_train = reg_train.drop(['user','movie','rating'], axis=1)
y_train = reg_train['rating']

# 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...
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..
[19:45:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

```
/Users/mayankgupta/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/Users/mayankgupta/anaconda3/lib/python3.7/site-packages/xgboost/core.py:588: FutureWarning:
Series.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \
```

```
Done. Time taken : 0:00:07.792542
```

```
Done

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

TEST DATA
------------------------------
RMSE :  1.076373581778953
MAPE :  34.48223172520999
```



4.4.2 Suprise BaselineModel

In [81]:

```python
from surprise import BaselineOnly
```

Predicted_rating : (baseline prediction)

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

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

- $\pmb \mu$: Average of all trainings in training data.
- $\pmb b_u$: User bias
- $\pmb b_i$: Item bias (movie biases)

Optimization function (Least Squares Problem)

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

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

In [82]:

```
# 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 : 0:00:00.566335

Evaluating the model with train data..
time taken : 0:00:00.914533
---------------
Train Data
---------------
RMSE : 0.9347153928678286

MAPE : 29.389572652358183

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.057080
---------------
Test Data
---------------
RMSE : 1.0730330260516174

MAPE : 35.04995544572911

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:00:01.538428
```

4.4.3 XGBoost with initial 13 features + Surprise Baseline predictor

Updating Train Data

In [83]:

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

Out[83]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg	rating	bslpr
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	3.370370	4.092437	4	3.898982
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	3.555556	4.092437	3	3.371403

Updating Test Data

In [84]:

```
# 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)
```

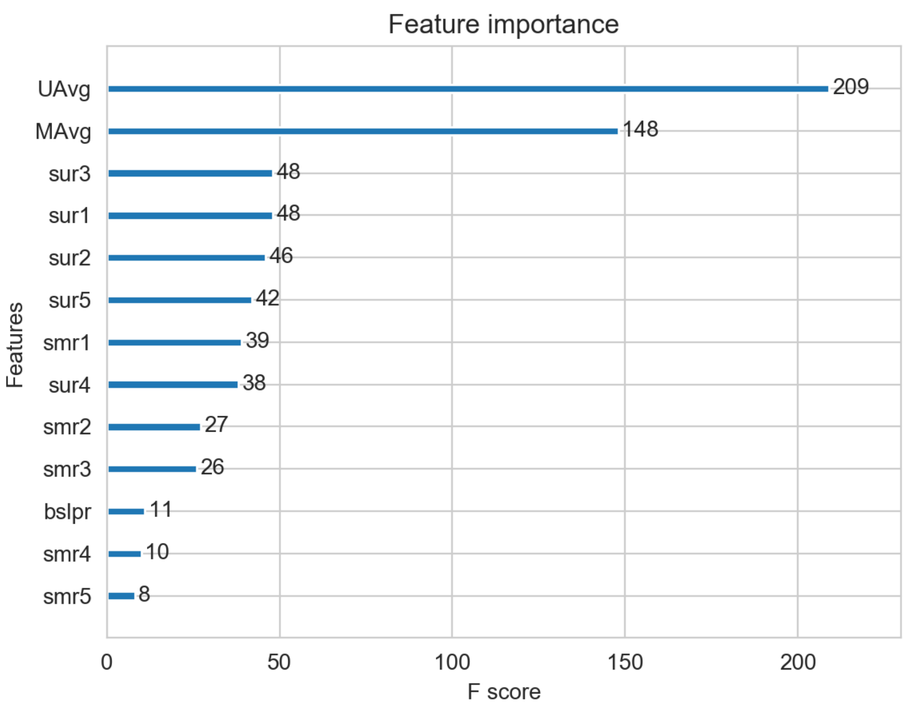
Out[84]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	U
0	808635	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581
1	941866	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581

In [85]:

```
# 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..
[19:45:51] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
Done. Time taken : 0:00:08.235488

Done

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

TEST DATA
------------------------------
RMSE :  1.0765603714651855
MAPE :  34.4648051883444
```



4.4.4 Surprise KNNBaseline predictor

In [86]:

```
from surprise import KNNBaseline
```

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

- PEARSON_BASELINE SIMILARITY
  - http://surprise.readthedocs.io/en/stable/similarities.html#surprise.similarities.pearson_baseline

- SHRINKAGE
  - *2.2 Neighborhood Models* in http://courses.ischool.berkeley.edu/i290-dm/s11/SECURE/a1-koren.pdf

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

\begin{align} \hat{r}_{ui} = b_{ui} + \frac{ \sum\limits_{v \in N^k_i(u)} \text{sim}(u, v) \cdot (r_{vi} - b_{vi})} {\sum\limits_{v \in N^k_i(u)} \text{sim}(u, v)} \end{align}

- \pmb{b_{ui}} - *Baseline prediction* of (user,movie) rating
- \pmb {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 pearsonBaseline similarity (we take base line predictions instead of mean rating of user/item)

- **Predicted rating** (based on Item Item similarity): \begin{align} \hat{r}_{ui} = b_{ui} + \frac{ \sum\limits_{j \in N^k_u(i)}\text{sim}(i, j) \cdot (r_{uj} - b_{uj})} {\sum\limits_{j \in N^k_u(j)} \text{sim}(i, j)} \end{align}
  - ***Notations follows same as above (user user based predicted rating)***

4.4.4.1 Surprise KNNBaseline with user user similarities

In [87]:

```
# 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,
verbose=True)

# Just store these error metrics in our models_evaluation datastructure
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 : 0:00:25.881378

Evaluating the model with train data..
time taken : 0:01:37.503940
```

```
---------------
Train Data
---------------
RMSE : 0.33642097416508826

MAPE : 9.145093375416348

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.061866
---------------
Test Data
---------------
RMSE : 1.0726493739667242

MAPE : 35.02094499698424

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:02:03.448036
```

4.4.4.2 Surprise KNNBaseline with movie movie similarities

In [88]:

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

# 'user_based' : Fals => 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,
verbose=True)

# 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 : 0:00:00.812559

Evaluating the model with train data..
time taken : 0:00:07.526391
---------------
Train Data
---------------
RMSE : 0.32584796251610554

MAPE : 8.447062581998374

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.059740
---------------
Test Data
---------------
RMSE : 1.072758832653683

MAPE : 35.02269653015042
```

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

---------------------------------------------
Total time taken to run this algorithm : 0:00:08.399141
```

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

- First we will run XGBoost with predictions from both KNN's (that uses User_User and Item_Item similarities along with our previous features.
- Then we will run XGBoost with just predictions form both knn models and preditions from our baseline model.

Preparing Train data

In [89]:

```python
# 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)
```

Out[89]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg	rating	bslpr	knn_b
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	3.370370	4.092437	4	3.898982	3.9
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	3.555556	4.092437	3	3.371403	3.1

Preparing Test data

In [90]:

```python
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)
```

Out[90]:

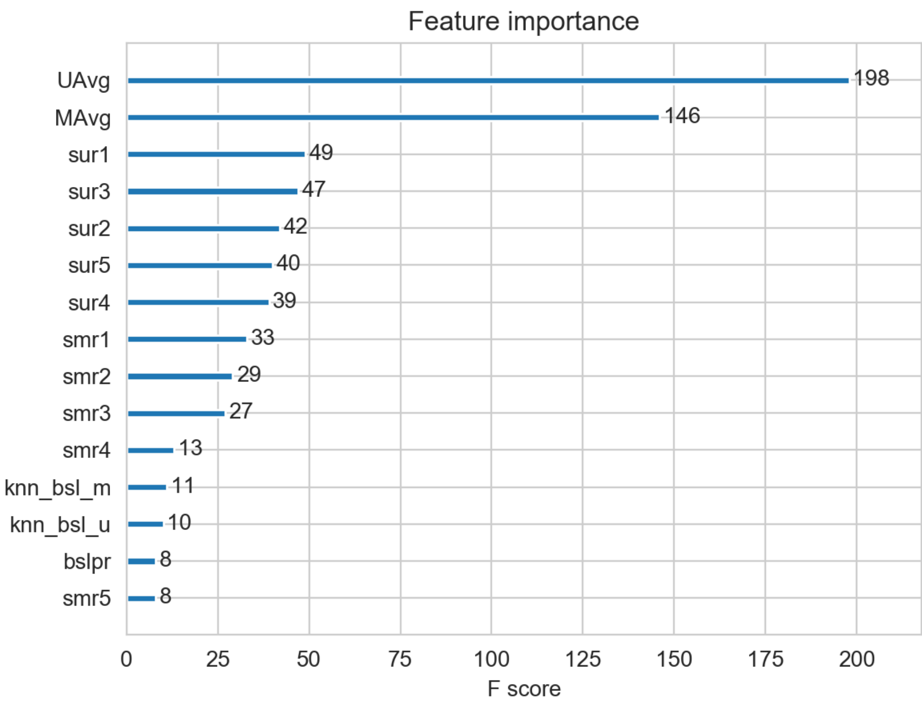
	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	U
0	808635	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581
1	941866	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581

In [91]:

```python
# 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..
[19:48:12] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Done. Time taken : 0:00:09.399726

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 intractions

In [92]:

```
from surprise import SVD
```

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

- Predicted Rating :

```
- $ \large  \hat r_{ui} = \mu + b_u + b_i + q_i^Tp_u $

    - $\pmb q_i$ - Representation of item(movie) in latent factor space

    - $\pmb p_u$ - Representation of user in new latent factor space
```

- A BASIC MATRIX FACTORIZATION MODEL in https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf

- Optimization problem with user item interactions and regularization (to avoid overfitting)

    - $\large \sum_{r_{ui} \in R_{train}} \left(r_{ui} - \hat{r}_{ui} \right)^2 +

\lambda\left(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2\right) $

In [93]:

```
# initiallize 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 : 0:00:06.302436

Evaluating the model with train data..
time taken : 0:00:00.975534
---------------
Train Data
---------------
RMSE : 0.6574721240954099

MAPE : 19.704901088660474

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.058222
---------------
Test Data
---------------
RMSE : 1.0726046873826458

MAPE : 35.01953535988152

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:00:07.336675
```

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

In [94]:

```
from surprise import SVDpp
```

- -----> 2.5 Implicit Feedback in http://courses.ischool.berkeley.edu/i290-dm/s11/SECURE/a1-koren.pdf

- Predicted Rating :

```
- $ \large \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) $
```

- \pmb{I_u} --- the set of all items rated by user u
- \pmb{y_j} --- Our new set of item factors that capture implicit ratings.

- Optimization problem with user item interactions and regularization (to avoid overfitting)

```
- $ \large \sum_{r_{ui} \in R_{train}} \left(r_{ui} - \hat{r}_{ui} \right)^2 +
```

\lambda\left(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2 + ||y_j||^2\right) $

In [95]:

```
# initiallize 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 : 0:01:55.829117

Evaluating the model with train data..
time taken : 0:00:05.887211
---------------
Train Data
---------------
RMSE : 0.6032438403305899

MAPE : 17.49285063490268

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.060804
---------------
Test Data
---------------
```

```
RMSE : 1.0728491944183447

MAPE : 35.03817913919887

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:02:01.777731
```

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

Preparing Train data

In [96]:

```
# 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)
```

Out[96]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	...	smr4	smr5	UAvg	MAvg	rating	bslpr	knn_bsl_
0	53406	33	3.581679	4.0	5.0	5.0	4.0	1.0	5.0	2.0	...	3.0	1.0	3.370370	4.092437	4	3.898982	3.9300
1	99540	33	3.581679	5.0	5.0	5.0	4.0	5.0	3.0	4.0	...	3.0	5.0	3.555556	4.092437	3	3.371403	3.1773

2 rows × 21 columns

Preparing Test data

In [97]:

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

reg_test_df.head(2)
```

Out[97]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	...	smr4	smr5	UAvg	
0	808635	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	...	3.581679	3.581679	3.581679	3
1	941866	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	...	3.581679	3.581679	3.581679	3

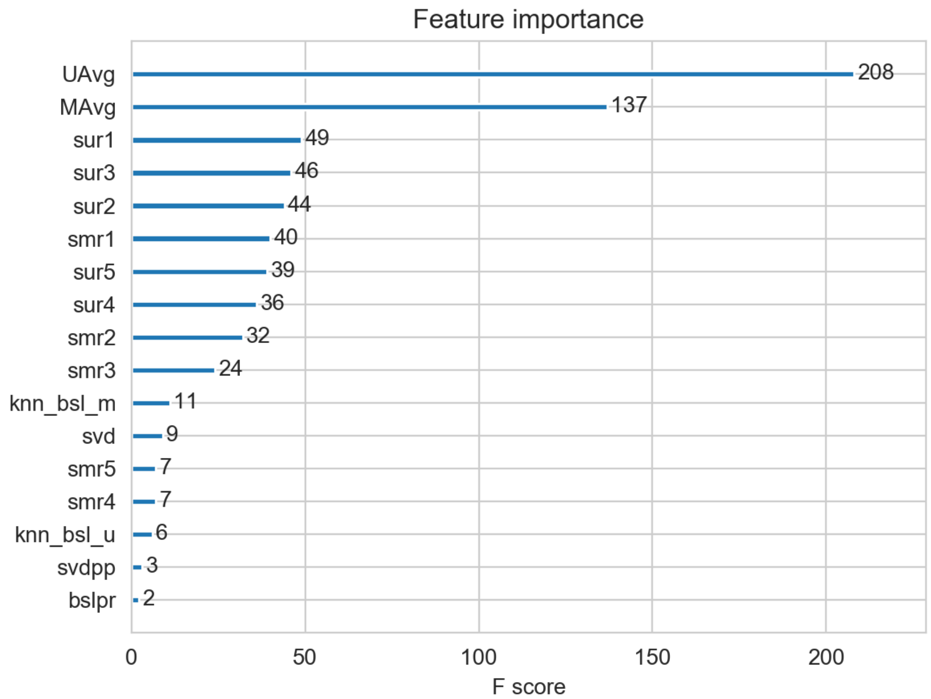
2 rows × 21 columns

In [98]:

```
# 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..
[19:50:31] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
Done. Time taken : 0:00:11.363650

Done

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

TEST DATA
------------------------------
RMSE :  1.0769599573828592
MAPE :  34.431788329400995
```



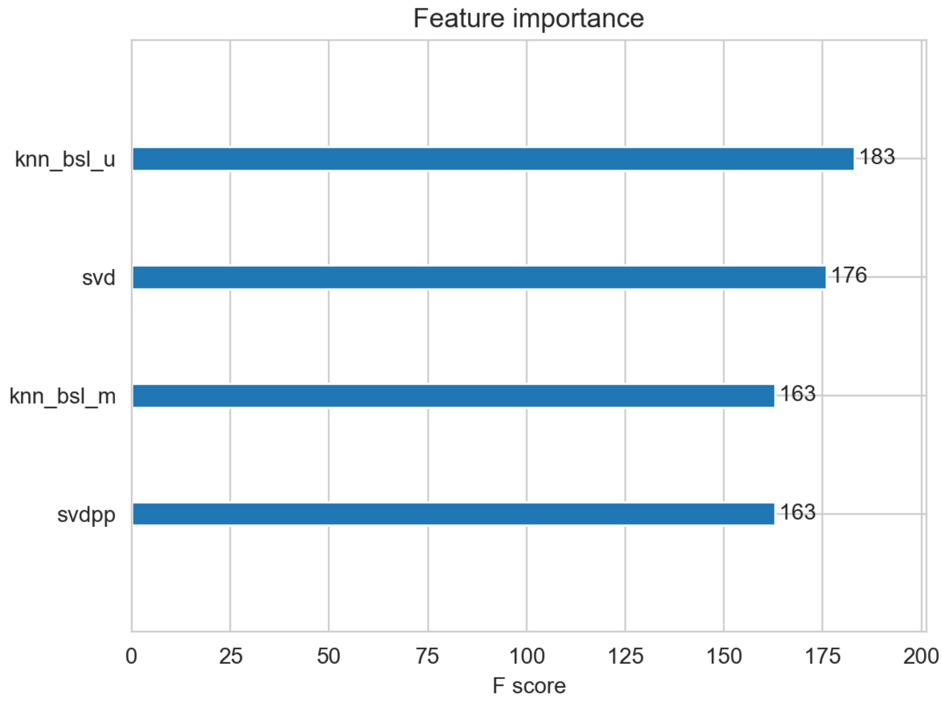
4.4.8 XgBoost with Surprise Baseline + Surprise KNNbaseline + MF Techniques

In [99]:

```
# 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()
```

```
plt.show()
```

```
Training the model..
[19:50:44] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Done. Time taken : 0:00:04.508906

Done

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

TEST DATA
------------------------------
RMSE :  1.0753047860953797
MAPE :  35.07058962951319
```



4.5 Comparision between all models

In [100]:

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

Out[100]:

```
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
```

In [101]:

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

```
----------------------------------------------------------------------------------------------------
Total time taken to run this entire notebook ( with saved files) is : 1:03:28.197192
```

In [49]:

```
from prettytable import PrettyTable

table = PrettyTable()
table.field_names = ['Model', 'Test Data RMSE', 'Test Data MAPE']

table.add_row(['XGBoost with initial 13 features', 1.076373581778953, 34.48223172520999])
table.add_row(['Suprise BaselineModel', 1.0730330260516174, 35.04995544572911])
table.add_row(['XGBoost with initial 13 features + Surprise Baseline predictor',
1.0765603714651855, 34.4648051883444])
table.add_row(['Surprise KNNBaseline with user user similarities', 1.0726493739667242,
35.02094499698424])
table.add_row(['Surprise KNNBaseline with movie movie similarities', 1.072758832653683,
35.02269653015042])
table.add_row(['XGBoost with initial 13 features + Surprise Baseline predictor + KNNBaseline predi
ctor', 1.0767793575625662, 34.44745951378593])
table.add_row(['SVD Matrix Factorization User Movie intractions', 1.0726046873826458,
35.01953535988152])
table.add_row(['SVD Matrix Factorization with implicit feedback from user ( user rated movies )',
1.0728491944183447, 35.03817913919887])
table.add_row(['XgBoost with 13 features + Surprise Baseline + Surprise KNNbaseline + MF Technique
s', 1.0769599573828592, 34.431788329400995])
table.add_row(['XgBoost with Surprise Baseline + Surprise KNNbaseline + MF Techniques', 1.07530478
60953797, 35.07058962951319])

print(table)
```

```
+-----------------------------------------------------------------------------------------+---------
-------+--------------------+
|                                          Model                                          |   Test D
RMSE    |   Test Data MAPE   |
+-----------------------------------------------------------------------------------------+---------
-------+--------------------+
|                             XGBoost with initial 13 features                            | 1.07637
1778953  | 34.48223172520999  |
|                                  Suprise BaselineModel                                  |
1.0730330260516174 | 35.04995544572911  |
|              XGBoost with initial 13 features + Surprise Baseline predictor             |
1.0765603714651855 |  34.4648051883444  |
|                     Surprise KNNBaseline with user user similarities                    | 1.072649
739667242 | 35.02094499698424  |
|                    Surprise KNNBaseline with movie movie similarities                   | 1.072758
32653683  | 35.02269653015042  |
| XGBoost with initial 13 features + Surprise Baseline predictor + KNNBaseline predictor | 1.07677
93575625662 | 34.44745951378593  |
|                     SVD Matrix Factorization User Movie intractions                     |
1.0726046873826458 | 35.01953535988152  |
|     SVD Matrix Factorization with implicit feedback from user ( user rated movies )     |
1.0728491944183447 | 35.03817913919887  |
|  XgBoost with 13 features + Surprise Baseline + Surprise KNNbaseline + MF Techniques   |
1.0769599573828592 | 34.431788329400995 |
|          XgBoost with Surprise Baseline + Surprise KNNbaseline + MF Techniques          | 1.075304
7860953797 | 35.07058962951319  |
+-----------------------------------------------------------------------------------------+---------
-------+--------------------+
```

5. Assignment

1.Instead of using 10K users and 1K movies to train the above models, use 25K users and 3K movies (or more) to train all of the above models. Report the RMSE and MAPE on the test data using larger amount of data and provide a comparison between various models as shown above.

NOTE: Please be patient as some of the code snippets make take many hours to compelte execution.

2.Tune hyperparamters of all the Xgboost models above to improve the RMSE.

In [102]:

```
%%javascript
// Converts integer to roman numeral
// https://github.com/kmahelona/ipython_notebook_goodies
// https://kmahelona.github.io/ipython_notebook_goodies/ipython_notebook_toc.js
function romanize(num) {
    var lookup = {M:1000,CM:900,D:500,CD:400,C:100,XC:90,L:50,XL:40,X:10,IX:9,V:5,IV:4,I:1},
 roman = '',
     i;
 for ( i in lookup ) {
     while ( num >= lookup[i] ) {
  roman += i;
  num -= lookup[i];
     }
 }
 return roman;
 }

// Builds a <ul> Table of Contents from all <headers> in DOM
function createTOC(){
    var toc = "";
    var level = 0;
    var levels = {}
    $('#toc').html('');

    $(":header").each(function(i){
     if (this.id=='tocheading'){return;}

     var titleText = this.innerHTML;
     var openLevel = this.tagName[1];

     if (levels[openLevel]){
  levels[openLevel] += 1;
     } else{
  levels[openLevel] = 1;
     }

     if (openLevel > level) {
  toc += (new Array(openLevel - level + 1)).join('<ul class="toc">');
     } else if (openLevel < level) {
  toc += (new Array(level - openLevel + 1)).join("</ul>");
  for (i=level;i>openLevel;i--){levels[i]=0;}
     }

     level = parseInt(openLevel);


     if (this.id==''){this.id = this.innerHTML.replace(/ /g,"-")}
     var anchor = this.id;

     toc += '<li><a style="text-decoration:none", href="#' + encodeURIComponent(anchor) + '">' + ti
tleText + '</a></li>';

 });


    if (level) {
 toc += (new Array(level + 1)).join("</ul>");
    }


    $('#toc').append(toc);

};

// Executes the createToc function
setTimeout(function(){createTOC();},100);

// Rebuild to TOC every minute
```

```
setInterval(function(){createTOC();},60000);
```

In [ ]:

```

```

5.1 Assignment

1.Instead of using 10K users and 1K movies to train the above models, use 25K users and 3K movies (or more) to train all of the above models. Report the RMSE and MAPE on the test data using larger amount of data and provide a comparison between various models as shown above.

NOTE: Please be patient as some of the code snippets make take many hours to compelte execution.

5.1 Sampling Data (30K users and 3K movies)

5.1.1 Build sample train data from the train data

In [5]:

```
start = datetime.now()
path = "assignment/final/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 30k users and 3k movies from available data
    sample_train_sparse_matrix = get_sample_sparse_matrix(train_sparse_matrix, no_users=30000, no_m
ovies=3000,
                                             path = path)

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

```
It is present in your pwd, getting it from disk....
DONE..
0:00:00.065053
```

5.1.2 Build sample test data from the test data

In [6]:

```
start = datetime.now()

path = "assignment/final/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 30k users and 3k movies from available data
    sample_test_sparse_matrix = get_sample_sparse_matrix(test_sparse_matrix, no_users=30000, no_mov
ies=3000,
                                                 path = path)
print(datetime.now() - start)
```

```
It is present in your pwd, getting it from disk....
DONE..
0:00:00.039727
```

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

In [7]:

```
sample_train_averages = dict()
```

5.2.1 Finding Global Average of all movie ratings

In [8]:

```
# 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
```

Out[8]:

```
{'global': 3.5902997718873504}
```

5.2.2 Finding Average rating per User

In [9]:

```
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
```

5.2.3 Finding Average rating per Movie

In [10]:

```
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 : 2.7974683544303796
```

5.3 Featurizing data

In [11]:

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

```
 No of ratings in Our Sampled train matrix is : 1029316


 No of ratings in Our Sampled test  matrix is : 313929
```

5.3.1 Featurizing data for regression problem

5.3.1.1 Featurizing train data

In [12]:

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

In [13]:

```
# Create separate functions to calculate similar users and similar movies rated by The user
def get_similar_user_train(sample_train_sparse_matrix, user, movie):
    '''
    Get top 5 similar users rating for the movie
    '''
    #     print(user, movie)
    #--------------------- Ratings of "movie" by similar users of "user" ---------------------
    # compute the similar Users of the "user"
    user_sim = cosine_similarity(sample_train_sparse_matrix[user],
sample_train_sparse_matrix).ravel()
    top_sim_users = user_sim.argsort()[::-1][1:] # we are ignoring 'The User' from its similar user
s.
    # 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=" ")
    return top_sim_users_ratings


def get_similar_movies_rated_by_user_train(sample_train_sparse_matrix, user, movie):
    '''
    Get top 5 similar movies rated by the user
    '''
    #--------------------- 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
.T).ravel()
    top_sim_movies = movie_sim.argsort()[::-1][1:] # we are ignoring 'The User' from its similar us
ers.
    # 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=" : -- ")
    return top_sim_movies_ratings
```

In [14]:

```
# Implemented multithreading instead of multiprocessing because we need same matrix values for com
puting similar user
# rating and similar movies rated by the user
############################################################
# It took me almost 60 hours to prepare this train dataset.#
############################################################
import concurrent.futures
start = datetime.now()
path = 'assignment/final/reg_train.csv'
if os.path.isfile(path):
    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(path, mode='w') as reg_data_file:
        count = 0
        for (user, movie, rating)  in zip(sample_train_users, sample_train_movies,
sample_train_ratings):
            st = datetime.now()
#         #     print(user, movie)
#             #--------------------- Ratings of "movie" by similar users of "user" ----------------
----
#             # compute the similar Users of the "user"
#             user_sim = cosine_similarity(sample_train_sparse_matrix[user],
sample_train_sparse_matrix).ravel()
#             top_sim_users = user_sim.argsort()[::-1][1:] # we are ignoring 'The User' from its
similar users.
#             # get the ratings of most similar users for this movie
#             top_ratings = sample_train_sparse_matrix[top_sim_users, movie].toarray().ravel()
#             # we will make it's length "5" by adding movie averages to .
#             top_sim_users_ratings = list(top_ratings[top_ratings != 0][:5])
#             top_sim_users_ratings.extend([sample_train_averages['movie'][movie]]*(5 -
```

```python
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_spa
rse_matrix.T).ravel()
#              top_sim_movies = movie_sim.argsort()[::-1][1:] # we are ignoring 'The User' from its
similar users.
#              # get the ratings of most similar movie rated by this user..
#              top_ratings = sample_train_sparse_matrix[user, top_sim_movies].toarray().ravel()
#              # we will make it's length "5" by adding user averages to.
#              top_sim_movies_ratings = list(top_ratings[top_ratings != 0][:5])
#              top_sim_movies_ratings.extend([sample_train_averages['user'][user]]*(5-len(top_sim_mo
vies_ratings)))
        #     print(top_sim_movies_ratings, end=" : -- ")


            #-----------------Threading Ratings of "movie" by similar users of "user"-------------
--#
            #-----------------Threading Ratings by "user"  to similar movies of "movie"-----------
--#
            with concurrent.futures.ThreadPoolExecutor() as executor:
                # similar user rating thread
                user_thread = executor.submit(get_similar_user_train, sample_train_sparse_matrix, u
ser, movie)

                # similar movie rating by user thread
                movie_thread = executor.submit(get_similar_movies_rated_by_user_train,
sample_train_sparse_matrix, user, movie)

                # collect result
                top_sim_users_ratings  = user_thread.result()
                top_sim_movies_ratings = movie_thread.result()

            #-----------------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)%1000 == 0:
                # print(','.join(map(str, row)))
                print("Done for {} rows----- {}".format(count, datetime.now() - start))


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

```
File already exists you don't have to prepare again...
0:00:00.000520
```

In [15]:

```python
# ############################################################
# # It took me almost 10 hours to prepare this train dataset.#
# ############################################################
# start = datetime.now()
```

```
# path = 'assignment/medium/reg_train.csv'
# if os.path.isfile(path):
#     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(path, mode='w') as reg_data_file:
#         count = 0
#         for (user, movie, rating)  in zip(sample_train_users, sample_train_movies,
sample_train_ratings):
#             st = datetime.now()
#         #     print(user, movie)
#             #--------------------- Ratings of "movie" by similar users of "user" ----------------
----
#             # compute the similar Users of the "user"
#             user_sim = cosine_similarity(sample_train_sparse_matrix[user],
sample_train_sparse_matrix).ravel()
#             top_sim_users = user_sim.argsort()[::-1][1:] # we are ignoring 'The User' from its
similar users.
#             # get the ratings of most similar users for this movie
#             top_ratings = sample_train_sparse_matrix[top_sim_users, movie].toarray().ravel()
#             # we will make it's length "5" by adding movie averages to .
#             top_sim_users_ratings = list(top_ratings[top_ratings != 0][:5])
#             top_sim_users_ratings.extend([sample_train_averages['movie'][movie]]*(5 -
len(top_sim_users_ratings)))
#         #     print(top_sim_users_ratings, end=" ")


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

In [2]:

```
reg_train = pd.read_csv('assignment/final/reg_train.csv', names = ['user', 'movie', 'GAvg', 'sur1', 'sur2', 'sur3', 'sur4', 'sur5','smr1', 'smr2', 'smr3', 'smr4', 'smr5', 'UAvg', 'MAvg', 'rating'], header=None)
reg_train.head()
```

Out[2]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg	rating
0	174683	10	3.5903	5.0	5.0	3.0	3.0	4.0	3.0	5.0	4.0	3.0	2.0	3.882353	3.578947	5
1	233949	10	3.5903	4.0	4.0	5.0	1.0	3.0	3.0	2.0	2.0	3.0	3.0	2.692308	3.578947	3
2	555770	10	3.5903	3.0	4.0	5.0	4.0	4.0	4.0	4.0	5.0	2.0	4.0	3.795455	3.578947	4
3	767518	10	3.5903	2.0	5.0	4.0	4.0	3.0	5.0	5.0	4.0	4.0	3.0	3.884615	3.578947	5
4	894393	10	3.5903	3.0	5.0	4.0	4.0	3.0	4.0	4.0	4.0	4.0	4.0	4.000000	3.578947	4

- **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.

5.3.1.2 Featurizing test data

In [17]:

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

In [18]:

```
sample_train_averages['global']
```

Out[18]:

```
3.5902997718873504
```

In [19]:

```
# Test data sampling, create separate functions to calculate similar users and similar movies rated by The user
def get_similar_user_test(sample_train_sparse_matrix, user, movie):
    '''
    Get top 5 similar users rating for the movie
    '''
    try:
        # compute the similar Users of the "user"
        user_sim = cosine_similarity(sample_train_sparse_matrix[user], sample_train_sparse_matrix).ravel()
        top_sim_users = user_sim.argsort()[::-1][1:] # we are ignoring 'The User' from its similar users.
        # get the ratings of most similar users for this movie
        top_ratings = sample_train_sparse_matrix[top_sim_users, movie].toarray().ravel()
        # we will make it's length "5" by adding movie averages to .
        top_sim_users_ratings = list(top_ratings[top_ratings != 0][:5])
        top_sim_users_ratings.extend([sample_train_averages['movie'][movie]]*(5 -
```

```
len(top_sim_users_ratings)))
        # print(top_sim_users_ratings, end="--")

    except (IndexError, KeyError):
        # It is a new User or new Movie or there are no ratings for given user for top similar
movies...
        ########## Cold STart Problem ##########
        top_sim_users_ratings.extend([sample_train_averages['global']]*(5 -
len(top_sim_users_ratings)))
        #print(top_sim_users_ratings)
    except:
        print(user, movie)
        # we just want KeyErrors to be resolved. Not every Exception...
        raise

    return top_sim_users_ratings


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

top_sim_movies_ratings.extend([sample_train_averages['global']]*(5-len(top_sim_movies_ratings)))
        #print(top_sim_movies_ratings)
    except :
        raise

    return top_sim_movies_ratings
```

In [20]:

```
# ############################################################
# # It took me almost 20 hours to prepare this train dataset.#
# ############################################################
start = datetime.now()

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

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

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

```
                top_sim_users_ratings.extend([sample_train_averages['movie'][movie]]*(5 -
len(top_sim_users_ratings)))
                # print(top_sim_users_ratings, end="--")

            except (IndexError, KeyError):
                # It is a new User or new Movie or there are no ratings for given user for top simi
lar movies...
                ########## Cold STart Problem ##########
                top_sim_users_ratings.extend([sample_train_averages['global']]*(5 -
len(top_sim_users_ratings)))
                #print(top_sim_users_ratings)
            except:
                print(user, movie)
                # we just want KeyErrors to be resolved. Not every Exception...
                raise



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

top_sim_movies_ratings.extend([sample_train_averages['global']]*(5-len(top_sim_movies_ratings)))
                #print(top_sim_movies_ratings)
            except :
                raise

#              #-----------------Threading Ratings of "movie" by similar users of "user"------------
----#
#              #-----------------Threading Ratings by "user"  to similar movies of "movie"----------
----#
#              with concurrent.futures.ThreadPoolExecutor() as executor:
#                  # similar user rating thread
#                  user_thread = executor.submit(get_similar_user_test, sample_train_sparse_matrix,
user, movie)

#                  # similar movie rating by user thread
#                  movie_thread = executor.submit(get_similar_movies_rated_by_user_test, sample_trai
n_sparse_matrix, user, movie)

#                  # collect result
#                  top_sim_users_ratings = user_thread.result()
#                  top_sim_movies_ratings = movie_thread.result()

            #-----------------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)
```

```
It is already created...
```

In [21]:

```
# start = datetime.now()

# path = 'assignment/final/reg_test.csv'
# if os.path.isfile(path):
#     print("It is already created...")
# else:

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

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

#             except (IndexError, KeyError):
#                 # It is a new User or new Movie or there are no ratings for given user for top si
milar movies...
#                 ########## Cold STart Problem ##########
#                 top_sim_users_ratings.extend([sample_train_averages['global']]*(5 - len(top_sim_u
sers_ratings)))
#                 #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_trai
_sparse_matrix.T).ravel()
#                   top_sim_movies = movie_sim.argsort()[::-1][1:] # we are ignoring 'The User' from
its similar users.
#                   # get the ratings of most similar movie rated by this user..
#                   top_ratings = sample_train_sparse_matrix[user, top_sim_movies].toarray().ravel()
#                   # we will make it's length "5" by adding user averages to.
#                   top_sim_movies_ratings = list(top_ratings[top_ratings != 0][:5])
#                   top_sim_movies_ratings.extend([sample_train_averages['user'][user]]*(5-len(top_s:
m_movies_ratings)))
#                   #print(top_sim_movies_ratings)
#               except (IndexError, KeyError):
#                   #print(top_sim_movies_ratings, end=" : -- ")
#                   top_sim_movies_ratings.extend([sample_train_averages['global']]*(5-
len(top_sim_movies_ratings)))
#                   #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

In [3]:

```
reg_test_df = pd.read_csv('assignment/final/reg_test.csv', names = ['user', 'movie', 'GAvg', 'sur1'
, 'sur2', 'sur3', 'sur4', 'sur5',
                                                            'smr1', 'smr2', 'smr3', 'smr4', 'smr5',
                                                            'UAvg', 'MAvg', 'rating'], header=None)
reg_test_df.head(4)
```

Out[3]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg	rating
0	1129620	2	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3
1	3321	5	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	4
2	368977	5	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	5
3	508584	5	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3

- **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.

5.3.2 Transforming data for Surprise models

In [9]:

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

5.3.2.1 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 models like SVD, KNNBaseLineOnly....etc..,in Surprise.
- We can form the trainset from a file, or from a Pandas DataFrame. http://surprise.readthedocs.io/en/stable/getting_started.html#load-dom-dataframe-py

In [10]:

```
# 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()
```

5.3.2.2 Transforming test data

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

In [11]:

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

Out[11]:

```
[(1129620, 2, 3), (3321, 5, 4), (368977, 5, 5)]
```

5.4 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)

In [12]:

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

models_evaluation_train, models_evaluation_test
```

Out[12]:

```
({}, {})
```

5.4.1 XGBoost with initial 13 features

In [13]:

```
import xgboost as xgb
```

In [17]:

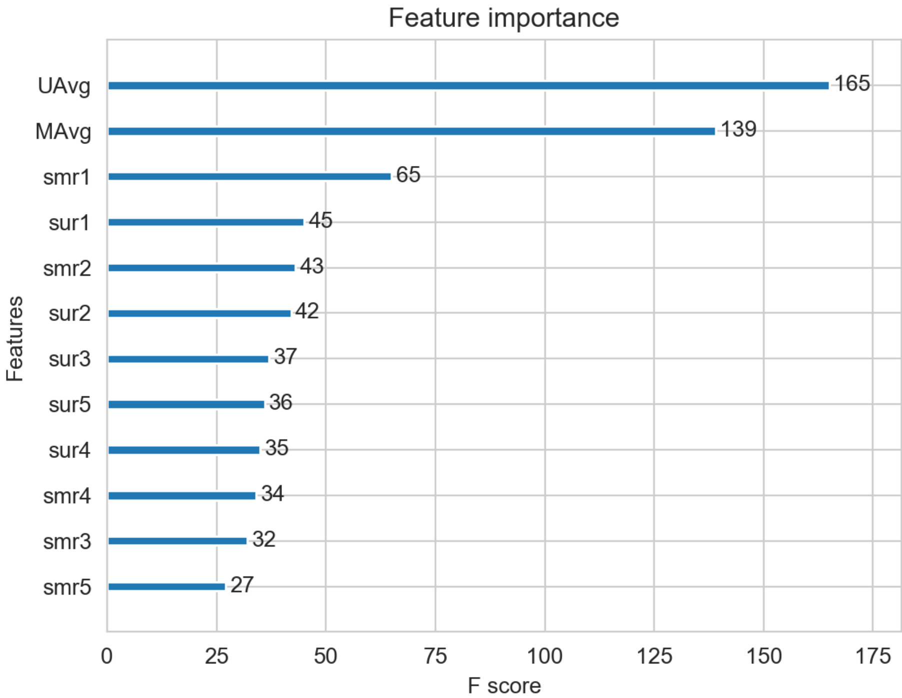
```
# prepare Train data
x_train = reg_train.drop(['user','movie','rating'], axis=1)
y_train = reg_train['rating']

# 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...
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..
```

```
/Users/mayankgupta/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/Users/mayankgupta/anaconda3/lib/python3.7/site-packages/xgboost/core.py:588: FutureWarning:
Series.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \
```

```
[16:58:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
Done. Time taken : 0:01:03.819119

Done

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

```
TEST DATA
------------------------------
RMSE :  1.0892270259040067
MAPE :  33.91250855584647
```



5.4.2 Suprise BaselineModel

In [18]:

```
from surprise import BaselineOnly
```

Predicted_rating : (baseline prediction)

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

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

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

Optimization function (Least Squares Problem)

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

> \large \sum_{r_{ui} \in R_{train}} \left(r_{ui} - (\mu + b_u + b_i)\right)^2 + \lambda \left(b_u^2 + b_i^2 \right).\text { [mimimize } {b_u, b_i]}

In [19]:

```
# 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)
```

```python
# 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 : 0:00:08.134921

Evaluating the model with train data..
time taken : 0:00:15.388887
---------------
Train Data
---------------
RMSE : 0.922396994629424

MAPE : 28.616128528159262

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:03.520589
---------------
Test Data
---------------
RMSE : 1.0812972316800673

MAPE : 34.08771602581355

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:00:27.044974
```

5.4.3 XGBoost with initial 13 features + Surprise Baseline predictor

Updating Train Data

In [20]:

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

Out[20]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg	rating	bslpr
0	174683	10	3.5903	5.0	5.0	3.0	3.0	4.0	3.0	5.0	4.0	3.0	2.0	3.882353	3.578947	5	3.675991
1	233949	10	3.5903	4.0	4.0	5.0	1.0	3.0	3.0	2.0	2.0	3.0	3.0	2.692308	3.578947	3	3.691201

Updating Test Data

In [21]:

```python
# 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)
```

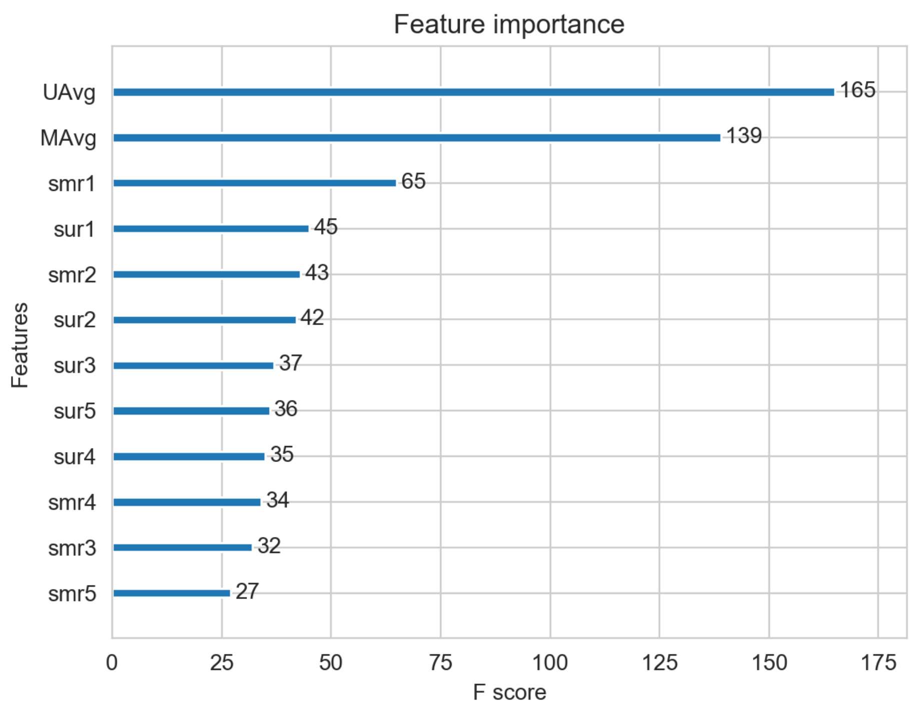
Out[21]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg	rating	bs
0	1129620	2	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3	3.59
1	3321	5	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	4	3.59

In [22]:

```
# 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..
[17:00:24] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
Done. Time taken : 0:01:17.988587

Done

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

TEST DATA
------------------------------
RMSE :  1.0892270259040067
MAPE :  33.91250855584647
```



5.4.4 Surprise KNNBaseline predictor

In [23]:

```
from surprise import KNNBaseline
```

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

- PEARSON_BASELINE SIMILARITY
  - http://surprise.readthedocs.io/en/stable/similarities.html#surprise.similarities.pearson_baseline

- SHRINKAGE
  - *2.2 Neighborhood Models* in http://courses.ischool.berkeley.edu/i290-dm/s11/SECURE/a1-koren.pdf

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

\begin{align} \hat{r}_{ui} = b_{ui} + \frac{ \sum\limits_{v \in N^k_i(u)} \text{sim}(u, v) \cdot (r_{vi} - b_{vi})} {\sum\limits_{v \in N^k_i(u)} \text{sim}(u, v)} \end{align}

- \pmb{b_{ui}} - *Baseline prediction* of (user,movie) rating
- \pmb {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 pearsonBaseline similarity (we take base line predictions instead of mean rating of user/item)

- **Predicted rating** (based on Item Item similarity): \begin{align} \hat{r}_{ui} = b_{ui} + \frac{ \sum\limits_{j \in N^k_u(i)}\text{sim}(i, j) \cdot (r_{uj} - b_{uj})} {\sum\limits_{j \in N^k_u(j)} \text{sim}(i, j)} \end{align}
  - ***Notations follows same as above (user user based predicted rating)***

5.4.4.1 Surprise KNNBaseline with user user similarities

In [24]:

```
# 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,
verbose=True)

# Just store these error metrics in our models_evaluation datastructure
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 : 0:39:35.141504

Evaluating the model with train data..
time taken : 0:33:47.263903
---------------
Train Data
---------------
RMSE : 0.4549495877986228

MAPE : 12.877854265412426

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:05.477787
---------------
Test Data
```

```
---------------
RMSE : 1.0815454676962384

MAPE : 34.07351172198283

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 1:13:27.908733
```

5.4.4.2 Surprise KNNBaseline with movie movie similarities

In [25]:

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

# 'user_based' : Fals => 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,
verbose=True)

# 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 : 0:00:12.610987

Evaluating the model with train data..
time taken : 0:01:57.373948
---------------
Train Data
---------------
RMSE : 0.5077513693386716

MAPE : 14.283684468196672

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:02.834246
---------------
Test Data
---------------
RMSE : 1.0817540629052533

MAPE : 34.07675953880748

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:02:12.819627
```

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

- - - First we will run XGBoost with predictions from both KNN's (that uses User_User and Item_Item similarities along with our previous features.

-
    -
        - Then we will run XGBoost with just predictions form both knn models and preditions from our baseline model.

Preparing Train data

In [26]:

```
# 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)
```

Out[26]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg	rating	bslpr	knn_bs
0	174683	10	3.5903	5.0	5.0	3.0	3.0	4.0	3.0	5.0	4.0	3.0	2.0	3.882353	3.578947	5	3.675991	4.984
1	233949	10	3.5903	4.0	4.0	5.0	1.0	3.0	3.0	2.0	2.0	3.0	3.0	2.692308	3.578947	3	3.691201	3.186

Preparing Test data

In [27]:

```
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)
```

Out[27]:

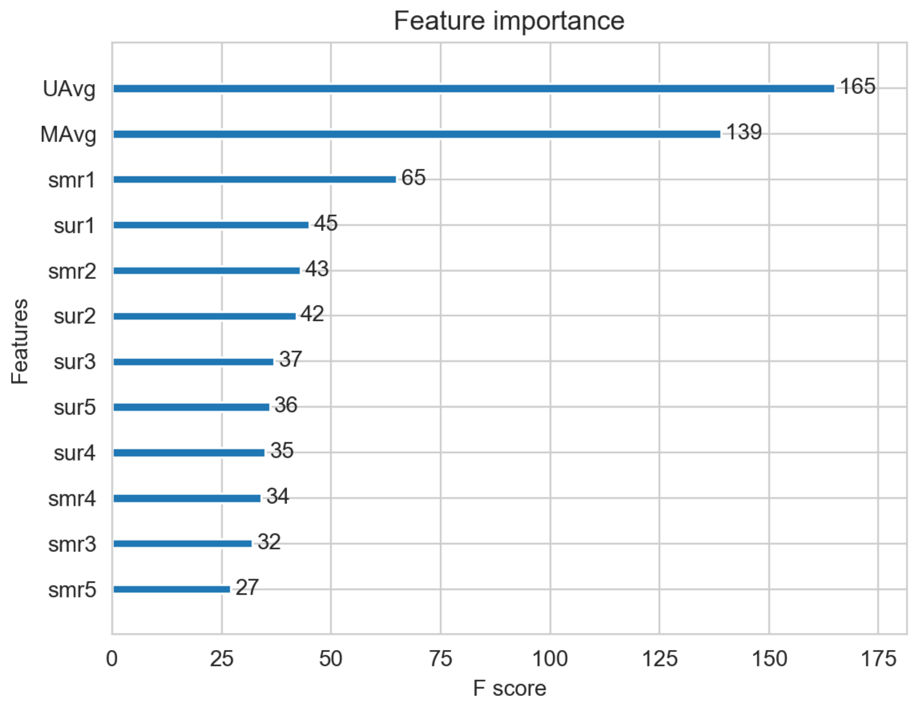
	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg	rating	bs
0	1129620	2	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3	3.59
1	3321	5	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	4	3.59

In [28]:

```
# 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..
[18:17:33] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
Done. Time taken : 0:01:20.531531

Done

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

TEST DATA
------------------------------
```

```
------------------------------
RMSE :  1.0892270259040067
MAPE :  33.91250855584647
```



5.4.6 Matrix Factorization Techniques

In [29]:

```
from surprise import SVD
```

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

- Predicted Rating :

```
- $ \large  \hat r_{ui} = \mu + b_u + b_i + q_i^Tp_u $

    - $\pmb q_i$ - Representation of item(movie) in latent factor space

    - $\pmb p_u$ - Representation of user in new latent factor space
```

- A BASIC MATRIX FACTORIZATION MODEL in https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf

- Optimization problem with user item interactions and regularization (to avoid overfitting)

```
- $\large \sum_{r_{ui} \in R_{train}} \left(r_{ui} - \hat{r}_{ui} \right)^2 +
```

\lambda\left(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2\right) $

In [30]:

```
# initiallize 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 : 0:00:47.988580

Evaluating the model with train data..
time taken : 0:00:08.146320
---------------
Train Data
---------------
RMSE : 0.6768482009451211

MAPE : 20.10641397000303

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:02.538424
---------------
Test Data
---------------
RMSE : 1.081447980675648

MAPE : 34.03251083980803

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:00:58.674898
```

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

In [31]:

```
from surprise import SVDpp
```

- -----> 2.5 Implicit Feedback in http://courses.ischool.berkeley.edu/i290-dm/s11/SECURE/a1-koren.pdf

- Predicted Rating :

    - $ \large \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) $

- \pmb{I_u} --- the set of all items rated by user u
- \pmb{y_j} --- Our new set of item factors that capture implicit ratings.

- Optimization problem with user item interactions and regularization (to avoid overfitting)

    - $ \large \sum_{r_{ui} \in R_{train}} \left(r_{ui} - \hat{r}_{ui} \right)^2 +

\lambda\left(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2 + ||y_j||^2\right) $

In [32]:

```
# initiallize 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 : 0:38:16.451446

Evaluating the model with train data..
time taken : 0:01:46.514664
---------------
Train Data
---------------
RMSE : 0.6690428706177345

MAPE : 19.359060771470638

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:04.291420
---------------
Test Data
---------------
RMSE : 1.0820843625280618

MAPE : 34.00031688894114

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:40:07.259585
```

5.4.7 XgBoost with 13 features + Surprise Baseline + Surprise KNNbaseline + MF Techniques

Preparing Train data

In [33]:

```
# add the predicted values from both knns to this dataframe
```

```python
reg_train['svd'] = models_evaluation_train['svd']['predictions']
reg_train['svdpp'] = models_evaluation_train['svdpp']['predictions']

reg_train.head(2)
```

Out[33]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	...	smr4	smr5	UAvg	MAvg	rating	bslpr	knn_bsl_u
0	174683	10	3.5903	5.0	5.0	3.0	3.0	4.0	3.0	5.0	...	3.0	2.0	3.882353	3.578947	5	3.675991	4.984922
1	233949	10	3.5903	4.0	4.0	5.0	1.0	3.0	3.0	2.0	...	3.0	3.0	2.692308	3.578947	3	3.691201	3.186564

2 rows × 21 columns

Preparing Test data

In [34]:

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

reg_test_df.head(2)
```

Out[34]:

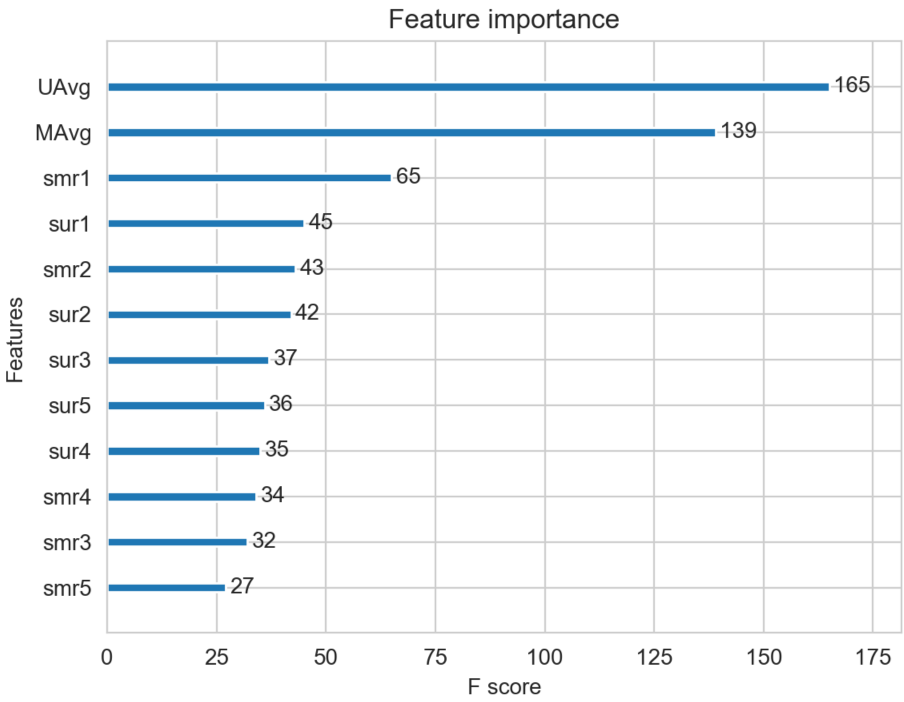
	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	...	smr4	smr5	UAvg	MAvg	rating	bslpr
0	1129620	2	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	...	3.5903	3.5903	3.5903	3.5903	3	3.5903
1	3321	5	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	3.5903	...	3.5903	3.5903	3.5903	3.5903	4	3.5903

2 rows × 21 columns

In [35]:

```python
# 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..
[19:00:08] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
Done. Time taken : 0:01:43.754865

Done

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

TEST DATA
------------------------------
RMSE :  1.0892270259040067
MAPE :  33.91250855584647
```

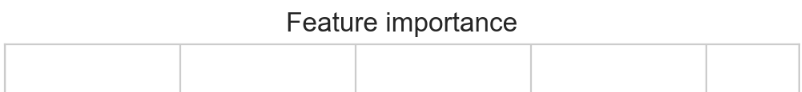


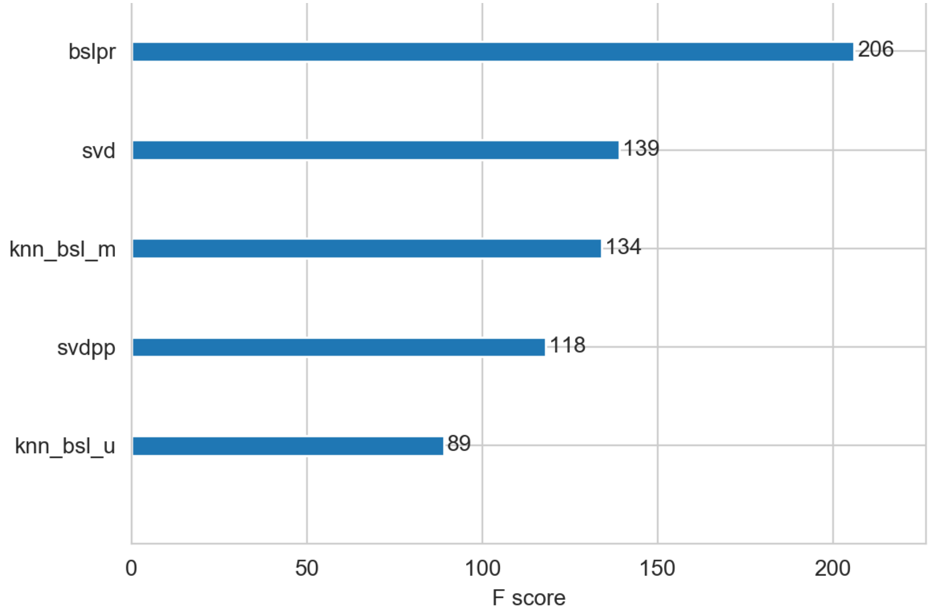
5.4.8 XgBoost with Surprise Baseline + Surprise KNNbaseline + MF Techniques

In [48]:

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

# test data
x_test = reg_test_df[['bslpr', '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..
[19:07:37] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
Done. Time taken : 0:00:56.646447

Done

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

TEST DATA
------------------------------
RMSE :  1.089130694762666
MAPE :  34.53940907923995
```





5.5 Comparision between all models

In [49]:

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

Out[49]:

```
bsl_algo          1.0812972316800673
svd                1.081447980675648
knn_bsl_u         1.0815454676962384
knn_bsl_m         1.0817540629052533
svdpp             1.0820843625280618
xgb_all_models     1.089130694762666
first_algo        1.0892270259040067
xgb_bsl           1.0892270259040067
xgb_knn_bsl       1.0892270259040067
xgb_final         1.0892270259040067
Name: rmse, dtype: object
```

In [50]:

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

```
----------------------------------------------------------------------------------------------------
Total time taken to run this entire notebook ( with saved files) is : 2:19:13.356329
```

In [51]:

```
from prettytable import PrettyTable

table = PrettyTable()
table.field_names = ['Model', 'Test Data RMSE', 'Test Data MAPE']

table.add_row(['XGBoost with initial 13 features', 1.0892270259040067, 33.91250855584647])
table.add_row(['Suprise BaselineModel', 1.0812972316800673, 34.08771602581355])
table.add_row(['XGBoost with initial 13 features + Surprise Baseline predictor',
1.0892270259040067, 33.91250855584647])
table.add_row(['Surprise KNNBaseline with user user similarities', 1.0815454676962384,
34.07351172198283])
table.add_row(['Surprise KNNBaseline with movie movie similarities', 1.0817540629052533,
34.07675953880748])
table.add_row(['XGBoost with initial 13 features + Surprise Baseline predictor + KNNBaseline predi
```

```
table.add_row(['XGBoost with initial 13 features + Surprise Baseline predictor + KNNBaseline predi
ctor', 1.0892270259040067, 33.91250855584647])
table.add_row(['SVD Matrix Factorization User Movie intractions', 1.081447980675648,
34.03251083980803])
table.add_row(['SVD Matrix Factorization with implicit feedback from user ( user rated movies )',
1.0820843625280618, 34.00031688894114])
table.add_row(['XgBoost with 13 features + Surprise Baseline + Surprise KNNbaseline + MF Technique
s', 1.0892270259040067, 33.91250855584647])
table.add_row(['XgBoost with Surprise Baseline + Surprise KNNbaseline + MF Techniques', 1.08913069
4762666, 34.53940907923995])

print(table)
```

```
+-----------------------------------------------------------------------------------------+---------
-------+-------------------+
|                                          Model                                          |   Test D
RMSE    |   Test Data MAPE  |
+-----------------------------------------------------------------------------------------+---------
-------+-------------------+
|                             XGBoost with initial 13 features                            | 1.089227
59040067 | 33.91250855584647 |
|                                  Suprise BaselineModel                                  |
1.0812972316800673 | 34.08771602581355 |
|              XGBoost with initial 13 features + Surprise Baseline predictor             |
1.0892270259040067 | 33.91250855584647 |
|                     Surprise KNNBaseline with user user similarities                    | 1.081545
676962384 | 34.07351172198283 |
|                    Surprise KNNBaseline with movie movie similarities                   | 1.081754
629052533 | 34.07675953880748 |
| XGBoost with initial 13 features + Surprise Baseline predictor + KNNBaseline predictor | 1.08922
70259040067 | 33.91250855584647 |
|                     SVD Matrix Factorization User Movie intractions                     |
1.081447980675648  | 34.03251083980803 |
|     SVD Matrix Factorization with implicit feedback from user ( user rated movies )     |
1.0820843625280618 | 34.00031688894114 |
|   XgBoost with 13 features + Surprise Baseline + Surprise KNNbaseline + MF Techniques   |
1.0892270259040067 | 33.91250855584647 |
|         XgBoost with Surprise Baseline + Surprise KNNbaseline + MF Techniques           | 1.089130
694762666  | 34.53940907923995 |
+-----------------------------------------------------------------------------------------+---------
-------+-------------------+
```

5.2 Assignment

2.Tune hyperparamters of all the Xgboost models above to improve the RMSE.

Initialize new dictionaries to store models output

In [52]:

```
models_evaluation_train_xgboost = dict()
models_evaluation_test_xgboost = dict()

models_evaluation_train_xgboost, models_evaluation_test_xgboost
```

Out[52]:

```
({}, {})
```

In [53]:

```
# print train columns
reg_train.columns
```

Out[53]:

```
Index(['user', 'movie', 'GAvg', 'sur1', 'sur2', 'sur3', 'sur4', 'sur5', 'smr1',
       'smr2', 'smr3', 'smr4', 'smr5', 'UAvg', 'MAvg', 'rating', 'bslpr',
       'knn_bsl_u', 'knn_bsl_m', 'svd', 'svdpp'],
      dtype='object')
```

In [54]:

```
# print test columns
reg_test_df.columns
```

Out[54]:

```
Index(['user', 'movie', 'GAvg', 'sur1', 'sur2', 'sur3', 'sur4', 'sur5', 'smr1',
       'smr2', 'smr3', 'smr4', 'smr5', 'UAvg', 'MAvg', 'rating', 'bslpr',
       'knn_bsl_u', 'knn_bsl_m', 'svd', 'svdpp'],
      dtype='object')
```

In [55]:

```
# create backup of original DataFrame
reg_train_orig = reg_train
reg_test_df_orig = reg_test_df
```

5.2.1 XGBoost with initial 13 features

Prepare Train Data

In [70]:

```
# prepare Train data
x_train = reg_train[['GAvg', 'sur1', 'sur2', 'sur3', 'sur4', 'sur5', 'smr1', 'smr2', 'smr3', 'smr4'
, 'smr5', 'UAvg', 'MAvg']]
#.drop(['user','movie','rating'], axis=1)
y_train = reg_train['rating']
```

Prepare Test Data

In [71]:

```
# Prepare Test data
x_test = reg_test_df[['GAvg', 'sur1', 'sur2', 'sur3', 'sur4', 'sur5', 'smr1', 'smr2', 'smr3',
'smr4', 'smr5', 'UAvg', 'MAvg']]
#.drop(['user','movie','rating'], axis=1)
y_test = reg_test_df['rating']
```

In [72]:

```
x_train.columns, x_test.columns
```

Out[72]:

```
(Index(['GAvg', 'sur1', 'sur2', 'sur3', 'sur4', 'sur5', 'smr1', 'smr2', 'smr3',
        'smr4', 'smr5', 'UAvg', 'MAvg'],
       dtype='object'),
 Index(['GAvg', 'sur1', 'sur2', 'sur3', 'sur4', 'sur5', 'smr1', 'smr2', 'smr3',
        'smr4', 'smr5', 'UAvg', 'MAvg'],
       dtype='object'))
```

Hyper Parameters

In [80]:

```
# max_depth=[int(x) for x in np.linspace(start=2, stop=10, num=9)]
# n_estimators=[int(x) for x in np.linspace(start=50, stop=300, num=6)]
# learning_rate=[float(x) for x in np.linspace(start=0.1, stop=0.9, num=9)]
# booster="gbtree", "gblinear",

# random_grid = {
#     'max_depth' : max_depth,
#     'n_estimators' : n_estimators,
#     'learning_rate' : learning_rate,
#     'booster' : booster
```

```
# }

random_grid={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
    'n_estimators':[100,200,500,1000,2000],
    'max_depth':[2,3,4,5],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}

random_grid
```

Out[80]:

```
{'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2],
 'n_estimators': [100, 200, 500, 1000, 2000],
 'max_depth': [2, 3, 4, 5],
 'colsample_bytree': [0.1, 0.3, 0.5, 1],
 'subsample': [0.1, 0.3, 0.5, 1]}
```

Hyper Parameter Tuning

In [62]:

```
from sklearn.model_selection import RandomizedSearchCV

# prams={
#     'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
#      'n_estimators':[100,200,500,1000,2000],
#      'max_depth':[3,5,10],
#     'colsample_bytree':[0.1,0.3,0.5,1],
#     'subsample':[0.1,0.3,0.5,1]
# }

# Use the random grid to search for best hyperparameters
# First create the base model to tune
#  class xgboost.XGBRegressor(max_depth=3, learning_rate=0.1, n_estimators=100, verbosity=1,
# objective='reg:squarederror', booster='gbtree', tree_method='auto',
# n_jobs=1, gamma=0, min_child_weight=1, max_delta_step=0, subsample=1,
# colsample_bytree=1, colsample_bylevel=1, colsample_bynode=1,
# reg_alpha=0, reg_lambda=1, scale_pos_weight=1, base_score=0.5,
# random_state=0, missing=None, num_parallel_tree=1, importance_type='gain', **kwargs)
xgb_random = xgb.XGBRegressor(n_jobs=-1, random_state=15)
# xgb_random = xgb.XGBRegressor(nthread=-1, objective='reg:linear', missing=None, seed=8)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
xgb_random = RandomizedSearchCV(estimator = xgb_random, param_distributions = random_grid, n_iter =
100, cv = 3, verbose=3, random_state=15, n_jobs = -1)
# Fit the random search model
xgb_random.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  24 tasks      | elapsed: 40.1min
[Parallel(n_jobs=-1)]: Done 120 tasks      | elapsed: 303.5min
[Parallel(n_jobs=-1)]: Done 280 tasks      | elapsed: 618.1min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 640.1min finished
```

```
[07:13:52] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

Out[62]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
                   estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                          colsample_bylevel=1,
                                          colsample_bynode=1,
                                          colsample_bytree=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=100,
```

```
                                            n_jobs=-1, nthread=None,
                                            objective='reg:linear',
                                            random_s...
                                            seed=None, silent=None, subsample=1,
                                            verbosity=1),
                   iid='warn', n_iter=100, n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.5, 1],
                                        'learning_rate': [0.01, 0.03, 0.05, 0.1,
                                                          0.15, 0.2],
                                        'max_depth': [2, 3, 4, 5],
                                        'n_estimators': [100, 200, 500, 1000,
                                                         2000],
                                        'subsample': [0.1, 0.3, 0.5, 1]},
                   pre_dispatch='2*n_jobs', random_state=15, refit=True,
                   return_train_score=False, scoring=None, verbose=3)
```

In [63]:

```
# select best params
xgb_random.best_params_
```

Out[63]:

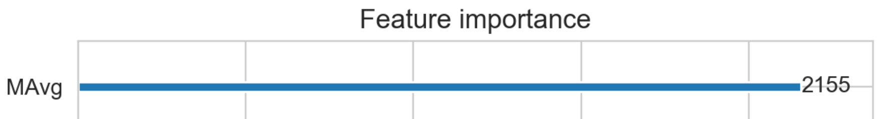
```
{'subsample': 1,
 'n_estimators': 1000,
 'max_depth': 4,
 'learning_rate': 0.1,
 'colsample_bytree': 0.3}
```

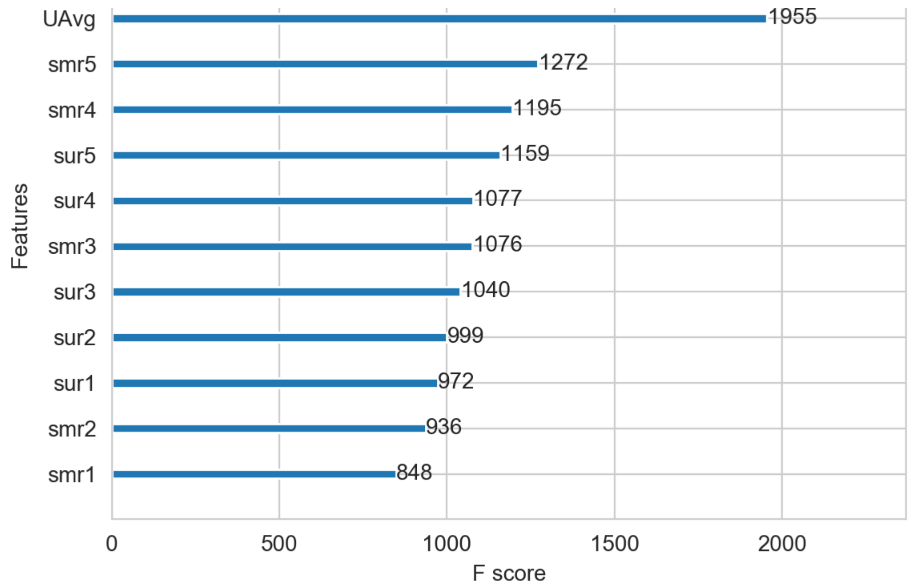
Run Model with Best Hyper Parameters

In [73]:

```
# {'subsample': 1,
#  'n_estimators': 1000,
#  'max_depth': 4,
#  'learning_rate': 0.1,
#  'colsample_bytree': 0.3}

# initialize Our first XGBoost model with 13 features...
first_xgb = xgb.XGBRegressor(silent=False, n_jobs=-1, random_state=15, subsample=1, n_estimators=10
00, max_depth=4, learning_rate=0.1, colsample_bytree=0.3)
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_xgboost['first_algo'] = train_results
models_evaluation_test_xgboost['first_algo'] = test_results

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

```
Training the model..
[18:49:39] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
Done. Time taken : 0:06:28.971470

Done

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

TEST DATA
------------------------------
RMSE :  1.1374496313732387
MAPE :  32.979426329406394
```





5.2.2 XGBoost with initial 13 features + Surprise Baseline predictor

Prepare Train Data

In [64]:

```
# prepare Train data
x_train = reg_train[['GAvg', 'sur1', 'sur2', 'sur3', 'sur4', 'sur5', 'smr1', 'smr2', 'smr3', 'smr4'
, 'smr5', 'UAvg', 'MAvg', 'bslpr']]
#.drop(['user','movie','rating'], axis=1)
y_train = reg_train['rating']
```

Prepare Test Data

In [65]:

```
# Prepare Test data
x_test = reg_test_df[['GAvg', 'sur1', 'sur2', 'sur3', 'sur4', 'sur5', 'smr1', 'smr2', 'smr3',
'smr4', 'smr5', 'UAvg', 'MAvg', 'bslpr']]
#.drop(['user','movie','rating'], axis=1)
y_test = reg_test_df['rating']
```

In [66]:

```
x_train.columns, x_test.columns
```

Out[66]:

```
(Index(['GAvg', 'sur1', 'sur2', 'sur3', 'sur4', 'sur5', 'smr1', 'smr2', 'smr3',
        'smr4', 'smr5', 'UAvg', 'MAvg', 'bslpr'],
       dtype='object'),
 Index(['GAvg', 'sur1', 'sur2', 'sur3', 'sur4', 'sur5', 'smr1', 'smr2', 'smr3',
        'smr4', 'smr5', 'UAvg', 'MAvg', 'bslpr'],
       dtype='object'))
```

Hyper Parameter Tuning

In [67]:

```
from sklearn.model_selection import RandomizedSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
#  class xgboost.XGBRegressor(max_depth=3, learning_rate=0.1, n_estimators=100, verbosity=1,
# objective='reg:squarederror', booster='gbtree', tree_method='auto',
# n_jobs=1, gamma=0, min_child_weight=1, max_delta_step=0, subsample=1,
```

```
# colsample_bytree=1, colsample_bylevel=1, colsample_bynode=1,
# reg_alpha=0, reg_lambda=1, scale_pos_weight=1, base_score=0.5,
# random_state=0, missing=None, num_parallel_tree=1, importance_type='gain', **kwargs)
xgb_random = xgb.XGBRegressor(n_jobs=-1, random_state=15)
# xgb_random = xgb.XGBRegressor(nthread=-1, objective='reg:linear', missing=None, seed=8)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
xgb_random = RandomizedSearchCV(estimator = xgb_random, param_distributions = random_grid, n_iter =
100, cv = 3, verbose=3, random_state=15, n_jobs = -1)
# Fit the random search model
xgb_random.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  24 tasks      | elapsed: 40.7min
[Parallel(n_jobs=-1)]: Done 120 tasks      | elapsed: 216.1min
[Parallel(n_jobs=-1)]: Done 280 tasks      | elapsed: 633.2min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 666.1min finished
```

```
[18:26:13] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

Out[67]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
                   estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                          colsample_bylevel=1,
                                          colsample_bynode=1,
                                          colsample_bytree=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=100,
                                          n_jobs=-1, nthread=None,
                                          objective='reg:linear',
                                          random_s...
                                          seed=None, silent=None, subsample=1,
                                          verbosity=1),
                   iid='warn', n_iter=100, n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.5, 1],
                                        'learning_rate': [0.01, 0.03, 0.05, 0.1,
                                                          0.15, 0.2],
                                        'max_depth': [2, 3, 4, 5],
                                        'n_estimators': [100, 200, 500, 1000,
                                                         2000],
                                        'subsample': [0.1, 0.3, 0.5, 1]},
                   pre_dispatch='2*n_jobs', random_state=15, refit=True,
                   return_train_score=False, scoring=None, verbose=3)
```

In [68]:

```
# select best params
xgb_random.best_params_
```

Out[68]:

```
{'subsample': 1,
 'n_estimators': 1000,
 'max_depth': 4,
 'learning_rate': 0.1,
 'colsample_bytree': 0.3}
```

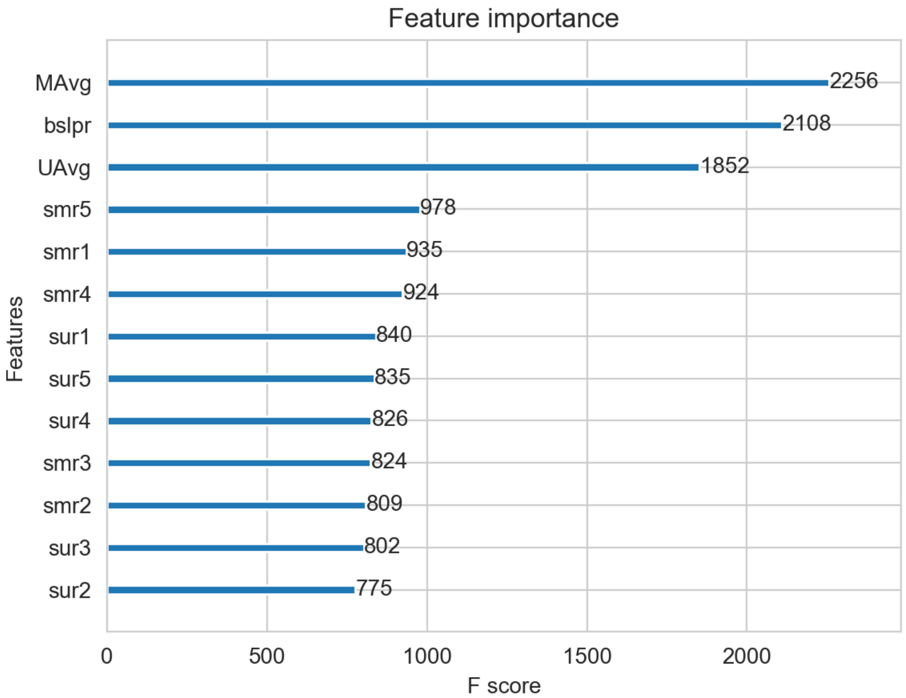
Run Model with Best Hyper Parameters

In [69]:

```
# {'subsample': 1,
#  'n_estimators': 1000,
#  'max_depth': 4,
#  'learning_rate': 0.1,
#  'colsample_bytree': 0.3
```

```
# }

# initialize Our first XGBoost model...
xgb_bsl = xgb.XGBRegressor(silent=False, n_jobs=13, random_state=15, subsample=1, n_estimators=1000
, max_depth=4, learning_rate=0.1, colsample_bytree=0.3)
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_xgboost['xgb_bsl'] = train_results
models_evaluation_test_xgboost['xgb_bsl'] = test_results

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

```
Training the model..
[18:40:47] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
Done. Time taken : 0:08:11.557807

Done

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

TEST DATA
------------------------------
RMSE :  1.112101467112806
MAPE :  33.37429345956948
```



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

Prepare Train Data

In [74]:

```
# prepare Train data
x_train = reg_train[['GAvg', 'sur1', 'sur2', 'sur3', 'sur4', 'sur5', 'smr1', 'smr2', 'smr3', 'smr4'
, 'smr5', 'UAvg', 'MAvg', 'bslpr', 'knn_bsl_u', 'knn_bsl_m']]
#.drop(['user','movie','rating'], axis=1)
y_train = reg_train['rating']
```

Prepare Test Data

Prepare Test Data

In [75]:

```
# Prepare Test data
x_test = reg_test_df[['GAvg', 'sur1', 'sur2', 'sur3', 'sur4', 'sur5', 'smr1', 'smr2', 'smr3',
'smr4', 'smr5', 'UAvg', 'MAvg', 'bslpr', 'knn_bsl_u', 'knn_bsl_m']]
#.drop(['user','movie','rating'], axis=1)
y_test = reg_test_df['rating']
```

In [76]:

```
x_train.columns, x_test.columns
```

Out[76]:

```
(Index(['GAvg', 'sur1', 'sur2', 'sur3', 'sur4', 'sur5', 'smr1', 'smr2', 'smr3',
        'smr4', 'smr5', 'UAvg', 'MAvg', 'bslpr', 'knn_bsl_u', 'knn_bsl_m'],
       dtype='object'),
 Index(['GAvg', 'sur1', 'sur2', 'sur3', 'sur4', 'sur5', 'smr1', 'smr2', 'smr3',
        'smr4', 'smr5', 'UAvg', 'MAvg', 'bslpr', 'knn_bsl_u', 'knn_bsl_m'],
       dtype='object'))
```

Hyper Parameter Tuning

In [77]:

```
from sklearn.model_selection import RandomizedSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
#  class xgboost.XGBRegressor(max_depth=3, learning_rate=0.1, n_estimators=100, verbosity=1,
# objective='reg:squarederror', booster='gbtree', tree_method='auto',
# n_jobs=1, gamma=0, min_child_weight=1, max_delta_step=0, subsample=1,
# colsample_bytree=1, colsample_bylevel=1, colsample_bynode=1,
# reg_alpha=0, reg_lambda=1, scale_pos_weight=1, base_score=0.5,
# random_state=0, missing=None, num_parallel_tree=1, importance_type='gain', **kwargs)
xgb_random = xgb.XGBRegressor(n_jobs=-1, random_state=15)
# xgb_random = xgb.XGBRegressor(nthread=-1, objective='reg:linear', missing=None, seed=8)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
xgb_random = RandomizedSearchCV(estimator = xgb_random, param_distributions = random_grid, n_iter =
100, cv = 3, verbose=3, random_state=15, n_jobs = -1)
# Fit the random search model
xgb_random.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  24 tasks      | elapsed: 60.3min
[Parallel(n_jobs=-1)]: Done 120 tasks      | elapsed: 448.9min
[Parallel(n_jobs=-1)]: Done 280 tasks      | elapsed: 766.6min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 792.6min finished
```

```
[08:09:26] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

Out[77]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
                   estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                          colsample_bylevel=1,
                                          colsample_bynode=1,
                                          colsample_bytree=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=100,
                                          n_jobs=-1, nthread=None,
                                          objective='reg:linear',
                                          random_s...
                                          seed=None, silent=None, subsample=1,
```

```
                                               verbosity=1),
                       iid='warn', n_iter=100, n_jobs=-1,
                       param_distributions={'colsample_bytree': [0.1, 0.3, 0.5, 1],
                                            'learning_rate': [0.01, 0.03, 0.05, 0.1,
                                                              0.15, 0.2],
                                            'max_depth': [2, 3, 4, 5],
                                            'n_estimators': [100, 200, 500, 1000,
                                                             2000],
                                            'subsample': [0.1, 0.3, 0.5, 1]},
                       pre_dispatch='2*n_jobs', random_state=15, refit=True,
                       return_train_score=False, scoring=None, verbose=3)
```

In [78]:

```
# select best params
xgb_random.best_params_
```

Out[78]:

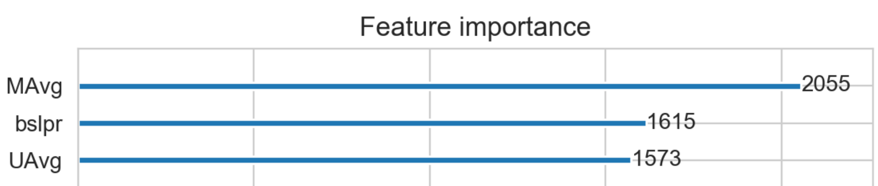
```
{'subsample': 1,
 'n_estimators': 1000,
 'max_depth': 4,
 'learning_rate': 0.1,
 'colsample_bytree': 0.3}
```

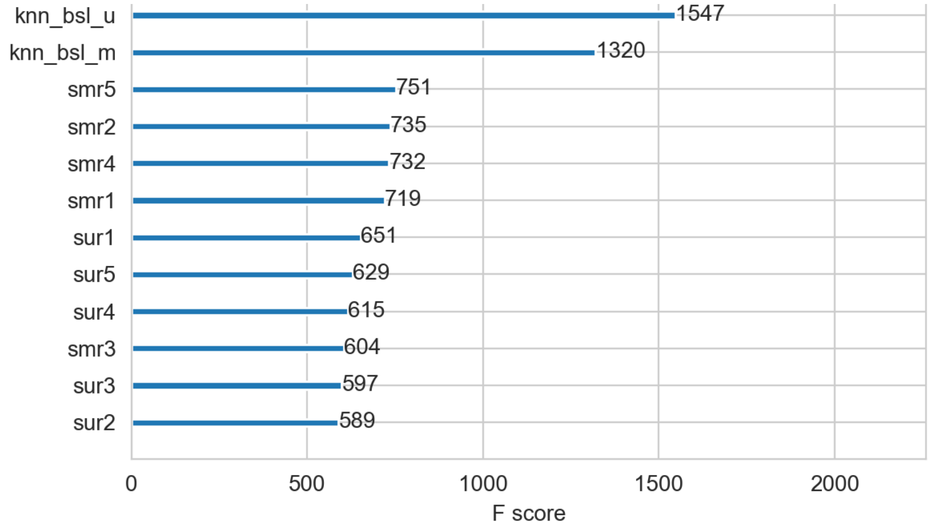
Run Model with Best Hyper Parameters

In [79]:

```
# {'subsample': 1,
#  'n_estimators': 1000,
#  'max_depth': 4,
#  'learning_rate': 0.1,
#  'colsample_bytree': 0.3}


# declare the model
xgb_knn_bsl = xgb.XGBRegressor(silent=False, n_jobs=13, random_state=15, subsample=1, n_estimators=
1000, max_depth=4, learning_rate=0.1, colsample_bytree=0.3)
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_xgboost['xgb_knn_bsl'] = train_results
models_evaluation_test_xgboost['xgb_knn_bsl'] = test_results


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

```
Training the model..
[08:27:20] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
Done. Time taken : 0:08:30.929013

Done

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

TEST DATA
------------------------------
RMSE :  1.1150845388569721
MAPE :  33.33288785296135
```





5.2.4 XgBoost with 13 features + Surprise Baseline + Surprise KNNbaseline + MF Techniques

Prepare Train Data

In [92]:

```
# prepare Train data
x_train = reg_train[['GAvg', 'sur1', 'sur2', 'sur3', 'sur4', 'sur5', 'smr1', 'smr2', 'smr3', 'smr4'
, 'smr5', 'UAvg', 'MAvg', 'bslpr', 'knn_bsl_u', 'knn_bsl_m', 'svd', 'svdpp']]
#.drop(['user','movie','rating'], axis=1)
y_train = reg_train['rating']
```

Prepare Test Data

In [93]:

```
# Prepare Test data
x_test = reg_test_df[['GAvg', 'sur1', 'sur2', 'sur3', 'sur4', 'sur5', 'smr1', 'smr2', 'smr3',
'smr4', 'smr5', 'UAvg', 'MAvg', 'bslpr', 'knn_bsl_u', 'knn_bsl_m', 'svd', 'svdpp']]
#.drop(['user','movie','rating'], axis=1)
y_test = reg_test_df['rating']
```

In [94]:

```
x_train.columns, x_test.columns
```

Out[94]:

```
(Index(['GAvg', 'sur1', 'sur2', 'sur3', 'sur4', 'sur5', 'smr1', 'smr2', 'smr3',
        'smr4', 'smr5', 'UAvg', 'MAvg', 'bslpr', 'knn_bsl_u', 'knn_bsl_m',
        'svd', 'svdpp'],
       dtype='object'),
 Index(['GAvg', 'sur1', 'sur2', 'sur3', 'sur4', 'sur5', 'smr1', 'smr2', 'smr3',
        'smr4', 'smr5', 'UAvg', 'MAvg', 'bslpr', 'knn_bsl_u', 'knn_bsl_m',
        'svd', 'svdpp'],
       dtype='object'))
```

Hyper Parameter Tuning

In [84]:

```
from sklearn.model_selection import RandomizedSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
#  class xgboost.XGBRegressor(max_depth=3, learning_rate=0.1, n_estimators=100, verbosity=1,
# objective='reg:squarederror', booster='gbtree', tree_method='auto',
# n_jobs=1, gamma=0, min_child_weight=1, max_delta_step=0, subsample=1,
# colsample_bytree=1, colsample_bylevel=1, colsample_bynode=1
```

```
# colsample_bytree=1, colsample_bylevel=1, colsample_bynode=1,
# reg_alpha=0, reg_lambda=1, scale_pos_weight=1, base_score=0.5,
# random_state=0, missing=None, num_parallel_tree=1, importance_type='gain', **kwargs)
xgb_random = xgb.XGBRegressor(n_jobs=-1, random_state=15)
# xgb_random = xgb.XGBRegressor(nthread=-1, objective='reg:linear', missing=None, seed=8)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
xgb_random = RandomizedSearchCV(estimator = xgb_random, param_distributions = random_grid, n_iter =
100, cv = 3, verbose=3, random_state=15, n_jobs = -1)
# Fit the random search model
xgb_random.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  24 tasks      | elapsed: 53.4min
[Parallel(n_jobs=-1)]: Done 120 tasks      | elapsed: 299.2min
[Parallel(n_jobs=-1)]: Done 280 tasks      | elapsed: 675.3min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 704.1min finished
```

```
[20:35:02] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

Out[84]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
                   estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                          colsample_bylevel=1,
                                          colsample_bynode=1,
                                          colsample_bytree=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=100,
                                          n_jobs=-1, nthread=None,
                                          objective='reg:linear',
                                          random_s...
                                          seed=None, silent=None, subsample=1,
                                          verbosity=1),
                   iid='warn', n_iter=100, n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.3, 0.5, 1],
                                        'learning_rate': [0.01, 0.03, 0.05, 0.1,
                                                          0.15, 0.2],
                                        'max_depth': [2, 3, 4, 5],
                                        'n_estimators': [100, 200, 500, 1000,
                                                         2000],
                                        'subsample': [0.1, 0.3, 0.5, 1]},
                   pre_dispatch='2*n_jobs', random_state=15, refit=True,
                   return_train_score=False, scoring=None, verbose=3)
```

In [85]:

```
# select best params
xgb_random.best_params_
```

Out[85]:

```
{'subsample': 1,
 'n_estimators': 2000,
 'max_depth': 4,
 'learning_rate': 0.01,
 'colsample_bytree': 0.3}
```

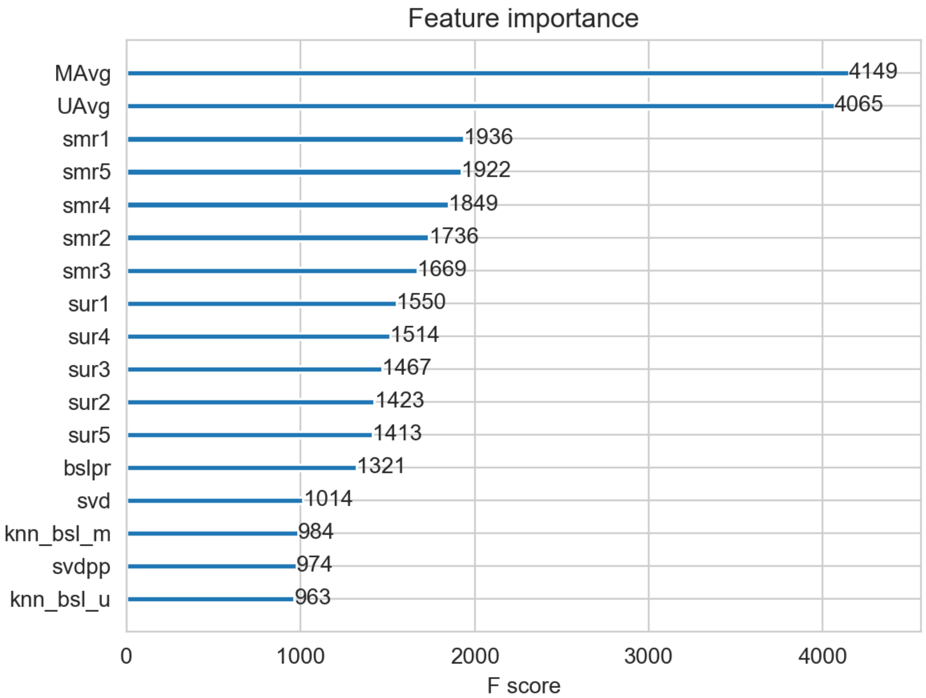
Run Model with Best Hyper Parameters

In [95]:

```
# {'subsample': 1,
#  'n_estimators': 2000,
#  'max_depth': 4,
#  'learning_rate': 0.01,
#  'colsample_bytree': 0.3}
```

```
xgb_final = xgb.XGBRegressor(silent=False, n_jobs=13, random_state=15, subsample=1, n_estimators=20
00, max_depth=4, learning_rate=0.01, colsample_bytree=0.3)
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_xgboost['xgb_final'] = train_results
models_evaluation_test_xgboost['xgb_final'] = test_results


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

```
Training the model..
[04:13:24] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
Done. Time taken : 0:21:28.896083

Done

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

TEST DATA
------------------------------
RMSE :  1.1108547023155746
MAPE :  33.38595904951013
```



5.2.5 XgBoost with Surprise Baseline + Surprise KNNbaseline + MF Techniques

Prepare Train Data

In [86]:

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

Prepare Test Data

In [87]:

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

In [88]:

```
x_train.columns, x_test.columns
```

Out[88]:

```
(Index(['bslpr', 'knn_bsl_u', 'knn_bsl_m', 'svd', 'svdpp'], dtype='object'),
 Index(['bslpr', 'knn_bsl_u', 'knn_bsl_m', 'svd', 'svdpp'], dtype='object'))
```

Hyper Parameter Tuning

In [89]:

```
# # Use the random grid to search for best hyperparameters
# # First create the base model to tune
# xgb_random = xgb.XGBRegressor(nthread=-1, objective='reg:linear', missing=None, seed=8)
# # Random search of parameters, using 3 fold cross validation,
# # search across 100 different combinations, and use all available cores
# xgb_random = RandomizedSearchCV(estimator = xgb_random, param_distributions = random_grid, n_ite
r = 100, cv = 3, verbose=2, random_state=42, n_jobs = -1)
# # Fit the random search model
# xgb_random.fit(x_train, y_train)
# Use the random grid to search for best hyperparameters
# First create the base model to tune
#  class xgboost.XGBRegressor(max_depth=3, learning_rate=0.1, n_estimators=100, verbosity=1,
# objective='reg:squarederror', booster='gbtree', tree_method='auto',
# n_jobs=1, gamma=0, min_child_weight=1, max_delta_step=0, subsample=1,
# colsample_bytree=1, colsample_bylevel=1, colsample_bynode=1,
# reg_alpha=0, reg_lambda=1, scale_pos_weight=1, base_score=0.5,
# random_state=0, missing=None, num_parallel_tree=1, importance_type='gain', **kwargs)
xgb_random = xgb.XGBRegressor(n_jobs=-1, random_state=15)
# xgb_random = xgb.XGBRegressor(nthread=-1, objective='reg:linear', missing=None, seed=8)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
xgb_random = RandomizedSearchCV(estimator = xgb_random, param_distributions = random_grid, n_iter =
100, cv = 3, verbose=3, random_state=15, n_jobs = -1)
# Fit the random search model
xgb_random.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  24 tasks      | elapsed: 26.0min
[Parallel(n_jobs=-1)]: Done 120 tasks      | elapsed: 152.9min
[Parallel(n_jobs=-1)]: Done 280 tasks      | elapsed: 340.1min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 355.7min finished
```

```
[02:51:10] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

Out[89]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
                   estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                          colsample_bylevel=1,
                                          colsample_bynode=1,
                                          colsample_bytree=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=100,
                                          n_jobs=-1, nthread=None,
                                          objective='reg:linear',
                                          random_s...
                                          seed=None, silent=None, subsample=1,
                                          verbosity=1),
                   iid='warn', n_iter=100, n_jobs=-1,
```

```
                 param_distributions={'colsample_bytree': [0.1, 0.3, 0.5, 1],
                                      'learning_rate': [0.01, 0.03, 0.05, 0.1,
                                                        0.15, 0.2],
                                      'max_depth': [2, 3, 4, 5],
                                      'n_estimators': [100, 200, 500, 1000,
                                                       2000],
                                      'subsample': [0.1, 0.3, 0.5, 1]},
                 pre_dispatch='2*n_jobs', random_state=15, refit=True,
                 return_train_score=False, scoring=None, verbose=3)
```

In [90]:

```
# select best params
xgb_random.best_params_
```

Out[90]:

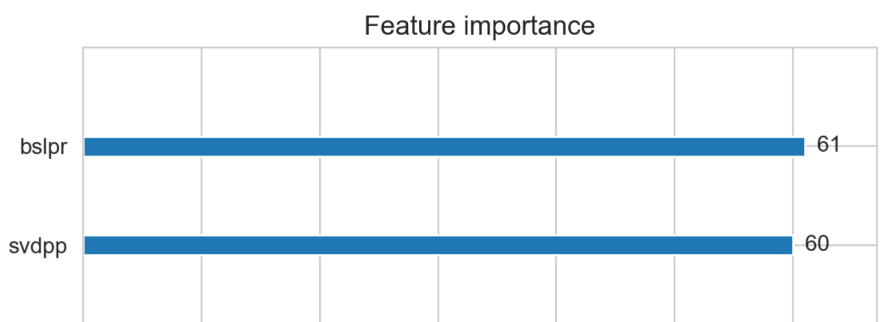
```
{'subsample': 0.5,
 'n_estimators': 100,
 'max_depth': 2,
 'learning_rate': 0.1,
 'colsample_bytree': 0.3}
```

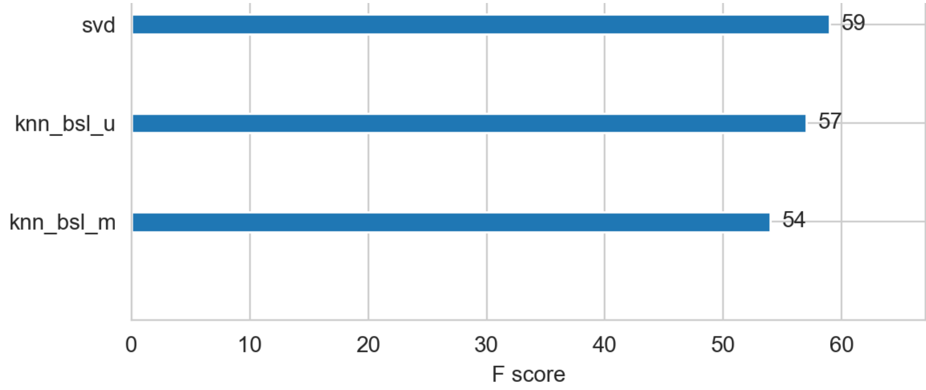
Run Model with Best Hyper Parameters

In [91]:

```
# {'subsample': 0.5,
#  'n_estimators': 100,
#  'max_depth': 2,
#  'learning_rate': 0.1,
#  'colsample_bytree': 0.3}

xgb_all_models = xgb.XGBRegressor(silent=False, n_jobs=13, random_state=15, subsample=0.5, n_estima
tors=100, max_depth=2, learning_rate=0.1, colsample_bytree=0.3)
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_xgboost['xgb_all_models'] = train_results
models_evaluation_test_xgboost['xgb_all_models'] = test_results

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

```
Training the model..
[04:10:17] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
Done. Time taken : 0:00:23.018425

Done

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

TEST DATA
------------------------------
RMSE :  1.0890338848602066
MAPE :  34.54356682744893
```





In [96]:

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

Out[96]:

```
xgb_all_models    1.0890338848602066
xgb_final         1.1108547023155746
xgb_bsl            1.112101467112806
xgb_knn_bsl       1.1150845388569721
first_algo        1.1374496313732387
Name: rmse, dtype: object
```

In [97]:

```
from prettytable import PrettyTable

table = PrettyTable()
table.field_names = ['Model', 'n_estimators', 'max_depth', 'Test Data RMSE', 'Test Data MAPE']

table.add_row(['XGBoost with initial 13 features', 1000, 4, 1.1374496313732387, 32.979426329406394
])
table.add_row(['XGBoost with initial 13 features + Surprise Baseline predictor', 1000, 4,
1.112101467112806, 33.37429345956948])
table.add_row(['XGBoost with initial 13 features + Surprise Baseline predictor + KNNBaseline predi
ctor', 1000, 4, 1.1150845388569721, 33.33288785296135])
table.add_row(['XgBoost with 13 features + Surprise Baseline + Surprise KNNbaseline + MF Technique
s', 2000, 4, 1.1108547023155746, 33.38595904951013])
table.add_row(['XgBoost with Surprise Baseline + Surprise KNNbaseline + MF Techniques', 100, 2, 1.
0890338848602066, 34.54356682744893])

print(table)
```

```
+-----------------------------------------------------------------------------------------+---------
-+------------+--------------------+--------------------+
|                                          Model                                          |
n_estimators | max_depth |    Test Data RMSE    |    Test Data MAPE    |
+-----------------------------------------------------------------------------------------+---------
-+------------+--------------------+--------------------+
|                             XGBoost with initial 13 features                            |     100C
|      4      | 1.1374496313732387 | 32.979426329406394 |
|              XGBoost with initial 13 features + Surprise Baseline predictor              |     100C
|      4      | 1.112101467112806  | 33.37429345956948  |
| XGBoost with initial 13 features + Surprise Baseline predictor + KNNBaseline predictor |     100
0      |      4      | 1.1150845388569721 | 33.33288785296135  |
|  XgBoost with 13 features + Surprise Baseline + Surprise KNNbaseline + MF Techniques  |
2000      |      4      | 1.1108547023155746 | 33.38595904951013  |
|          XgBoost with Surprise Baseline + Surprise KNNbaseline + MF Techniques          |     100
|      2      | 1.0890338848602066 | 34.54356682744893  |
+-----------------------------------------------------------------------------------------+---------
-+------------+--------------------+--------------------+
```

Conclusion

- It took approx. 60hrs to sample train data for 30K users and 3K movies
- It took approx. 20hrs to sample test data for 30K users and 3K movies
- It took approx. 54hrs to tune all 5 xgboost models
- Surprise Library models performs better than xgboost in this scenario even after hyper parameter tuning.
- Multithreading works well in sample train data.

In [ ]:

```
```